Testing

Slides #3

CMPT 276 - © Dr. B. Fraser

Topics

1) What are common types of testing?
    a) Testing like a user: through the UI.
    b) Testing like a dev: through the code.
2) How can we write code to test code (via JUnit 5)?
3) How to do effective unit testing?
4) What makes a good bug report?

Types of Testing

Types of Testing

- Test to find bugs and to show a product works.
- How can we test (types of testing)?
  - ..
    - Test overall application's features
    - *"Is the program acceptable to customer?"*
  - ..
    - Test each class in isolation
    - *"Does this **class** do anything wrong?"*
- Testing can be done by a human (manual) or by code (automatic).

..

White vs Black Box

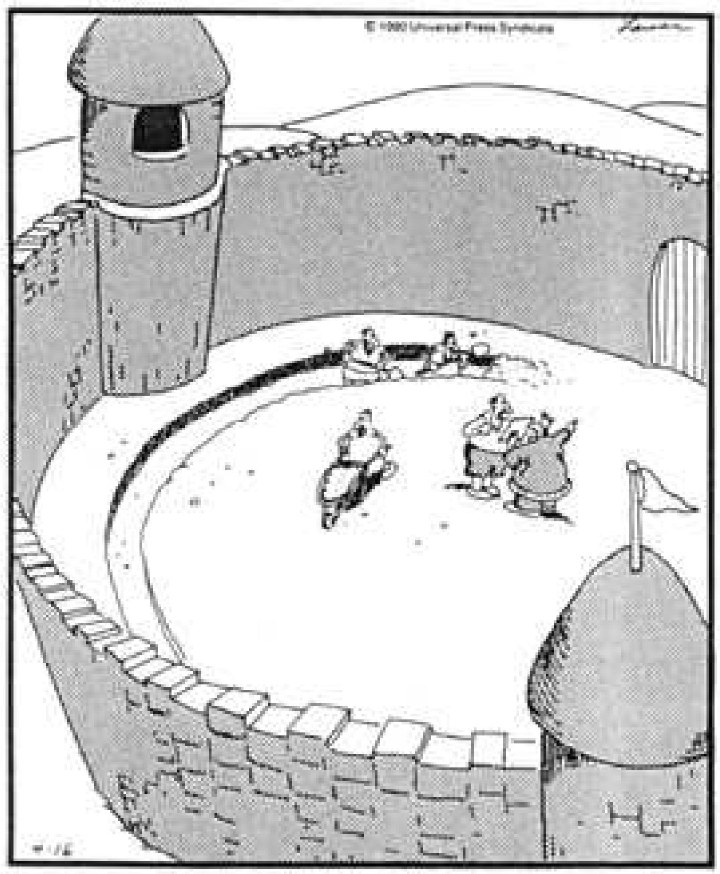
- When creating tests, ..
  do you have access to the system's code/design?
  - Knowing the code can help you..
  - Not knowing the code can help you see the big picture and..
- ..
  - Can see source code when writing tests.
  - Also called clear box or glass box.
- ..
  - Have no access to system internals.
  - Often for user interface testing.

Acceptance Tests

Acceptance Testing

- Acceptance Testing:..
  - Are needed features included?
  - Do the features work as expected?
- Can generate acceptance tests from..

THE FAR SIDE By GARY LARSON



Suddenly, a heated exchange took place between the king and the moat contractor.

Ex: Requirements to Acceptance Tests

Requirement

- Scroll bar’s slider shows the proportion of how much of the content is shown in the window.
- Scroll bar only visible when all content can not be shown in window at once.

Acceptance Tests

- With enough content to need scroll bar, double amount of content and slider should be half as tall.
- With enough content to need scroll bar, double window height and slider height should double.
- ... etc.

..

Acceptance Testing details

- Acceptance tests often manually done by a tester.

Quality Assurance Tester Job:

- Writing Test Cases and Scripts based on business and functional requirements
- Executing high complexity testing tasks
- Recording and reporting testing task results
- Proactively working with project team members to improve the quality of project deliverables

- Acceptance tests may be part of deploying a product
  - Alpha testing: users try out software at developer’s site.
  - Beta testing: software deployed for limited initial testing at customer’s site.

