Goal: Explore how architecture and testing inform each other

• Architecture support for testing refers to using a system's (software) architecture to inform its testing process.
• There has been substantial research to this topic.
• It is less common in practice.

• Our goal is to help answer this question:

How can we use, or better use, architecture to help test our systems?
Overview

- We seek to establish connections between architecture and testing, in both directions.
  - In one direction (from testing to architecture) the goal is to understand how, given an architecture, testers can use it to make testing more effective.
  - In the other direction (from architecture to testing), the goal is to design an architecture up front with the goal to make testing more effective.
In practice, aren’t architecture and testing connected already?

- Well, yes.
  - Unit testing could not go on without a definition of what the units are, and in an architecture-centric development culture those units are defined by an architecture’s module decomposition view.
  - Integration testing would be meaningless without the architecturally defined units that are being integrated.
- But: There are more sophisticated ways in which architecture can inform testing (and vice versa) that are not currently being routinely utilized.
The payoff

- Testing is very expensive.
- Data shows that 30% to 90% of a project’s schedule and budget are often devoted to testing.
- In many systems, testing is used to save lives and protect property.
- Even a small improvement will make a very big difference.
Overview of architecture support for testing

1. Architecture-based analysis
   - Use an architecture or architectural model to make predictions about a system, generally absent any implementation.
   - In architecture-based testing, we concentrate on Architecture analysis that can be used to improve testing outcomes in some way, for example, reducing the cost of testing.

1. Generation of code-targeted test artifacts
   - Use architecture to produce artifacts (e.g., test cases, test plans, coverage measures) useful in testing an implementation of a system.

1. Designing for testability
   - Techniques for designing a system so that it is more testable – that is, more likely to reveal a fault when tested.

1. Testing conformance of implementation to architecture
   - Compare an implementation to the architecture that it is supposed to follow, to see if it complies.
Overview of architecture support for testing

- Architecture-based predictive analysis
- Architecture-based generation of code-targeted test artifacts
- Designing for testability
- Testing conformance of implementation to architecture
Designing for testability: Our work

- We conducted an extensive literature review

- Three main strategies emerged
  1. Choose styles or patterns that are highly testable
  2. Choose tactics to increase testability
  3. Design so as to keep certain testability-affecting metrics low

- A full report is forthcoming.
Our main work: “Testability profiles”

- We call an architectural style, pattern, or tactic an *architectural design approach (ADA)*.
- Each ADA can have *testability profile* associated with it.
- A testability profile has four parts.
  1. A fault model associated with the ADA
     - Positive faults: Faults you might see when you choose this ADA
     - Negative faults: Faults that the choice of this ADA rules out
  1. A set of “observables” associated with the ADA that show up when the ADA is chosen.
  1. A set of analyses associated with the ADA (for example, deadlock detection, state machine correctness, presence or lack of “illegal” communication among components in the architecture, etc.)
  1. A set of tests that the negative faults and the analysis can replace.
Using a testability profile

1. A tester can use the observables to see if a particular claimed ADA is present in the code.
2. A tester can use the negative faults in the fault model to de-emphasize (or even omit) the corresponding tests.
3. A tester can use analysis in the testability profile to de-emphasize (or even omit) the corresponding tests.
4. A tester can use the positive faults in the fault model (not covered by analysis) to create tests.

The result should be a more productive, more focused test suite. Faults that cannot occur are de-emphasized in testing. Architecture-based faults that can occur can receive increased attention.
The basic idea

1. Analyze architecture for these faults
2. Eliminate these tests
3. Eliminate these tests

Full test suite

Tests for faults that cannot occur
Tests made redundant by analysis
The basic idea: Example

1. Analyze architecture for these faults
2. Eliminate these tests
3. Eliminate these tests

Java Path Finder (JPF)

Full test suite

Tests made redundant by analysis

Tests for faults that cannot occur

Faults that can occur

Faults that cannot occur

Available analysis

Tests made redundant

Available analysis

Missing state. Missing transition, Unreachable state, Transition never taken, etc.

No errors in a system’s control logic can occur outside the state machine.
Bringing research and practice together

- A pair of workshops
  - Practitioner’s workshop, Pittsburgh, February: Posing problems
  - Researcher’s workshop, Pisa, March: Offering solutions
Practitioner’s Workshop

- February, 2011, Pittsburgh
- Attended by 14 invited “architecture-aware” testing practitioners from U.S. Government, industry, and research
- Results was a set of 29 *model problems*
  - A model problem is a problem that, if solved, would result in a significant improvement over current testing practice.
Participants

