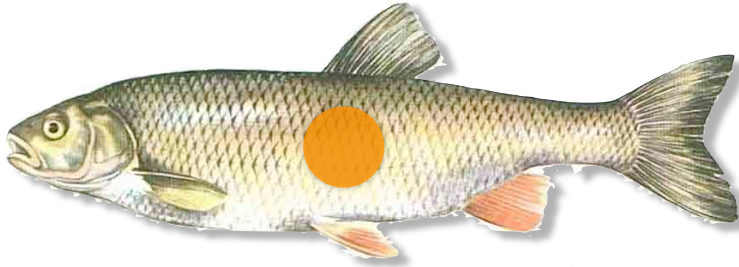


# Mutation Testing

Andreas Zeller



1

---

---

---

---

---

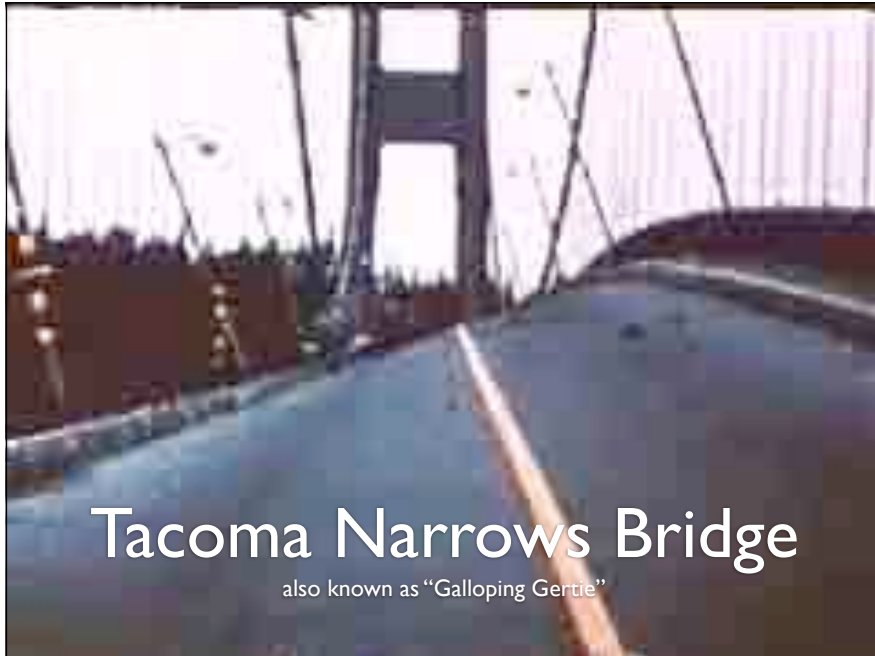
---

---

---

---

---



Tacoma Narrows Bridge  
also known as "Galloping Gertie"

2

The Tacoma Narrows Bridge is a pair of mile-long (1600 meter) suspension bridges with main spans of 2800 feet (850 m), they carry Washington State Route 16 across the Tacoma Narrows of Puget Sound between Tacoma and the Kitsap Peninsula, USA. The first bridge, nicknamed Galloping Gertie, was opened to traffic on July 1, 1940, and became famous four months later for a dramatic wind-induced structural collapse that was caught on color motion picture film.

---

---

---

---

# Learning from Mistakes

- Key idea: Learning from earlier mistakes to prevent them from happening again
- Key technique: *Simulate earlier mistakes* and see whether the resulting defects are found
- Known as *fault-based testing* or *mutation testing*

3

---

---

---

---

---

---

---

---

---

---

# A Mutant



A blue lobster (one in two million), an example of a genuine mutant. Blue American lobster (*Homarus americanus*). Taken at the New England Aquarium (Boston, MA, December 2006. Copyright © 2006 Steven G. Johnson and donated to Wikipedia under GFDL and CC-by-SA.

4

## Seeding Defects

- We seed defects into the program  
generating a *mutant* – a mutation of the original program
- We run the test suite to see whether it detects the defects (“kills the mutants”)
- A mutant not killed indicates a weakness of the test suite  
The mutant may also be 100% equivalent to the original program

5

## Saarbrücken



6





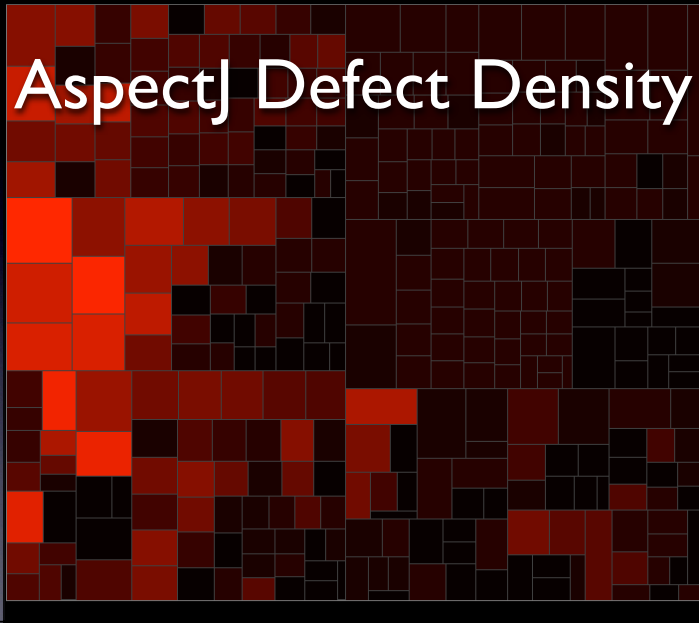
# Weyuker's Hypothesis

The adequacy of a coverage criterion can only be intuitively defined.

