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/

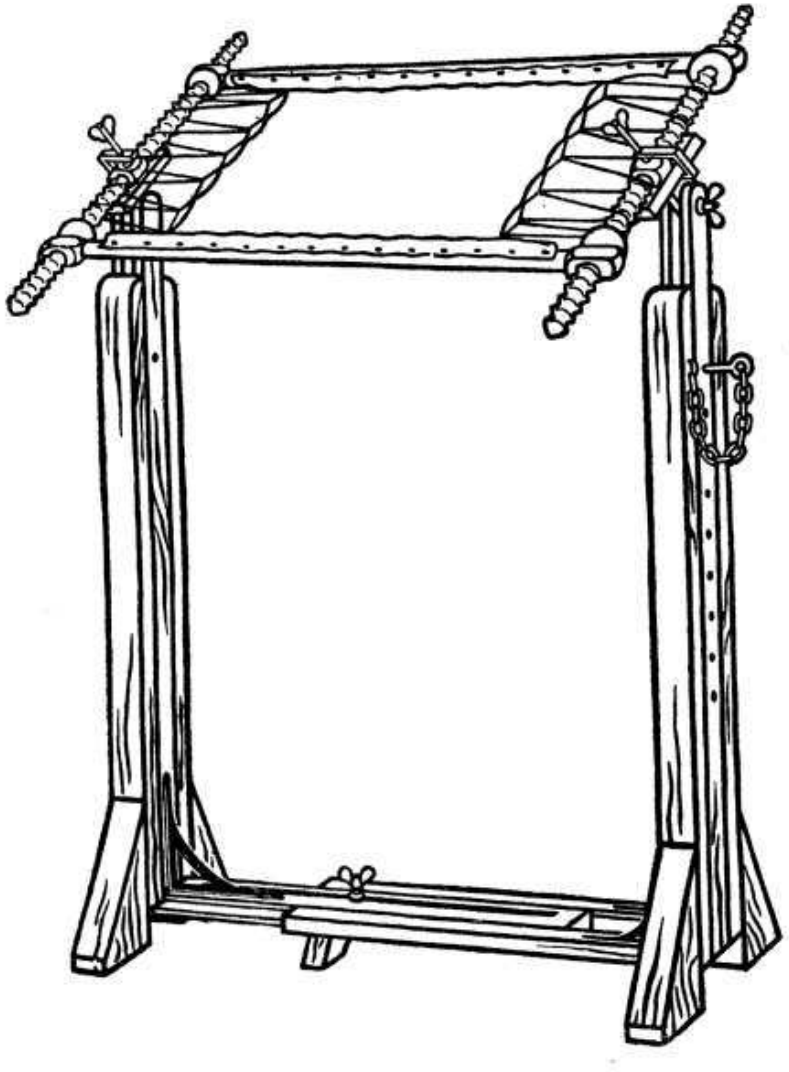
SGT Global, February 2008

† 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

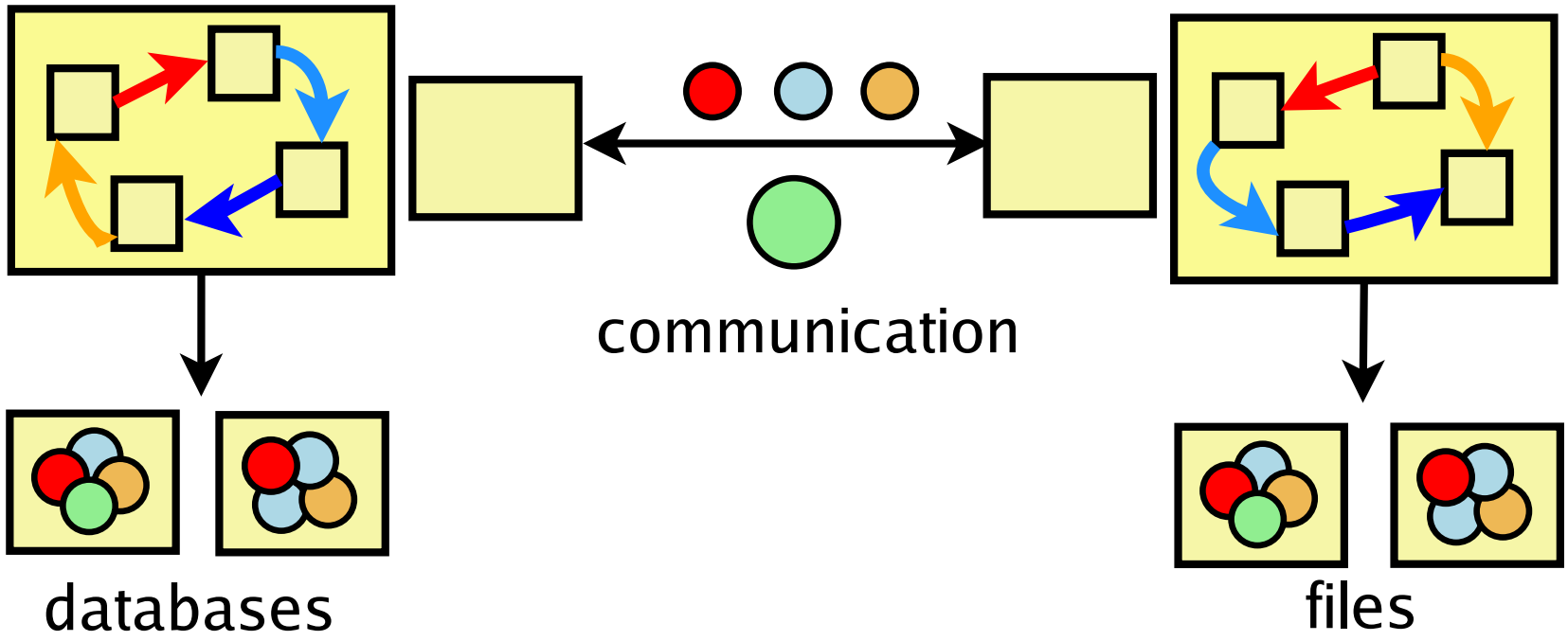


The Challenge of Software Testing

> 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.
>
> Edsger W. Dijkstra, *Communications of the ACM,* 1968