- Antonia Bertolino, Researcher, Italian National Research Council; Leader of Software Engineering Research Laboratory at Istituto di Scienza e Tecnologie dell'Informazione A. Faedo in Pisa
- Michael Caughey, Enterprise Performance Test Manager, Capital One
- Rik Drummond, CEO, Drummond Group, Inc.
- Ronny Kolb, Global software architect, Honeywell International
- Jenny Li, Research scientist, Avaya Labs
- Randy Lyvers, Director - Core II Controls, Cummins, Inc.
- David McClung, Technical Director, DB-0855-IV, Whitfill Central Technical Support Facility (CTSF), Ft Hood, Texas
- Henry Muccini, Assistant Professor, Universita' degli Studi dell'Aquila
- Don O'Connell, Technical Fellow, The Boeing Company
- Dan Showalter, Embedded Software Test Architect, Cummins Inc.
- Chris Walter, President, WW Technology Group
- Tom Wissink, Corporate Integration, Test & Evaluation Director, Lockheed Martin
- Peter Zimmerer, Principal engineer, Siemens AG Corporate Technology
Ranking the model problems

- After the workshop, participants ranked the model problems:
  - VH (Very High) = 5 (meaning that the respondent places a very high value on this capability)
  - H (High) = 4
  - M (Medium) = 3
  - L (Low) = 2
  - VL (Very Low) = 1 (meaning that this is a capability that is not at all valuable to the respondent)
### Top Three Model Problems (average rank ≥ 4.00)

<table>
<thead>
<tr>
<th>4.30</th>
<th>Scenario</th>
<th>REQ1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Tester</td>
<td></td>
</tr>
<tr>
<td>Stimulus</td>
<td>A tester chooses a test set to test the system for requirements satisfaction.</td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>Architecture is complete. System test not yet begun.</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>The tester uses an architecture analysis tool that identifies the smallest number of tests to run to provide coverage of 98% of the requirements. Redundant tests are eliminated.</td>
<td></td>
</tr>
<tr>
<td>Response measure</td>
<td>Performing the analysis is much less costly and time-consuming to run than the tests it replaces.</td>
<td></td>
</tr>
<tr>
<td>Original notes from workshop</td>
<td>Can we use the architecture to tell us that (out of all of the huge number of tests possible) if we run a small number of tests, we will have covered 98% of the requirements?</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td>REQ2</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>Tester</td>
<td></td>
</tr>
<tr>
<td>Stimulus</td>
<td>A tester chooses a test set.</td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>The architecture is complete.</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>The tester uses an architecture analysis tool that identifies the smallest number of tests to run to provide coverage of the highest-risk areas of the system. Redundant tests are eliminated.</td>
<td></td>
</tr>
<tr>
<td>Response measure</td>
<td>Performing the analysis is much less costly and time-consuming to run than the tests it replaces.</td>
<td></td>
</tr>
<tr>
<td>Original notes from workshop</td>
<td>Can I use architecture to use tests to cover high-risk or high-probability-of-error areas in the system?</td>
<td></td>
</tr>
</tbody>
</table>

- Design for testability of the architecture to meet *those* requirements (encapsulation, isolation, …)
- Analyze for satisfaction of *those* req’s., generate test artifacts to test for *those* req’s.
- Conformance is necessary for this to work.
<table>
<thead>
<tr>
<th>Scenario</th>
<th>INT15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Tester</td>
</tr>
<tr>
<td>Stimulus</td>
<td>The tester needs to develop integration tests</td>
</tr>
<tr>
<td>Environment</td>
<td>Architecture of the system is complete and detailed</td>
</tr>
<tr>
<td>Response</td>
<td>Automated tools use the architecture to generate test cases and test subs for the system’s components</td>
</tr>
<tr>
<td>Response measure</td>
<td>The tester asserts that 90% of the generated tests should be run</td>
</tr>
</tbody>
</table>
| Original notes from workshop | How can architecture be used to drive integration testing?  
  • Generation of test stubs from high level descriptions (Common Information Model CIM)  
  • Generation of test cases from more detailed descriptions (Multispeak)  
  • Keyword-based test automation. Can we use the domain specific language in the architecture models so that we can test sequences of the keywords? |
## Scenario SPL1

<table>
<thead>
<tr>
<th>Source</th>
<th>A stakeholder who wants a change to product B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>A change request, which will result in core assets being modified, is initiated for product B, which is a certified product.</td>
</tr>
<tr>
<td>Environment</td>
<td>The products in the product line must be certified to a specified standard. Core assets have been used as part of the product implementation. Several products have used these core assets.</td>
</tr>
<tr>
<td>Response</td>
<td>An architecture-based analysis proves that no recertification of A, which only has new implementations of the same core assets, is needed. Evidence accumulated during the changes is used to expedite the recertification of B.</td>
</tr>
<tr>
<td>Response measure</td>
<td>The effort to recertify B is significantly less than the effort to certify B originally while maintaining the level of confidence.</td>
</tr>
</tbody>
</table>
| Original notes from workshop | • Change Core or Variant Assets for next Product B without affecting certification of Product A  
• What does it take to certify a parent product having capability and then apply certification to other products? |
Onward!

• We are pleased to be part of this international effort.

• We believe this work has the potential to make a significant difference in software engineering.

• We look forward to the results of this workshop and future work!
Questions? Now or later...

Paul Clements
clements@sei.cmu.edu

U.S. Mail:
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213

World Wide Web:
http://www.sei.cmu.edu/architecture

SEI Fax:
412-268-5758
NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.