Unit testing with JUnit

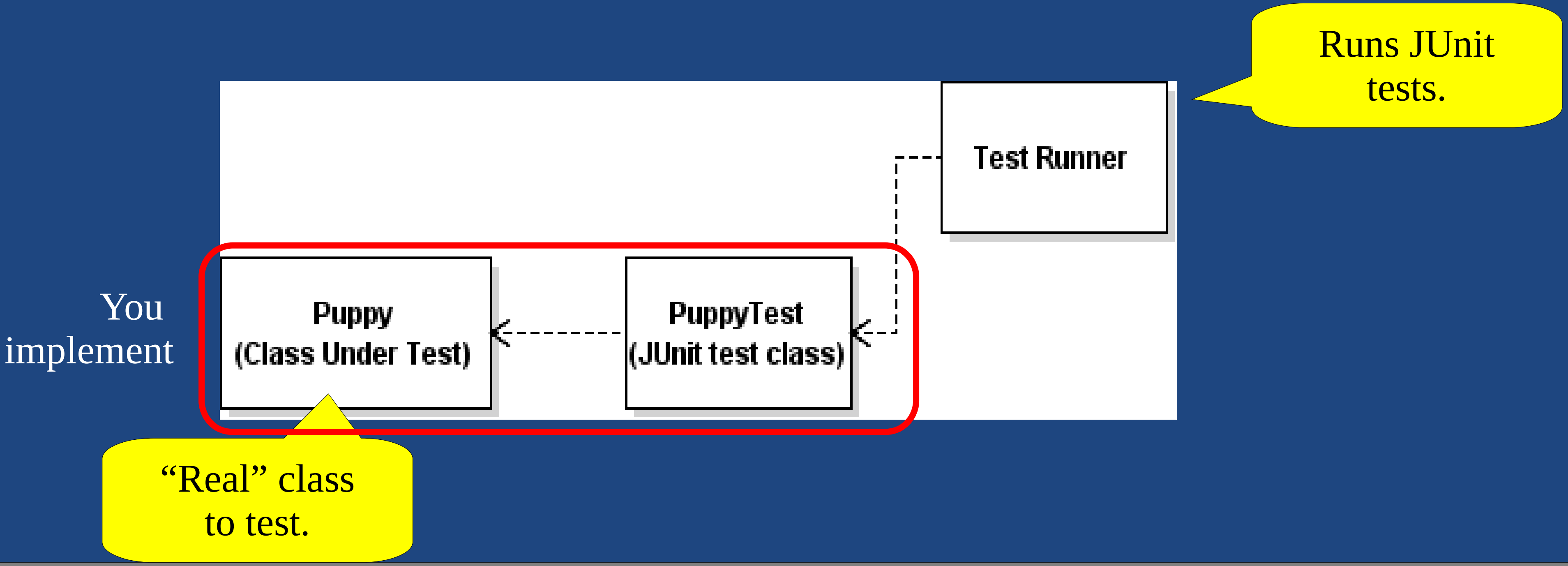
JUnit Unit Testing

- Unit Tests..