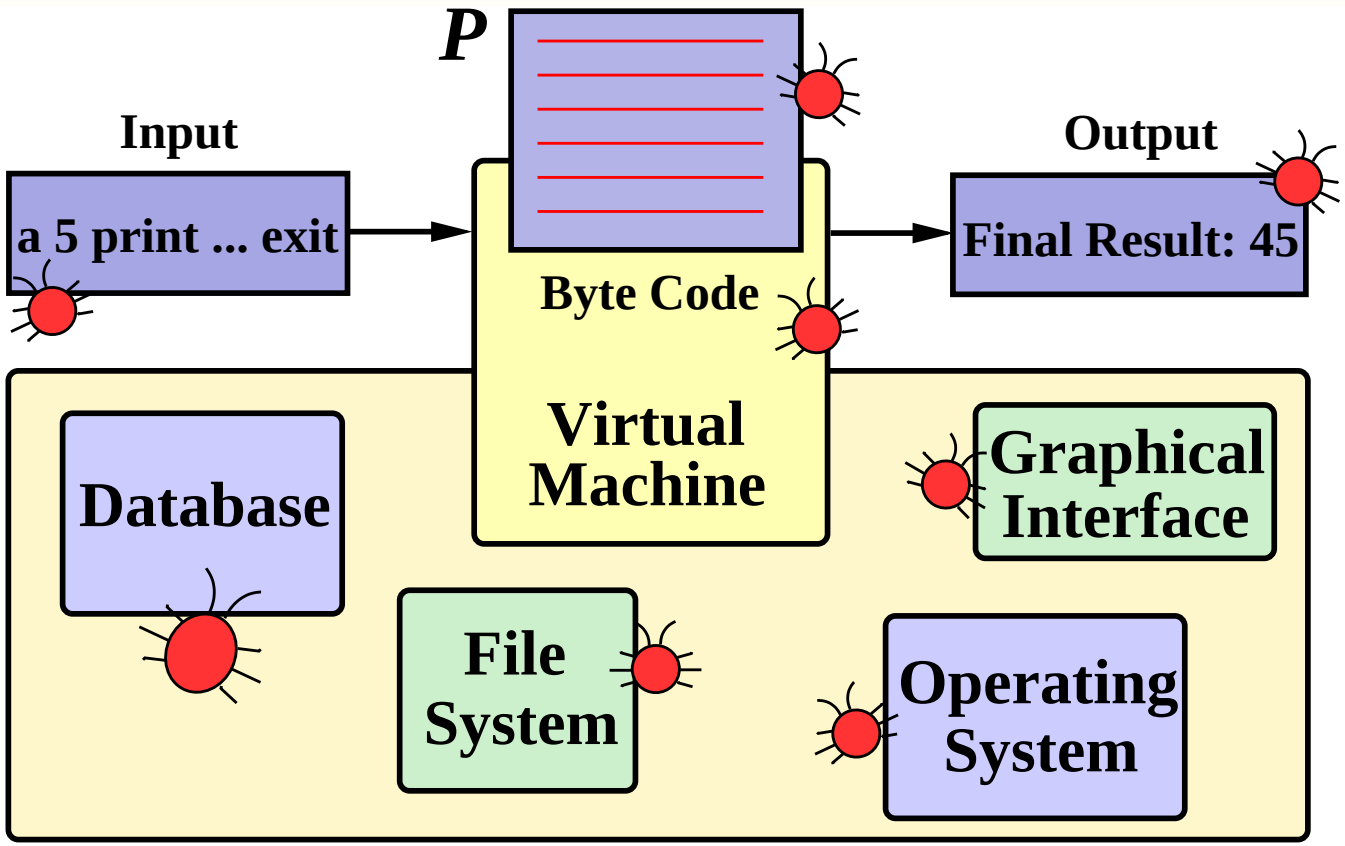
Important Question: What are your software development and testing **challenges**? What are your best **solutions**?

Modern Software is Complex



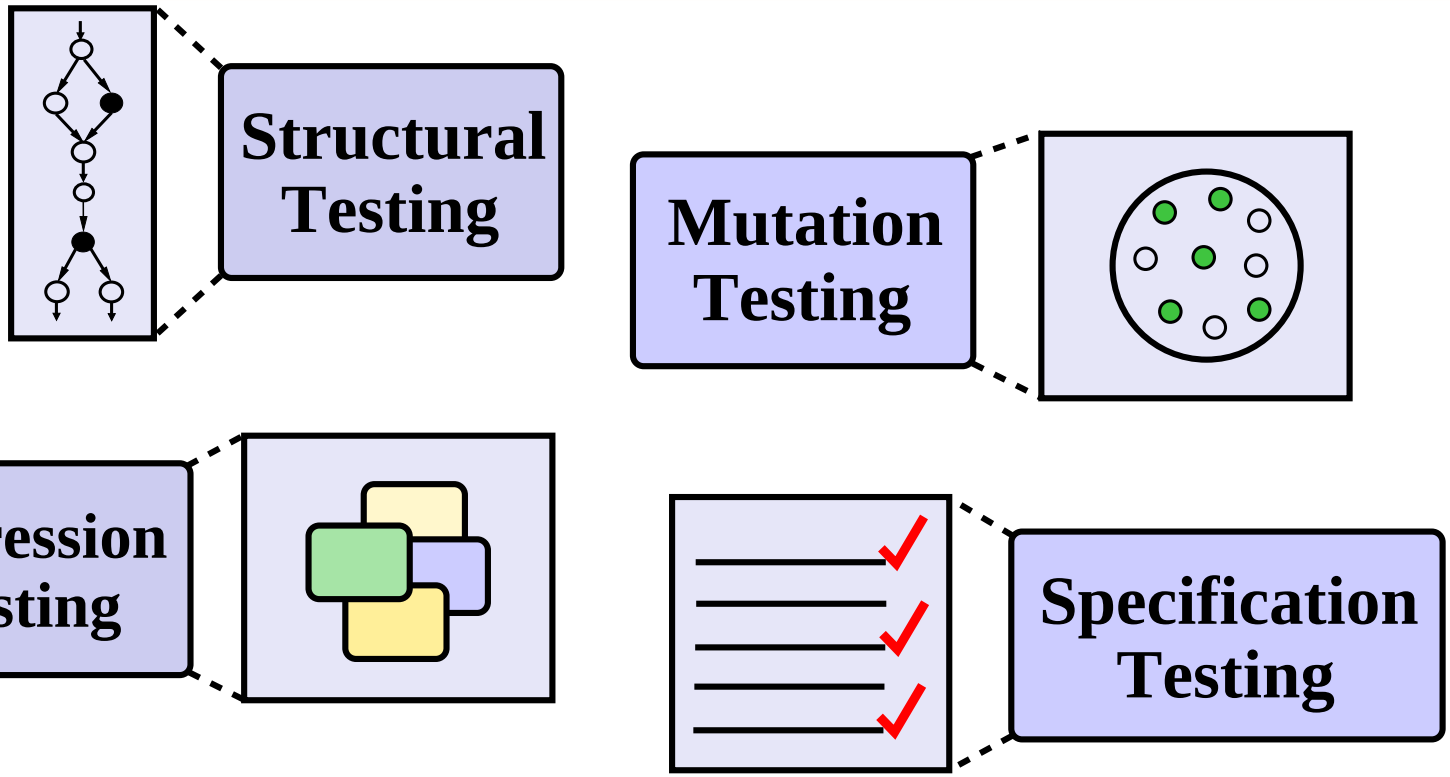
- Complex source code, database, files, and network communication
- Can we **increase** reliability by **simplifying** software?

