Week 9: More details of white-box testing

- What is it?
- Comparison with black-box testing
- What we should not validate
- Automated versus interactive testing
- Testing conditional and loop constructs

Terminology
- Some experts (e.g., Pezze & Young) call it structural testing
- Most others call it white box testing for contrast with "black box"
- Test cases are contrived to validate boundary conditions and other properties of the implementation.
  - TFD / TDD is obviously impossible

Comparison

- **White-box testing**
  - Well suited to early stages (unit and integration test)
  - Usually performed by original programmer
  - Validates internal logic/structure

- **Black-box testing**
  - Well-suited to later stages (system and acceptance test)
  - Can be performed by professional tester
  - Validates conformance with specifications

Is one "better" than the other?

Loop testing

- Tests should validate
  - the result if the loop is executed
    - 0 times (if allowed)
    - 1 time
    - multiple times
    - the maximum permitted number of times
  - post conditions in each case
  - any abnormal exit (break, throw, goto)
Decision/Condition coverage

- Test suite assures that every condition takes on every possible value at least once.

- That's great, but it doesn't guarantee:
  - that every possible path (combination of conditions) gets executed.
  - that every possible data value yields correct result.

  How can we minimize the probability of undetected errors?

Condition coverage and Boolean optimization

- Consider these two Boolean expressions:
  - \( a \&\& b \&\& c \&\& d \&\& e \)
  - \( (((a \| b) \&\& c) \| d) \&\& e \)

- Both contain 5 Boolean terms. How many tests (max. and avg.) are needed to validate them?
  - Assume left-to-right evaluation with logical short cuts:
    - \( \text{true} \| \text{anything} \) yields \text{true}
    - \( \text{false} \&\& \text{anything} \) yields \text{false}

- Why do we worry about that in a test?

Limits of Boolean optimization

- Is \( (((a \| b) \&\& c) \| d) \&\& e \) equivalent to \( e \&\& (d \| (c \&\& (a \| b))) \)?

- Should a very smart optimizing compiler consider them equivalent?

- Should a competent programmer have written the second form in the first place?
Taking advantage of Boolean optimization

```java
if (x != null && x.price() < 1000) . .
```

- The right side of `&&`
  - may not exist, or
  - may be expensive to evaluate (e.g. requiring disk accesses)

- Therefore, be very careful in suggesting rearrangement of terms in a Boolean expression.

Testing visual objects

- It's impossible or very difficult to contrive an automated verification of the layout of a window, a form, a report or some other display

- Therefore we usually have to rely upon interactive testing by a human operator.

So, what tests can we automate?

- Validation of **value objects**. They include:
  - Almost all **elementary items**.
  - Many **collections** (containers), especially those that are too large to examine manually.
  - **Composite items** that aren't too complicated or messy.

How do we validate the result of a function (method)?

a. Compare it to a hand computed value.
   - Most reliable, but tedious

b. Perform the function two different ways and compare the results.
   - Beware of duplicate errors, especially if the same programmer implements both.

c. For a 1-to-1 function: perform the inverse function and compare with the original argument.
   - But beware of compensating errors.
Step one

- Focus on validating the `equals` predicate (Java) or `==` operator (C++, C#), etc.
- Then you can use it in validating all the other methods that produce a value result.
- For floating-point internal representations, consider how close the results have to be to be considered `equal`.
  - Accuracy requirement may depend on the application.
  - Money and dates almost always have to be exact.
  - Many physical constants are known only to 7 or 8 digits.

Need we test conversion to external representation?

- Every C# class has (or inherits) a `ToString()` method.
  - It produces an external representation of an object.
  - C++ and other languages have similar facilities.
- When should an automated test validate the results of `ToString()`
  - i.e. by `AssertEquals` or another test for exact match to an expected result.

What's the problem?

Limitation of `ToString()`

- The exact form of the result is not part of the class's specification.
  - It's intended to provide some human-readable text, perhaps for interactive testing.
  - It inherits from `Object`, which is useless.

- It's just one acceptable possibility
  - Spacing, sequence, punctuation, etc, are up to the programmer who implemented it.
  - It may not even include all the member data items.
  - It can be changed in the future without affecting the class's specification.

Validating `ToString()`

- Don't compare for exact results
  - A future innocent "improvement" to `ToString()` might invalidate much of your test driver.
  - That may discourage maintenance programmers from improving `ToString()`.

- Do verify that it exists and that its result is legible.
Generalizing the rule

- We agree that we shouldn't automate exact tests for the result of `ToString()`.
- More generally, never automate exact tests for any result that isn't rigorously specified.
  - Just knowing how it's supposed to work now is insufficient justification.
  - The presence of dozens or hundreds of tests (JUnit, Assert, or otherwise) for exact result can discourage maintenance programmers from making needed future improvements.

Final word (opinion):
Using debugging tools

- "There is also experimental evidence from both students and experienced programmers, showing that debugging aids do not assist the debugging process, and that, in terms of the speed and accuracy of finding the error, people who use their brains rather than a set of 'aids' seem to exhibit superior performance."
  - Glenford Myers, p. 133

- Use them when you have a reason to expect them to help, but don't automatically apply them to every testing situation.