

# Faults' Context Matters

Jaymie Strecker and Atif Memon  
{strecker,atif}@cs.umd.edu



# Fault Context

- Fault = problem in source code
  - Failure = resulting problem in execution

```
function subtract(a,b) {  
    return a + b;  
}
```

- How likely that fault exists?
- How important to eliminate it?
- How likely that testing technique X detects it?
- *Answers determined by context*
  - Think outside the fault

# Studying Fault Detection

- Want to compare 2 testing techniques
  - Which detects the most faults?
    - Context: “representative” faults and programs
  - Which detects the benchmark faults?
    - Context: benchmark faults and programs
  - Which detects the important faults?
    - Context: severity model, usage model
- *How will they perform in my context?*

# Studying Fault Detection

- Two applications of testing technique T1

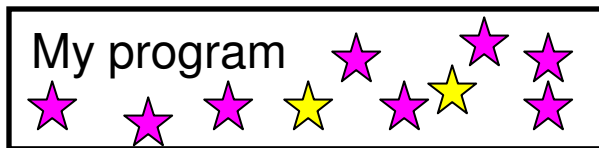


T1 detects 40% of faults ( ★ )



T1 detects 10% of faults ( ★ )

- What happened? Different kinds of faults.



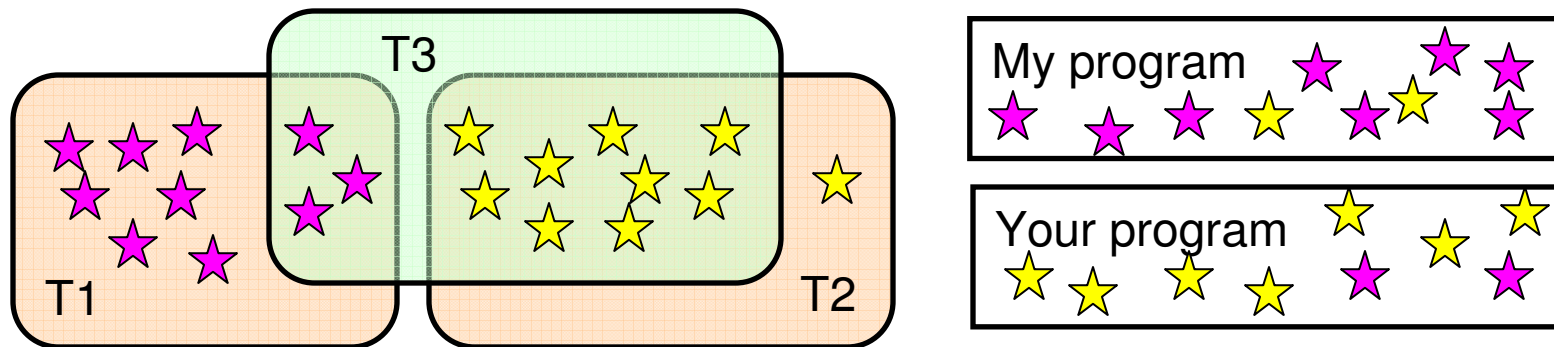
T1 detects 50% of ★ faults and 0% of ★ faults

# Studying Fault Detection

- Characterize faults in a way that
  - Provides useful information
    - Prevention, detection, repair, cost assessment
  - Is consistent
  - Is automatable
  - Requires no special artifacts or information
    - Formal specs may not exist
- Necessary? We think so.
- Sufficient? We don't know yet.

# Characterize by Testing Techniques

- F1 is a fault that T1 can detect
- F2 is a fault that T2 cannot detect
- T3 detects 30% of faults that T1 does and 90% of faults that T2 does
  - Consistent? Automatable? No special artifacts? **yes, if true of techniques**
  - Useful information? **yes**



# Conclusions

- When studying fault detection, characterize faults in a way that translates across contexts
  - Our criteria: useful information, consistent, automatable, no special artifacts
  - Characterize relative to testing techniques
- Future work
  - Empirically explore pros/cons and sufficiency
  - Tools to recognize and seed faults w.r.t. characterizations
  - New ways to characterize faults

# Ways to Characterize Faults

- Syntactically and semantically
  - Offutt and Hayes, 1996
- Relative to program models
  - Howden, 1976; Harrold et al., 1997; Dinh-Trong et al., 2005
- Relative to testing techniques
- Categorically
  - Basili and Selby, 1987; IEEE Standard Classification for Software Anomalies

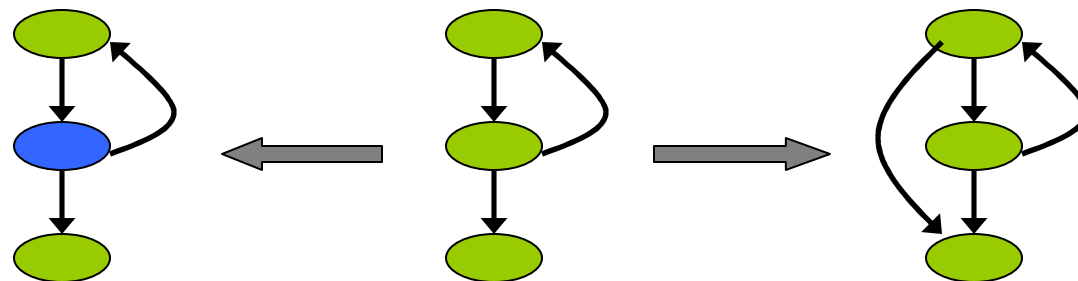
# Syntactic and Semantic Size

- Syntactic: change in source code
  - Actionable information? *no*
- Semantic: change in I/O mapping
  - Consistent? *no*
    - Sample of inputs
  - Useful information? *maybe not*
    - Elusiveness, cost

```
print("Pick a number between 1 and 100.");  
read(number);  
if (number == 4058295011)  
    launchMissiles(); // Oops
```

# Characterize by Program Models

- Program dependence graph (Harrold et al., 1997)
  - Structural fault = graph change
  - Statement-level fault = no graph change
  - Consistent? Automatable? No special artifacts? **yes**
  - Useful information? **yes**
- Other graph representations



# Categories

- Omissive and commissive
  - Consistent & automatable & useful information? *no*
- Initialization, control, data, computation, interface, cosmetic, ...
- IEEE Standard examples
  - Logic → misinterpretation
  - Data handling → data referenced out of bounds
  - Consistent? Automatable? *no*