LECTURE 4: TESTING

Software Engineering

Mike Wooldridge

1.1 Bugs

- Errors of all kinds are known as "bugs".
- Bugs come in two main types:
  - compile-time (e.g., syntax errors) which are *cheap* to fix
  - run-time (usually logical errors) which are *expensive* to fix.

1 Testing

- Testing is critically important for quality software:
- Industry averages:
  - 30-85 errors per 1000 lines of code;
  - 0.5-3 errors per 1000 lines of code not detected before delivery.
- The ability to test a system depends on a thorough, competent requirements document.

1.2 Testing Strategies

- Never possible for designer to anticipate every possible use of system.
- Systematic testing therefore essential.
- Offline strategies:
  1. syntax checking & "lint" testers;
  2. walkthroughs ("dry runs");
  3. inspections
- Online strategies:
  1. black box testing;
  2. white box testing.

1.3 Syntax Checking

- *Detecting errors at compile time is preferable to having them occur at run time!*
- Syntax checking will simply determine whether a program "looks" acceptable — but completely dumb exercise.
- "lint" programs try to do deeper tests on program code:
  - will detect "this line will never be executed"
  - "this variable may not be initialised"

  (The Java compiler does a lot of this in the form of "warnings".)

1.4 Inspections

- Formal procedure, where a team of programmers read through code, explaining what it does.
- Inspectors play "devils advocate", trying to find bugs.
- Time consuming process!
- Can be divisive/lead to interpersonal problems.
- Often used only for safety/time critical systems.

1.5 Walkthroughs/Dry Runs

- Similar to inspections, except that inspectors "mentally execute" the code using simple test data.
- Expensive in terms of human resources.
- Impossible for many systems.
- Usually used as discussion aid.

1.6 Black Box Testing

- In *black box* testing, we ignore the internals of the system, and focus on relationship between *inputs* and *outputs*.
- *Exhaustive* testing would mean examining output of system for every conceivable input.

  Clearly not practical for any real system!
- Instead, we use *equivalence partitioning* and *boundary analysis* to identify *characteristic* inputs.

Equivalence Partitioning

- Suppose system asks for "a number between 100 and 999 inclusive".
- This gives three *equivalence classes* of input:
  - less that 100
  - 100 to 999
  - greater than 999
- We thus test the system against characteristic values from each equivalence class.

  Example: 50 (invalid), 500 (valid), 1500 (invalid).

Boundary Analysis

- Arises from the fact that most program *fail* at *input boundaries*.
- Suppose system asks for "a number between 100 and 999 inclusive".
- The boundaries are 100 and 999.
- We therefore test for values:

$\underbrace{99 \quad 100 \quad 101}_{\text{lower boundary}} \qquad \underbrace{998 \quad 999 \quad 1000}_{\text{upper boundary}}$

1.7 White Box Testing

- In white box testing, we use knowledge of the internal structure of systems to guide development of tests.
- The ideal: examine every possible run of a system.

  Not possible in practice!
- Instead: aim to test every statement at least once!
- EXAMPLE.

```
if (x > 5) {
 System.out.println(‘‘hello’’);
} else {
 System.out.println(‘‘bye’’);
}
```

There are two possible paths through this code, corresponding to $x > 5$ and $x \leq 5$.

Aim to cause each one to be executed.

2 Testing Plans

- Testing must be taken seriously, and rigorous *test plans* or *test scripts* developed.
- These are generated from requirements analysis document (for black box) and program code (for white box).
- Distinguish between:
  1. unit tests;
  2. integration tests;
  3. system tests.

3 Alpha and Beta Testing

- In-house testing is usually called *alpha testing*.
- For software products, there is usually an additional stage of testing, called *beta testing*.
- Involves distributing tested code to "beta test sites" (usually prospective customers) for evaluation and use.
- Typically involves a formal procedure for reporting bugs.
- Delivering buggy beta test code is embarrassing!