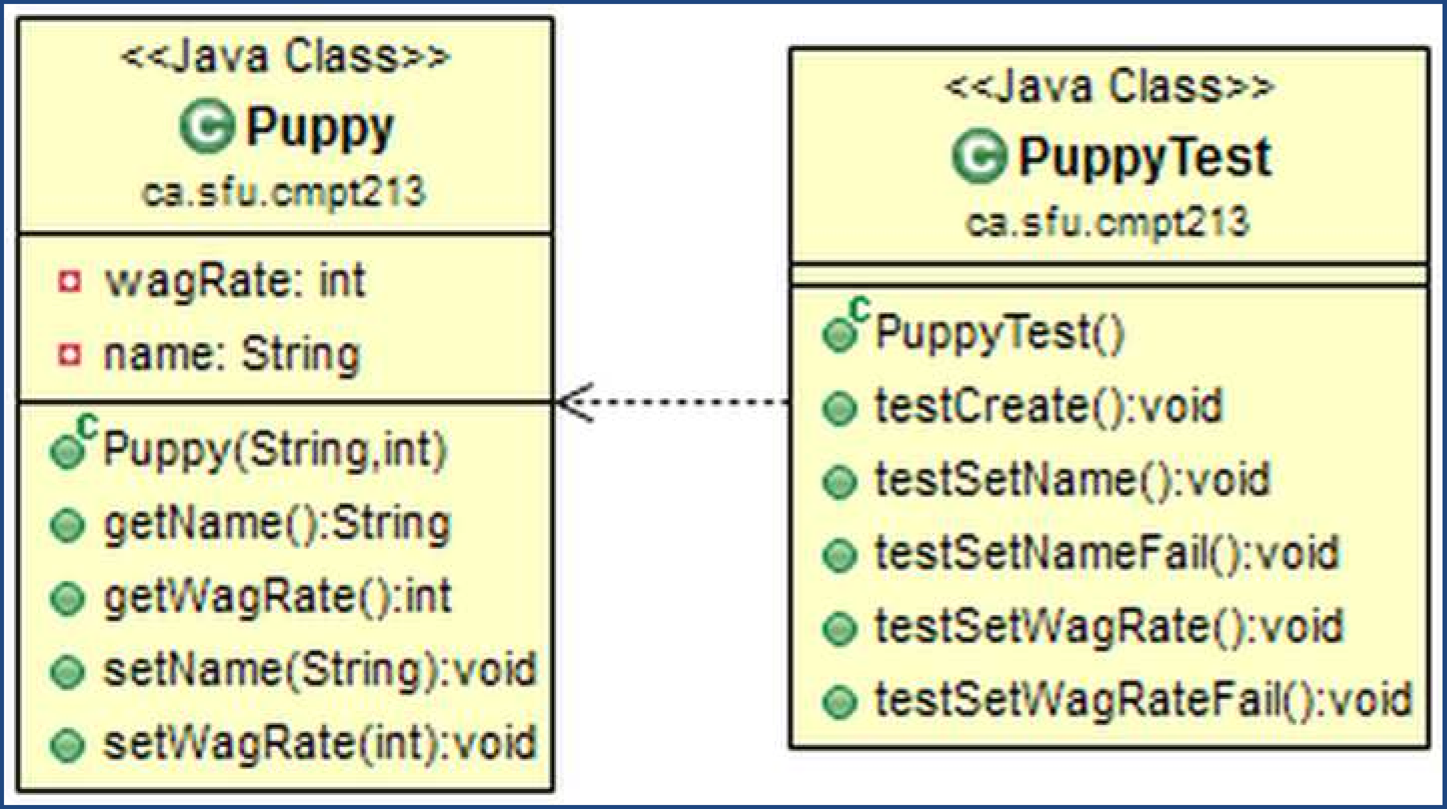
- Purpose:
  For *you* to “know” *your* code works.
  - Should test ~100% of a class.
  - Helps improve quality of code.
  - Supports aggressive refactoring because you can..

JUnit Context

- You create a test class which is..
- JUnit test runner executes your test class.



Basic JUnit Architecture



JUnit:
“Test Runner”
executes
methods with..

JUnit 5 Example

```
package ca.cmpt276.junit5;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class PuppyTest {
	@Test
	void testCreate() {
		Puppy rover = new Puppy("Rover", 100);
		assertEquals("Rover", rover.getName());
		assertEquals(100, rover.getWagRate());
	}

	@Test
	void testSetName() {
		Puppy rover = new Puppy("Rover", 100);
		rover.setName("Fluffy");
		assertEquals("Fluffy", rover.getName());
	}

	//... more tests omitted.
}
```

..