Defect Locations



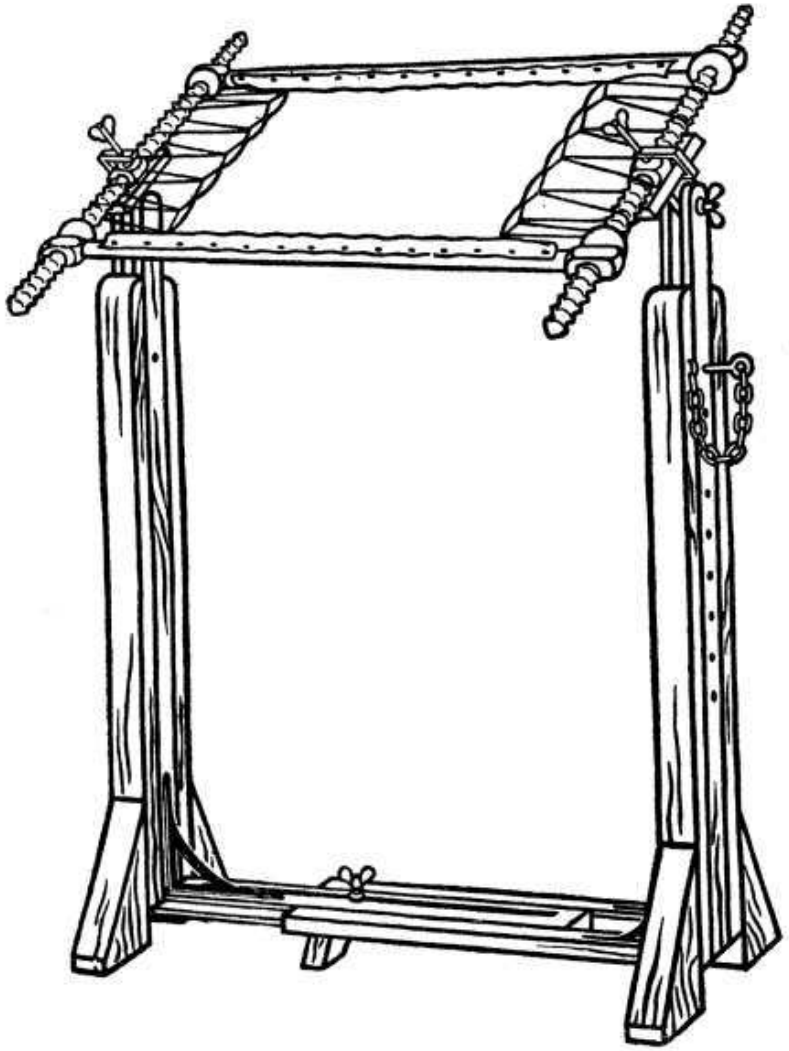
Defects may exist in the individual **components** or the **interactions**

Approaches to Software Testing

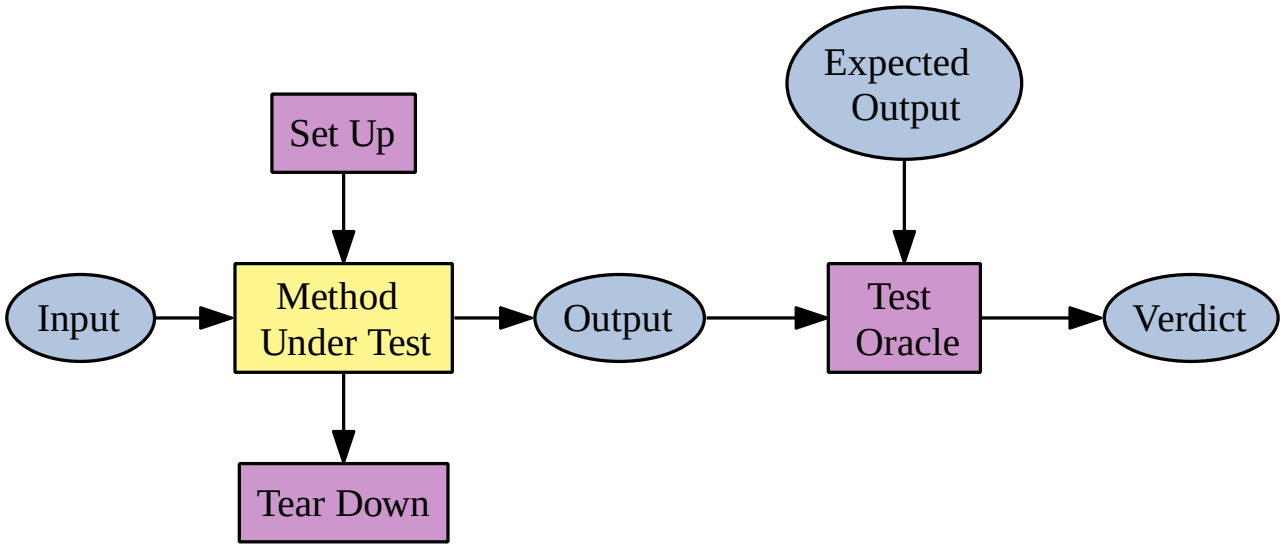


Testing **isolates defects** and establishes a **confidence in the correctness** of a software application

