System V&V

PSU CS 300 Lecture 8-1

Bart Massey
Assoc Prof Computer Science
Portland State University
<bart@cs.pdx.edu>

Check it

- System V&V builds on
  - requirements, design
  - unit test
  - inspection
- AKA “QA”, “Testing”
- Alternative: “User Testing”

Verification

- System operates according to requirements
- System implements design
- “Did we build the product right?”
- “It's just what I asked for, but not what I want”

Validation

- System actually works in wanted / intended way
- “Did we build the right product?”
- How to validate early?
  - work products
  - prototypes

Test is not validation

- **You can't validate a system by testing it**
  - **System test cases are generated from requirements**
  - **Valid system = verification + valid requirements**
- **User-generated tests help capture requirements**

Efficient, thorough testing

- **Big issue: how to find small test set with big leverage**
  - **Use inspection to eliminate uninteresting pieces**
  - **Use formal methods to “prove” big domains correct**
  - **Test what's left as best you can**

Inspecting for test

- Most code is boring; just moves data around
- Unit test works well on boring code
- Code without many defects doesn't need much test
- Simple requirements don't need much test

Subdomain proofs

- **Example (Massey / Haertel)**
  - **Print / round FP numbers = base conversion problem**
  - ***Most* numbers round right way automatically**
  - ***Prove* that rounding is right on all but special inputs**
  - ***Test* and *special-case* those**

More about coverage

- How do we estimate / measure that a set of test cases is “good”?
  - Domain coverage
  - Code coverage
  - Fault seeding / mutation

Branch coverage

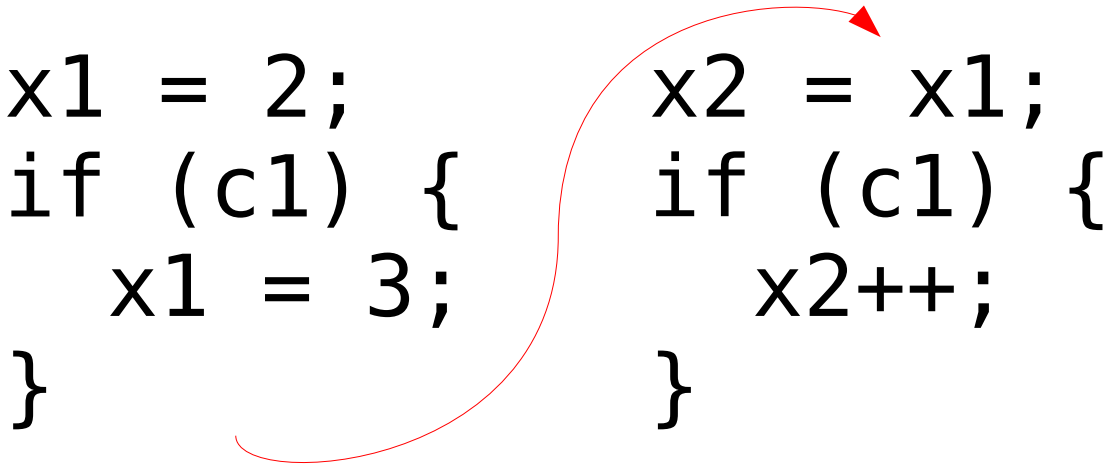
- **Branches taken each way**

```
if (true) {
  x = 3;
}
```

- **Exercises conditionals**
- **Subsumes statement coverage (cf dead code)**

Path coverage

- **All paths covered (4 here)**



- **Exercises data paths**

Bayes' Rule

- **It's worse than you think**

$$Pr(H|E) = \frac{Pr(E|H) \cdot Pr(H)}{Pr(E)}$$

- **Even if you find a bug, finding a fix is hard**

Risk

- **Risk equation**

$$R = \langle V(F) \rangle = \sum_{f \in F} Pr(f) \cdot V(f)$$

- **Risk management = minimizing R through decreasing Pr(f) for various f**

Various things that don't work in practice

- Testing only (must have recovery plan)
- Random testing only (must do other testing)
- 100% test coverage
- Multiple independent implementations

Has SW quality improved?

- **Heck yes. Over the last 25 years we have learned to**
  - **routinely build programs > largest 1980 programs**
  - **ship programs to naïve end users in unrepairable systems**
  - **routinely build mission / safety critical systems**

What are current woes?

- **Inappropriate tech for application (esp language)**
- **Insufficient application of**
  - **formal methods**
  - **inspection**
  - **root cause analysis**
- **Emphasis on fast *vs* good**

System V&V

PSU CS 300 Lecture 8-1

Bart Massey
Assoc Prof Computer Science
Portland State University
<bart@cs.pdx.edu>

Check it

- System V&V builds on
  - requirements, design
  - unit test
  - inspection
- AKA “QA”, “Testing”
- Alternative: “User Testing”

Verification

- System operates according to requirements
- System implements design
- “Did we build the product right?”
- “It's just what I asked for, but not what I want”

Validation

- System actually works in wanted / intended way
- “Did we build the right product?”
- How to validate early?
  - work products
  - prototypes

Test is not validation

- You can't validate a system by testing it
  - System test cases are generated from requirements
  - Valid system = verification + valid requirements
- User-generated tests help capture requirements

Efficient, thorough testing

- **Big issue: how to find small test set with big leverage**
  - **Use inspection to eliminate uninteresting pieces**
  - **Use formal methods to “prove” big domains correct**
  - **Test what's left as best you can**

Inspecting for test

- Most code is boring; just moves data around
- Unit test works well on boring code
- Code without many defects doesn't need much test
- Simple requirements don't need much test

Subdomain proofs

- **Example (Massey / Haertel)**
  - **Print / round FP numbers = base conversion problem**
  - ***Most* numbers round right way automatically**
  - ***Prove* that rounding is right on all but special inputs**
  - ***Test* and *special-case* those**

More about coverage

- How do we estimate / measure that a set of test cases is “good”?
  - Domain coverage
  - Code coverage
  - Fault seeding / mutation

Branch coverage

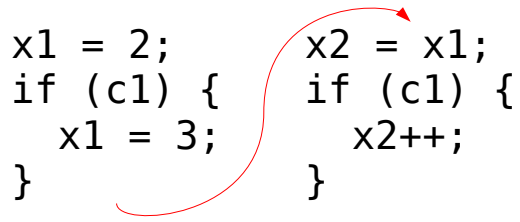
- **Branches taken each way**

```
if (true) {
  x = 3;
}
```

- **Exercises conditionals**
- **Subsumes statement coverage (cf dead code)**

Path coverage

- **All paths covered (4 here)**



- **Exercises data paths**

Bayes' Rule

- **It's worse than you think**

$$Pr(H|E) = \frac{Pr(E|H) \cdot Pr(H)}{Pr(E)}$$

- **Even if you find a bug, finding a fix is hard**

Risk

- **Risk equation**

$$R = \langle V(F) \rangle = \sum_{f \in F} Pr(f) \cdot V(f)$$

- **Risk management = minimizing R through decreasing Pr(f) for various f**

Various things that don't work in practice

- Testing only (must have recovery plan)
- Random testing only (must do other testing)
- 100% test coverage
- Multiple independent implementations

Has SW quality improved?

- **Heck yes. Over the last 25 years we have learned to**
  - **routinely build programs > largest 1980 programs**
  - **ship programs to naïve end users in unrepairable systems**
  - **routinely build mission / safety critical systems**

What are current woes?

- Inappropriate tech for application (esp language)
- Insufficient application of
  - formal methods
  - inspection
  - root cause analysis
- Emphasis on fast *vs* good