Test runner executes all methods with @Test annotaiton

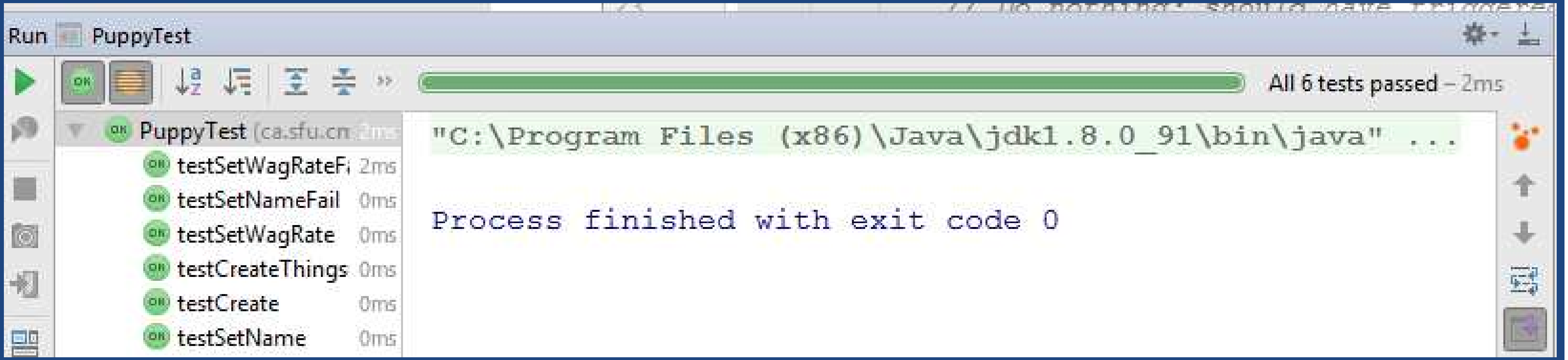
Tests are done using..

New instance of PuppyTest created for each JUnit test method:

Behaviour of one..

Test Runner

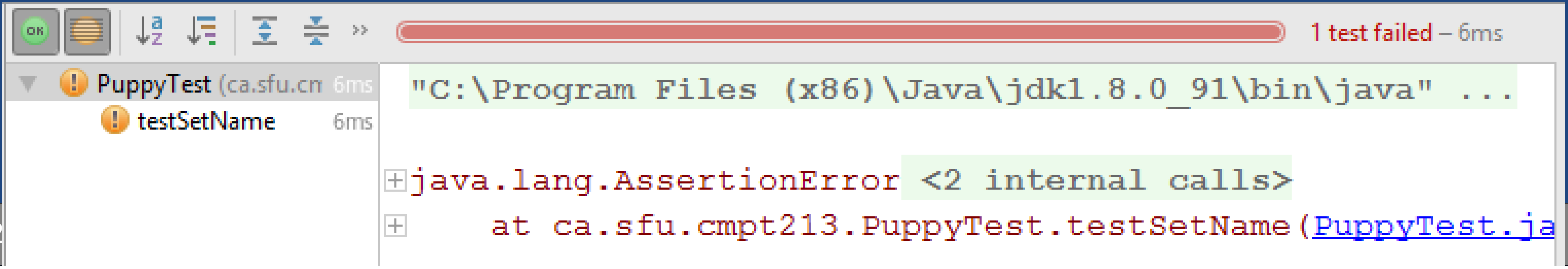
- Test runner executes @Test methods in test class.
- Displays results & coloured bar
  - Green-bar..



```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

Process finished with exit code 0
```

  - Red-bar..



```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

java.lang.AssertionError <2 internal calls>
    at ca.sfu.cmpt213.PuppyTest.testSetName(PuppyTest.ja
```

JUnit 5 Asserts: Basics

```
public class JUnitAssertTest {
    @Test
    public void demoAssertEquals() {
        String name = "Dr. Evil";
        assertEquals("Dr. Evil", name);
    }
    @Test
    public void demoOtherAsserts() {
        int i = 10;
        assertEquals(10, i);
        assertTrue(i == 10);
        assertFalse(i == -5);
    }
    @Test
    public void demoAssertEqualsOnDouble() {
        double weight = (1 / 10.0);
        assertEquals(0.1, weight, 0.000001);
    }
    // Array support: assertArrayEquals()
}
```

Doubles have limited precision.
3rd arg is the “delta” to tolerate

..

JUnit 5 Asserts: Exceptions

```
public class JUnitAssertTest {
    private void throwOnNegative(int i) {
        if (i < 0) {
            throw new IllegalArgumentException();
        }
    }
    @Test
    void testThrows() {
        assertThrows(IllegalArgumentException.class, () -> {
            throwOnNegative(-1);
        });

    }
    @Test
    void testNoThrows() {
        throwOnNegative(1);
    }
}
```

Code likely in class under test
(shown here for simplicity)

Use to test exception throwing..