Presentation Outline



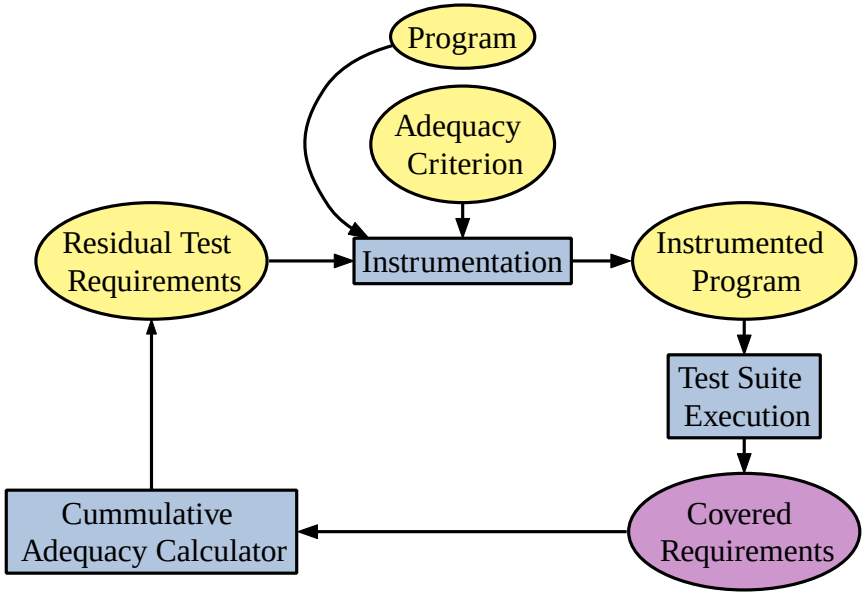
What is a Test Case?



