The Theory and Practice of Software Testing: 
Can we Test it? Yes we Can!

Gregory M. Kapfhammer†
Department of Computer Science
Allegheny College, Pennsylvania, USA
http://www.cs.allegheny.edu/~gkapfham/

SGT Global, February 2008

† In Conjunction with Mary Lou Soffa, Kristen Walcott (UVa/CS)
Suvarshi Bhadra, Joshua Geiger, Adam Smith, Gavilan Steinman, Yuting Zhang (Allegheny/CS)

Featuring images from Embroidery and Tapestry Weaving, Grace Christie (Project Gutenberg)
Presentation Outline

1. Software Testing Challenges
2. Structural Testing
3. Regression Testing
4. Mutation Testing
5. Future Work
6. Conclusion
The Challenge of Software Testing

I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked “a relevant state” and to convince oneself that the testing programs generate them all is no simple matter. The encouraging thing is that (as far as we know!) it could be done.


**Important Question**: What are your software development and testing challenges? What are your best solutions?
Modern Software is Complex

- Complex source code, database, files, and network communication
- Can we **increase** reliability by **simplifying** software?
Defect Locations

Defects may exist in the individual **components** or the **interactions**
Testing isolates defects and establishes a confidence in the correctness of a software application.
Presentation Outline

1. Software Testing Challenges
2. Structural Testing
3. Regression Testing
4. Mutation Testing
5. Future Work
6. Conclusion

The Theory and Practice of Software Testing: Can we Test it? Yes we Can!
What is a Test Case?

Overview

- Test suite executor (JUnit) runs each test case independently
- Each test invokes a method within the program and then compares the actual and expected output values
Overview

- Structural **adequacy criteria** focus on the coverage of nodes, edges, paths, and definition-use associations
- Instrumentation **probes** track the coverage of test requirements
Calculating the Coverage of a Test

Calculating Coverage
Use instrumentation probes to **capture** and **analyze** a test suite’s coverage of the program

Regression Testing
The adequacy measurements can be used to support both test suite **reduction** and **prioritization**
1. Software Testing Challenges
2. Structural Testing
3. Regression Testing
4. Mutation Testing
5. Future Work
6. Conclusion

The Theory and Practice of Software Testing: Can we Test it? Yes we Can!
Finding the Overlap in Coverage

- $R_j \rightarrow T_i$ means that requirement $R_j$ is **covered by** test $T_i$
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements
- May include the **remaining** tests so that they can **redundantly** cover the requirements
Reducing and Prioritizing the Tests

Regression Testing Overview

**Reduction** creates a smaller test suite that covers the same requirements as the original suite. **Prioritization** re-orders the tests so that they cover the requirements more effectively. Techniques use **heuristics** to solve NP-complete problems.
Evaluating a Test Prioritization

Prioritize to **increase** the CE of a test suite $CE = \frac{Actual}{Ideal} \in [0, 1]$
## Characterizing a Test Suite

### Test Information

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Cost (sec)</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>5</td>
<td>$R_1$ $\checkmark$ $R_2$ $\checkmark$</td>
</tr>
<tr>
<td>$T_2$</td>
<td>10</td>
<td>$R_1$ $\checkmark$ $R_2$ $\checkmark$ $R_3$ $\checkmark$</td>
</tr>
<tr>
<td>$T_3$</td>
<td>4</td>
<td>$R_1$ $\checkmark$ $R_3$ $\checkmark$ $R_5$ $\checkmark$</td>
</tr>
</tbody>
</table>

Total Testing Time = 19 seconds

### Formulating the Metrics

$CE$ considers the **execution time** of each test while $CE_u$ assumes that all test cases execute for a **unit cost**
# Coverage Effectiveness Values

## Calculating $CE$ and $CE_u$

<table>
<thead>
<tr>
<th>Ordering</th>
<th>CE</th>
<th>$CE_u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1 \ T_2 \ T_3$</td>
<td>.3789</td>
<td>.4</td>
</tr>
<tr>
<td>$T_1 \ T_3 \ T_2$</td>
<td>.5053</td>
<td>.4</td>
</tr>
<tr>
<td>$T_2 \ T_1 \ T_3$</td>
<td>.3789</td>
<td>.5333</td>
</tr>
<tr>
<td>$T_2 \ T_3 \ T_1$</td>
<td>.4316</td>
<td>.6</td>
</tr>
<tr>
<td>$T_3 \ T_1 \ T_2$</td>
<td>.5789</td>
<td>.4557</td>
</tr>
<tr>
<td>$T_3 \ T_2 \ T_1$</td>
<td>.5789</td>
<td>.5333</td>
</tr>
</tbody>
</table>

## Observations

- Including test case costs does impact the $CE$ metric
- Depending upon the characteristics of the test suite, we may see $CE = CE_u$, $CE > CE_u$, or $CE < CE_u$
The Theory and Practice of Software Testing: Can we Test it? Yes we Can!
Mutation Testing Techniques

**Mutant Creation**

- A mutation testing tool (e.g., $\mu$Java or Jumble) inserts defects into the program under test.
- **Question**: Why are we inserting faults into the programs that we are testing?

**Test Quality**

**Goal**: measure the quality of the test suite by determining whether or not it can differentiate between faulty and non-faulty programs.
**Fault Seeding**
- Use **known faults** or a **mutation testing** tool (e.g., $\mu$Java or Jumble) to insert defects into the program.
- Determine which test(s) are able to detect the seeded faults and construct a **fault table**.

**APFD Calculation**
A test ordering has a higher APFD score if it **rapidly** detects the faults.
Use **heuristic search** (HC, SANN, GA) to prioritize the test suite
Detailed Empirical Evaluations

- New Testing Techniques
- Real World Programs

**Systematically** study the **efficiency** and **effectiveness** trade-offs of different software testing techniques.
Conclusions

Concluding Remarks

- Software development and testing is fun and exciting!
- There are many new developments in research and practical tools – some of which are ready for use today!
- What are your favorite software testing tools and techniques?

Resources

- Conferences: ICSE, FSE, ISSTA, ASE, ICSM, ISSRE
- Journals: TSE, TOSEM, IST, JSS, JSME
- Many articles are available online from Google Scholar
- http://www.cs.allegheny.edu/~gkapfham/