Established by a number of studies done by E. Weyuker at AT&T. "Any explicit relationship between coverage and error detection would mean that we have a fixed distribution of errors over all statements and paths, which is clearly not the case".

13

# AspectJ Defect Density



14

# A Bad Test

```
class TrueStoryTest {
    public int test_all(Object other)
    {
        executeForSomeTime();
        assertTrue(true);
    }
}
```

100% coverage – and never fails

15













# Measuring Impact

- How do we characterize “impact” on program execution?
- Idea: Look for changes in *pre- and postconditions*
- Use *dynamic invariants* to learn these

34

---

---

---

---

---

---

---

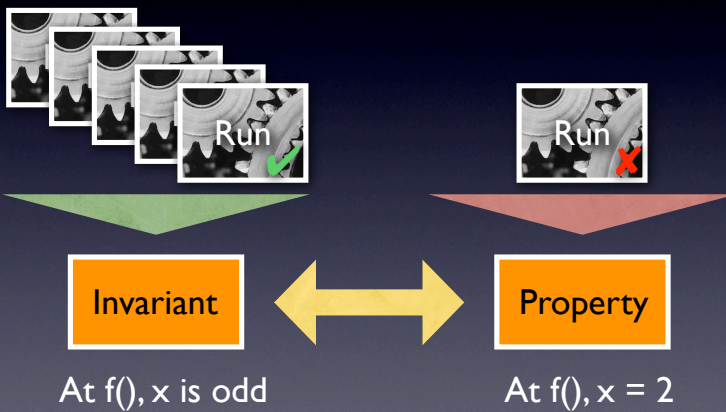
---

---

---

# Dynamic Invariants

pioneered by Mike Ernst's Daikon



35

---

---

---

---

---

---

---

---

---

---

# Example

```
public int ex1511(int[] b, int n)
{
    int s = 0;
    int i = 0;
    while (i != n) {
        s = s + b[i];
        i = i + 1;
    }
    return s;
}
```

**Precondition**  
n == size(b[])  
b != null  
n <= 13  
n >= 7

**Postcondition**  
b[] = orig(b[])  
return == sum(b)

- Run with 100 randomly generated arrays of length 7–13

36

---

---

---

---

---

---

---

---

---

---













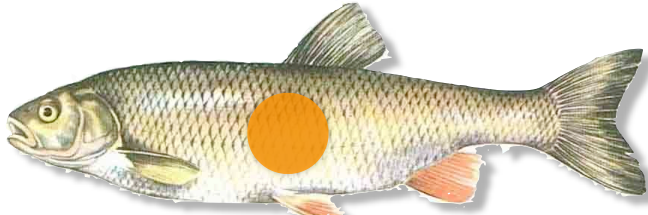




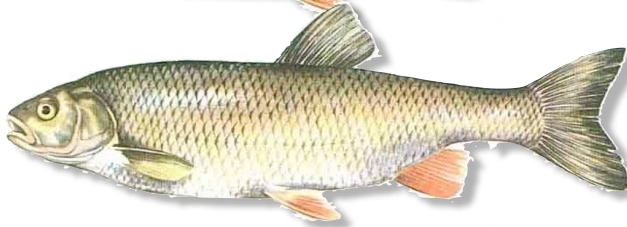


# Counting Tags

50



300



Let's assume over the next week, we ask fishermen to count the number of tags. We find 300 untagged and 50 tagged fish.

---

---

---

---

---

---

---

---

---

---

64

# Estimate

$$\frac{1,000}{\text{untagged fish population}} = \frac{50}{300}$$

...and we can thus estimate that there are about 6,000 remaining untagged fish in the lake.

---

---

---

---

---

---

---

---

---

---

65



That's how we can tell how many fish there are.

---

---

---

---

---

---

---

---

---

---

66



# Estimate

$$\frac{1,000}{\text{remaining defects}} = \frac{50}{300}$$

...and we can again estimate that there are about 6,000 remaining defects in our program. (A test suite finding only 50 out of 1,000 mutations is a real bad sign.)

---

---

---

---

---

---

---

---

---

---

70

**Issues**

**Efficiency** **Protection**

**Mutation Testing with Javalanche**

1. Learn invariants from test suite
2. Insert invariant checkers into code
3. Detect impact of mutations
4. Select mutations with the most invariants violated (i.e. the highest impact)

**Conclusion**

Are mutations with the highest impact *most useful*?

■ Remaining mutants detected ■ Top 25 killing mutants detected

Mutations that violate several invariants are most likely to be detected by actual tests – and thus the most useful!

---

---

---

---

---

---

---

---

---

---

71

# Assumptions

- Mutations are representatives for earlier mistakes  
so-called *competent programmer hypothesis*
- Failures come to be because of a combination of minor mistakes  
but there may be *logical errors* that cross-cut the program
- These hypotheses are not proven

---

---

---

---

---

---

---

---

---

---

72