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

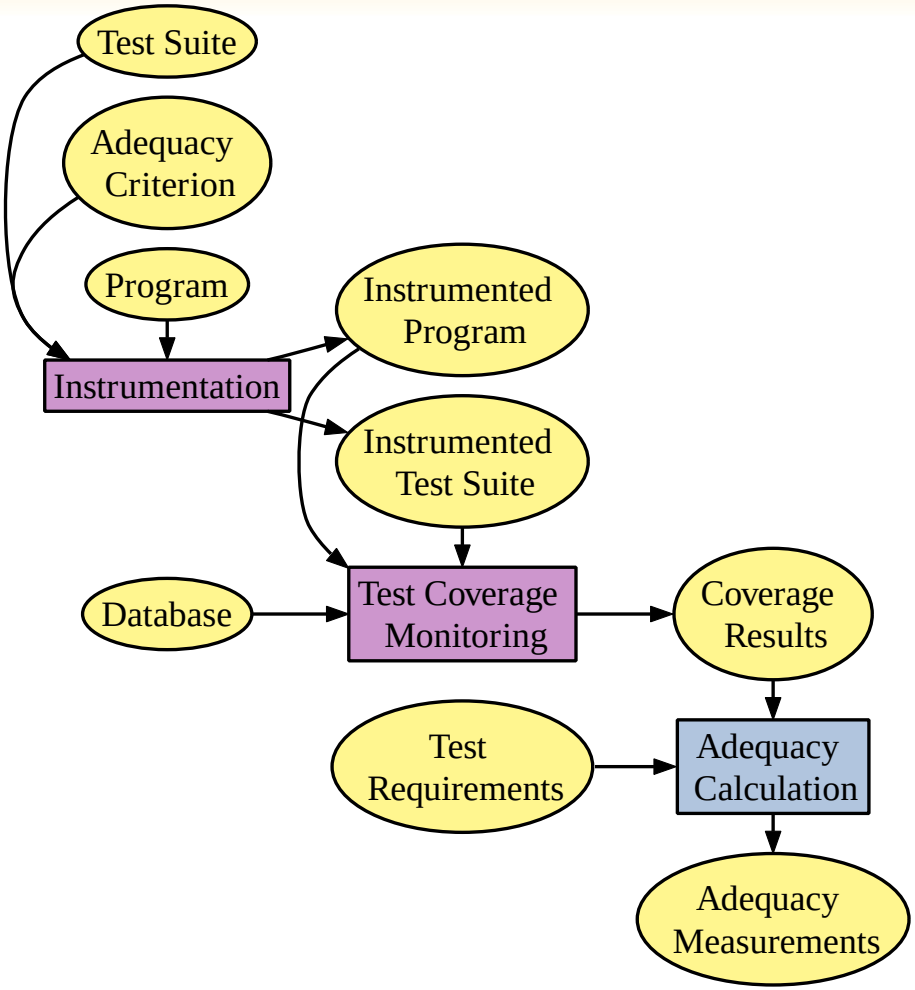
Test Coverage Monitoring



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



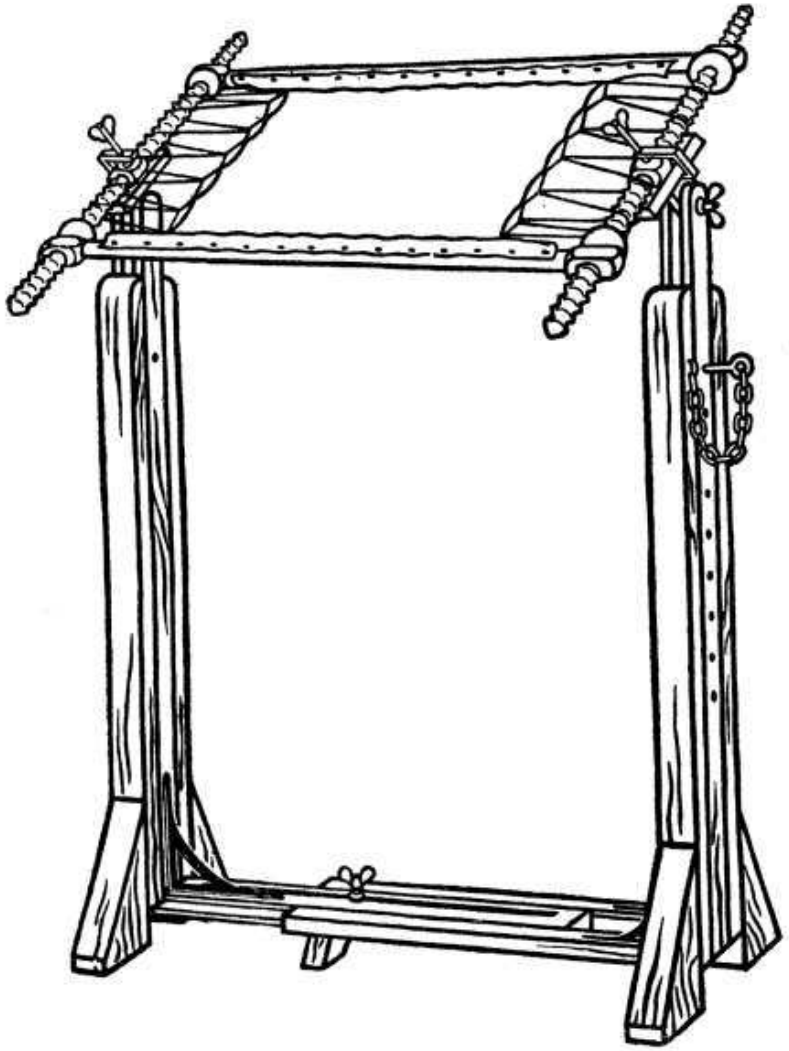
Calculating Coverage

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

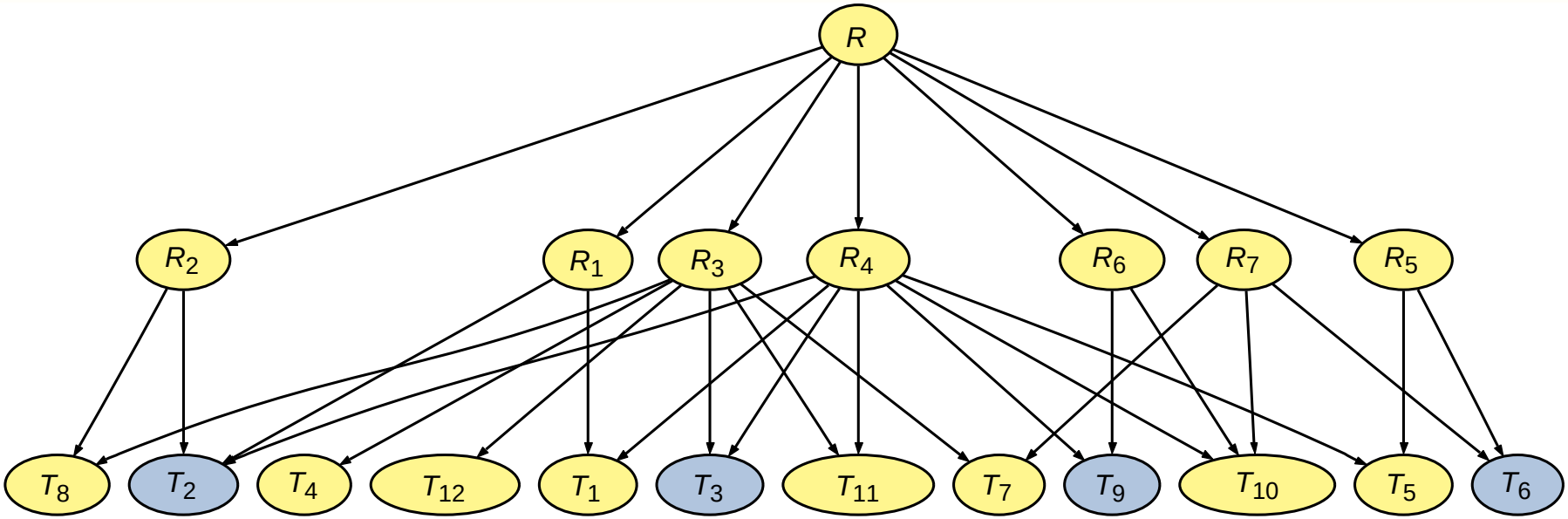
Regression Testing

The adequacy measurements can be used to support both test suite **reduction** and **prioritization**