IllegalArgumentExecption

Lambdas: needs Java 1.8+ compatibility
File --> Project Structure --> Module -->
Select "app" in list, select Properties tab
Set *Source Compatibility* to 1.8 (Java 8)
Set *Target Compatibility* to 1.8 (Java 8)

JUnit 5 Asserts: Disable

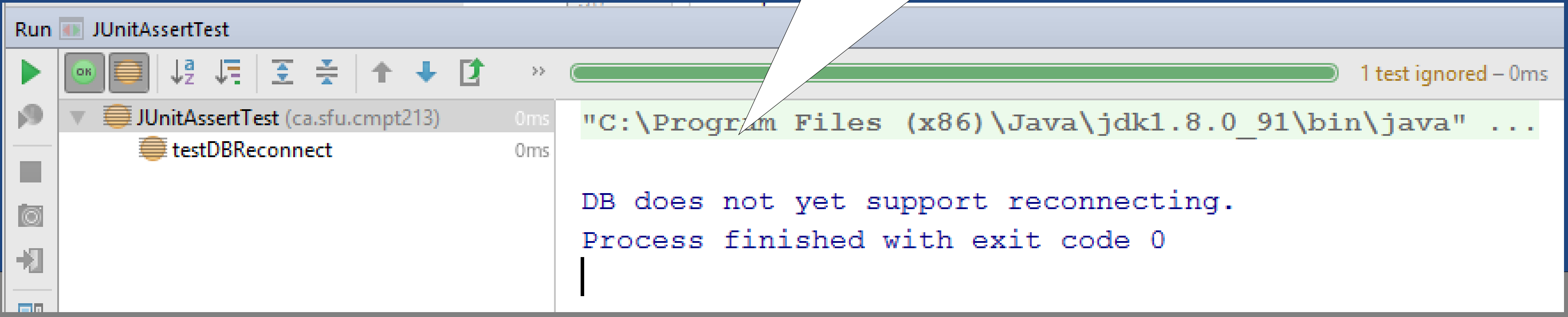
```
public class JUnitAssertTest {

    @Disabled("DB does not yet support reconnecting.")
    @Test
    void testDBReconnect() {
        // ... put your JUnit tests of the not-yet implemented code....
        fail();         // Automatic fail...
    }
                ..

}
```

Ignore the test so "to-be-done" style tests do not break testing.

Gives warning message to highlight that some tests not yet enabled.



```
"C:\Program Files (x86)\Java\jdk1.8.0_91\bin\java" ...

DB does not yet support reconnecting.
Process finished with exit code 0
```

Android Studio Demo

1) Create JUnit Test Class:
   1) Open class under test,
   2) Click class name, alt-enter --> *Create Test*
   3) Select *JUnit 5*, click *OK*
   4) Select *...\app\**src**\test\java\\....* folder
2) Execute Tests:
   1) *Run --> Run... (alt-shift-F10)*
   2) Select your JUnit test class.
3) Run app: *Run --> Run*...; select "*app*"

IntelliJ JUnit Video Tutorials:
Basics: https://www.youtube.com/watch?v=Bld3644bIAo&t
More: https://www.youtube.com/watch?v=xHk9yGZ1z3k&t

Unit Testing Discussion

FIRST: Properties of Good Tests

- ..
  - Run all tests very often; slow tests less useful
- ..
  - Each test has a small “Single Responsibility”
- ..
  - Not random: if they fail for you, they fail for me
- ..
  - User does not have to read through output
- ..
  - Write tests soon after (before?) production code

Effective unit tests

- Unit testing should be..
- Test ‘class under test’ for:
  - Works for expected normal inputs.
  - Works for extreme or invalid inputs.
- Testing strategies
  -
    - group input values which are "similar"
    - test based on these groupings.
  -
    - use guidelines to choose test cases.
    - guidelines cover common programming errors.

Input Vector

- Input Vector
  ..
  - Ex: printf("Hello %d", 42);
    Has input vector {"Hello %d", 42}
  - When calling a function with an input vector, the function follows a path of execution through its code:

    Ex: the "*then*" for one if statement, and the "*else*" for another
- Test Vectors
  ..
  - Use a small (but good!) set of test vectors to keep testing efficient

Equivalence Class Partition (ECP) Testing

- Equivalence Class
  - A region of values in the input data for which ..
  - The boundaries between these regions are the Equivalence Class Partitions
- Ex: Multiplying two integers

```
int multiply(int a, int b) {
    return a * b;
}
```

  - Input: Positive vs negative input values yields positive vs negative output.