Presentation Outline

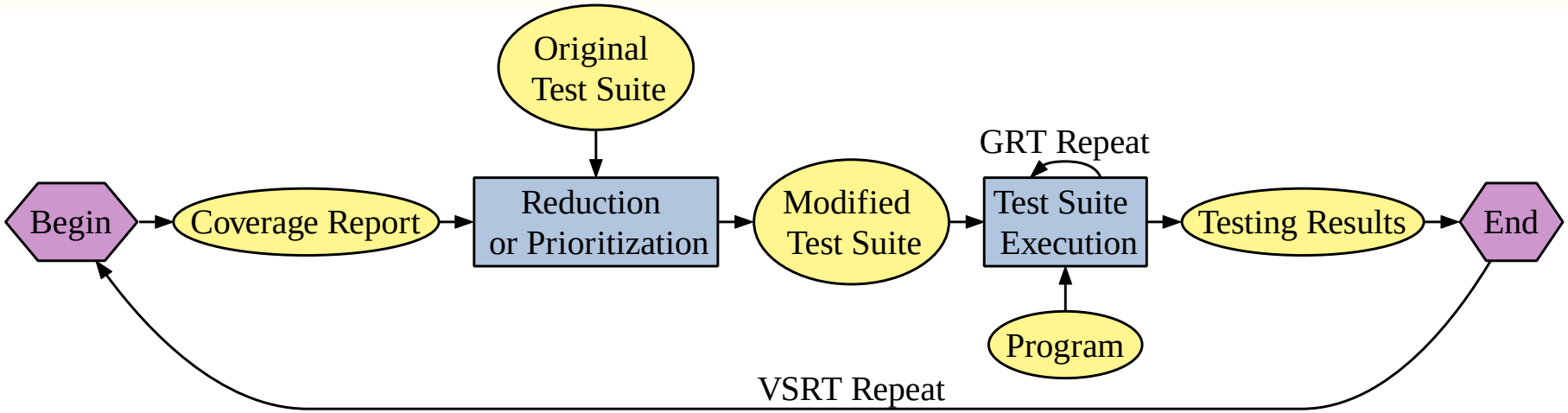


Finding the Overlap in Coverage



- $R_j \rightarrow T_i$ 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

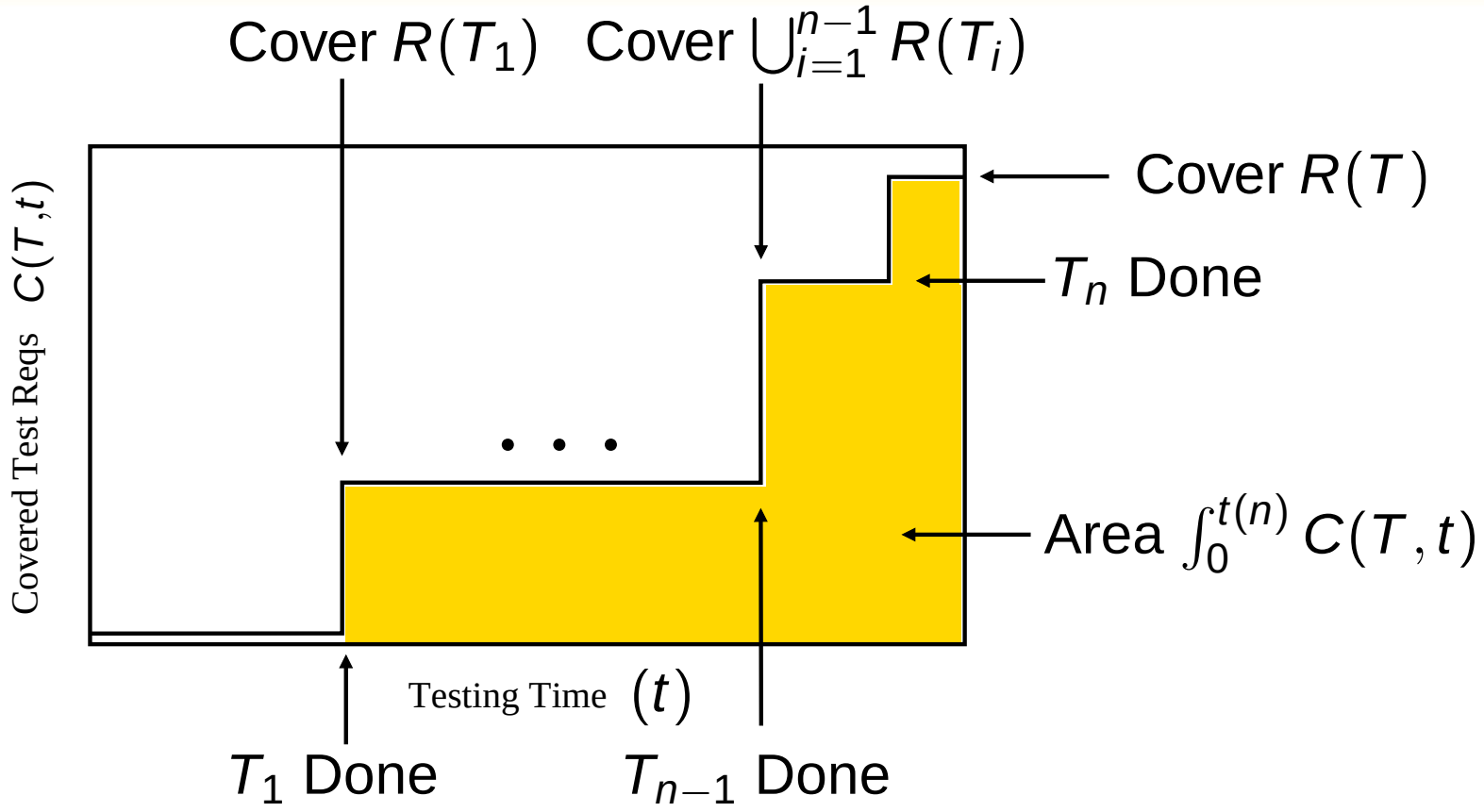
Reducing and Prioritizing the Tests



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 $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0,1]$

Characterizing a Test Suite

Test Information

Test Case	Cost (sec)	Requirements				
		R_1	R_2	R_3	R_4	R_5
T_1	5	✓	✓			
T_2	10	✓	✓	✓		✓
T_3	4	✓			✓	✓

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

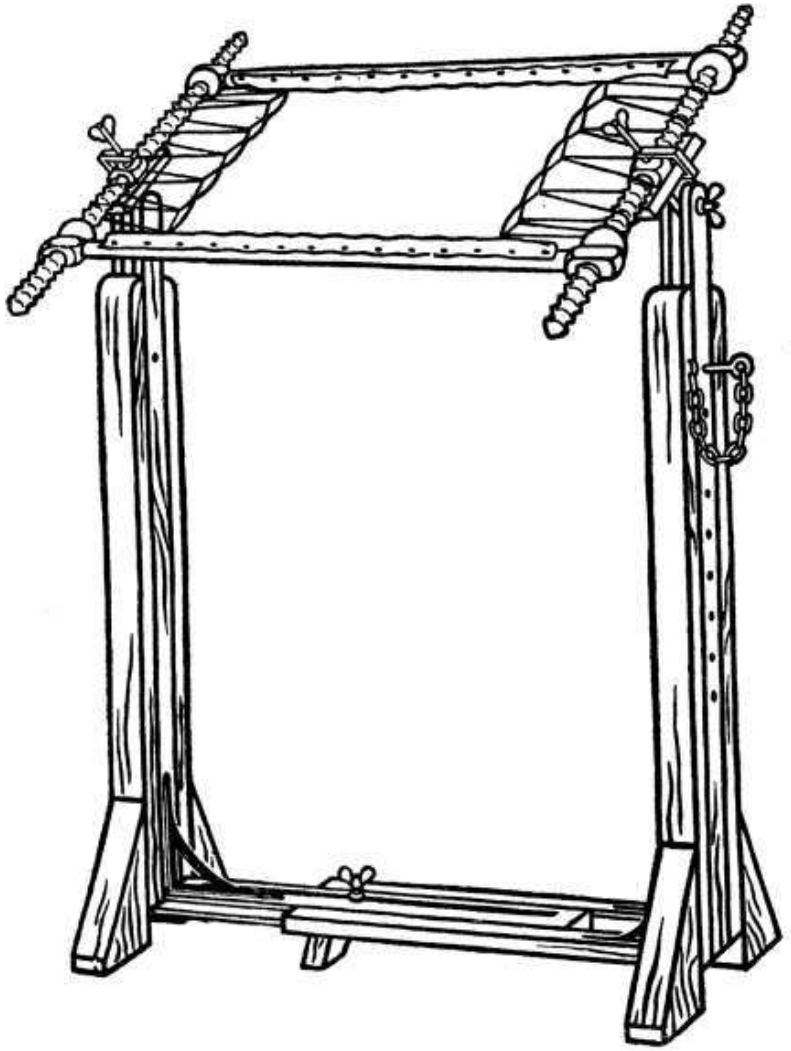
Calculating CE and CE_u

Ordering	CE	CE_u
$T_1\ T_2\ T_3$	.3789	.4
$T_1\ T_3\ T_2$	.5053	.4
$T_2\ T_1\ T_3$	.3789	.5333
$T_2\ T_3\ T_1$	.4316	.6
$T_3\ T_1\ T_2$	.5789	.4557
$T_3\ T_2\ T_1$	.5789	.5333

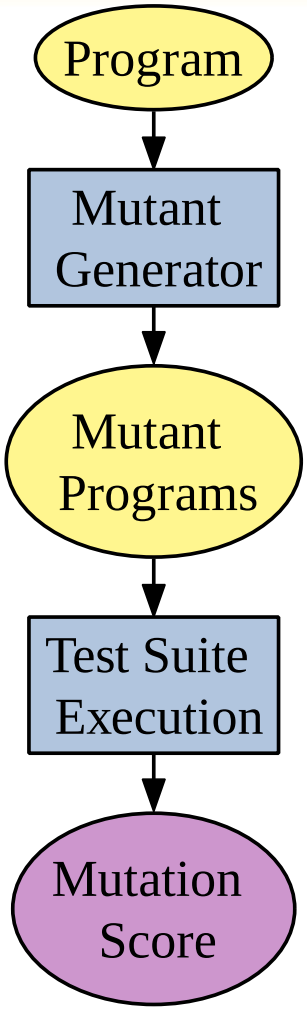
Observations

- 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 $CE < CE_u$

Presentation Outline



Mutation Testing Techniques



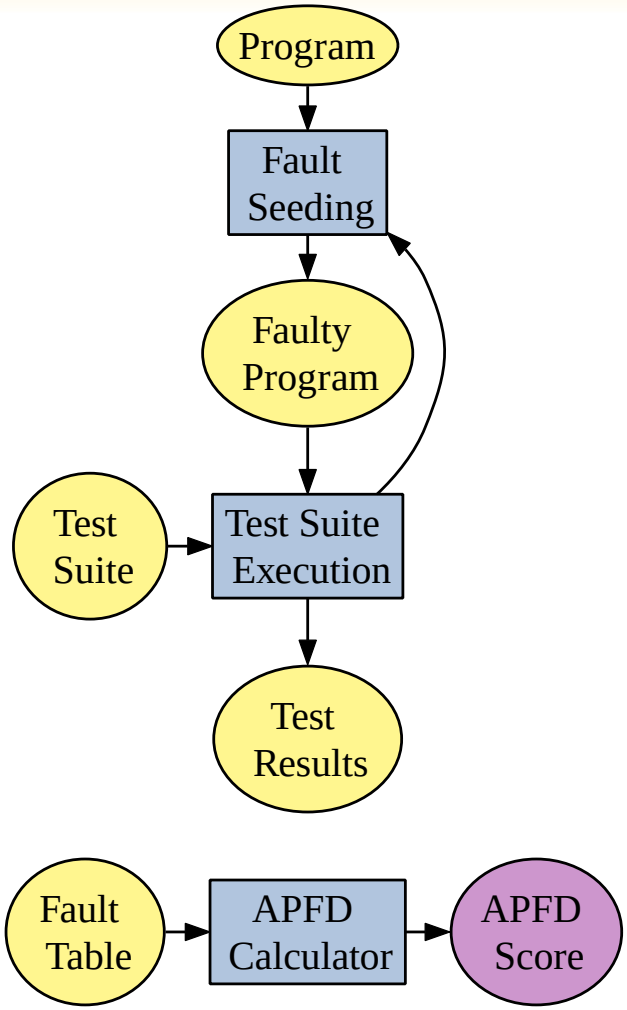
Mutant Creation

- A **mutation testing** tool (e.g., μJava or Jumble) inserts defects into the program under test
- **Question**: Why are we **inserting** faults into the 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

Average Percentage of Faults Detected



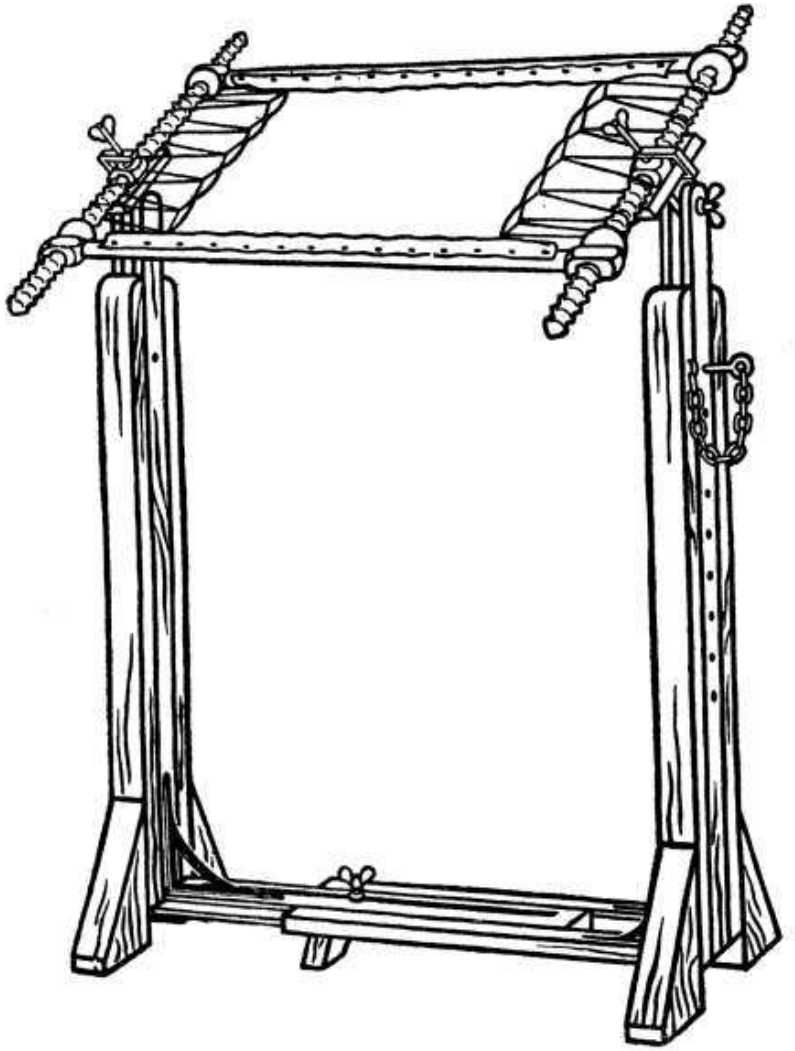
Fault Seeding

- Use **known faults** or a **mutation testing** tool (e.g., μJava or Jumble) to insert defects into the program
- Determine which test(s) are able to detect the seeded faults and construct a **fault table**

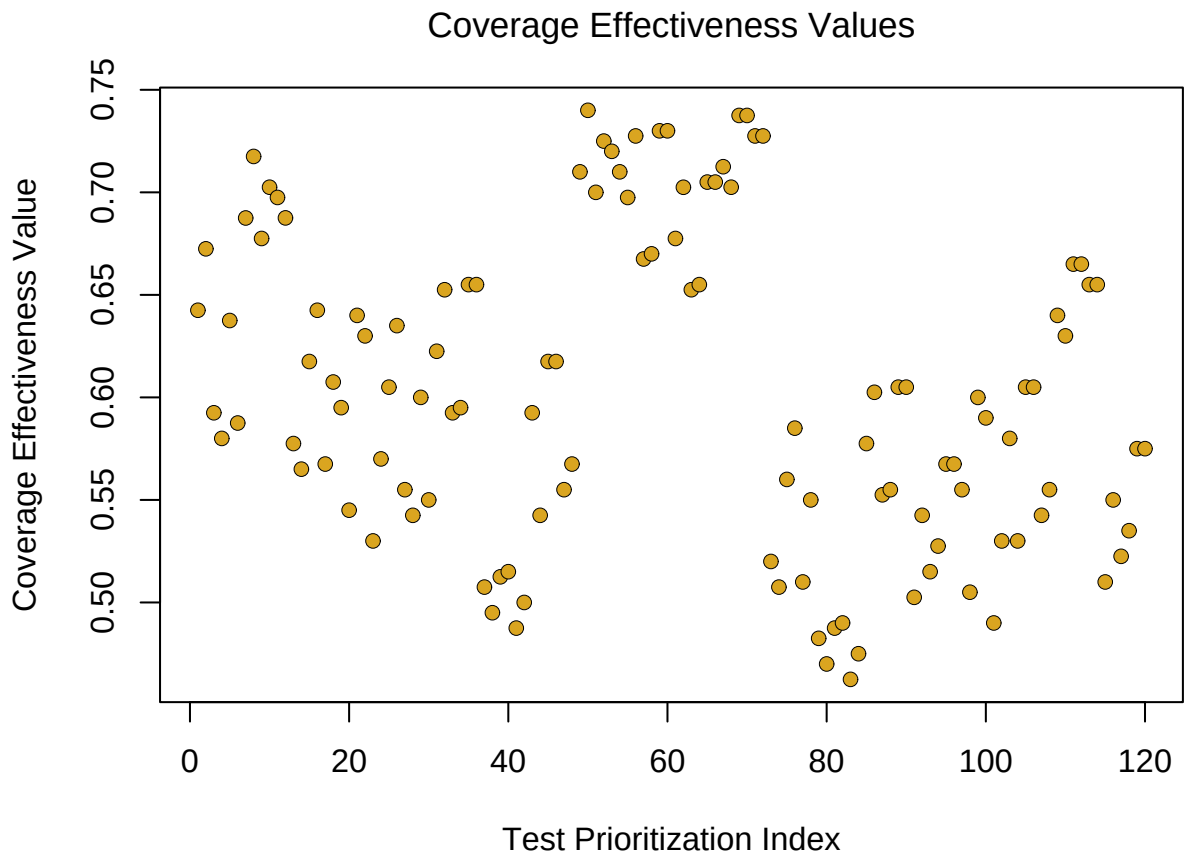
APFD Calculation

A test **ordering** has a higher APFD score if it **rapidly** detects the faults

Presentation Outline

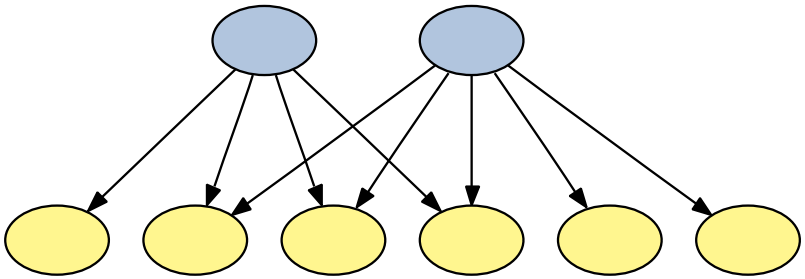


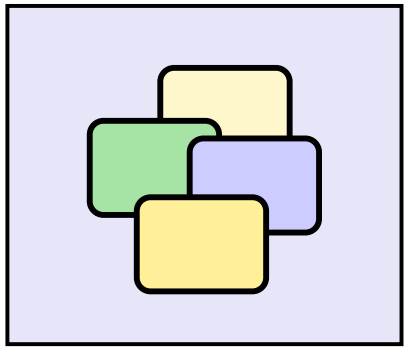
Search-Based Test Suite Prioritization



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/**