Equivalence Classes

- Identify the equivalence classes, and the equivalence class partitions for the following:

```
/** Return a grade based on the percent:
 *   50 to 100 = 'P'
 *   0 to <50 = 'F'
 *   otherwise throw an exception.
 */
char assignGrade(int percent);
```

Equivalence Class Partition (ECP) Testing

- Since all values inside an EC behave similarly:
  - it is likely that the paths of execution for all input vectors within a single equivalence class are the same.
  - Therefore, with ECP Testing we test one value from each equivalence class. Therefore,..

- Example

  ```
  char assignGrade(int percent);
  ```

  We might test:.. -10, 10, 60, 110

Boundary Value Analysis

- ECP testing is..
  - Testing one value per partition does not adequately test the boundaries of the partitions.
  - Could have boundary too high/low:
    - off by one
    - < vs <=
- Boundary Value Analysis
  - For each ECP (the boundary between two equivalence classes),
    ..

Test Vector Selection

- Complete the table

```
/** Print age to screen.
 *   Throw an exception
 *    if age < 0 or > 120.
 */
void printAge(int age);
```

Equivalence class partitions	Equivalence class partition test vectors	Boundary value analysis test vectors

CORRECT: Boundary conditions

- Think about the following for boundary conditions

– Conformance:	Does value conform to expected format?
– ..	Is array of values ordered correctly?
– Range:	Is value within min/max?
– Reference:	Consider “external” code references
– ..	Does value exist (not null? not zero?)
– Cardinality:	Are there exactly enough values
– ..	Everything happen in order? At right time? Fast enough?

General testing guidelines

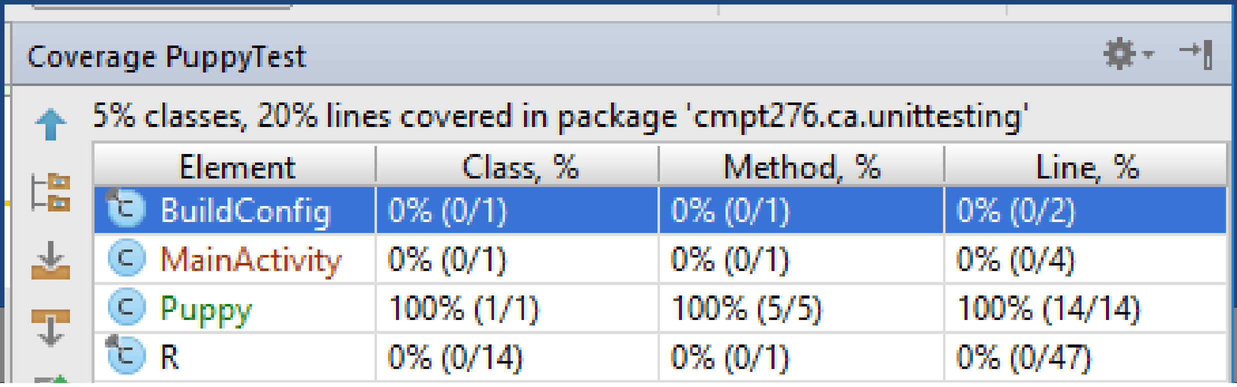
Choose test vectors based on some rules-of-thumb or guidelines to try and catch many common errors:

- ..
- Cause buffers to overflow;
- Force calculation result to be too large (or small): (overflow & underflow).
- Testing With Arrays:
  - Different # elements. Ex..
  - Put desired element..

Code Coverage

- Code Coverage:..

- Want ~100% Code Coverage
  - All lines of code executed at least once.
  - Quite hard to achieve (complex error cases, asserts, ..)
  - This should almost be the *bare minimum*:
    Tests run..
- Demo (Android Studio or IntelliJ)
  Run --> Run PuppyTest with Coverage



Coverage PuppyTest

5% classes, 20% lines covered in package 'cmpt276.ca.unittesting'

Element	Class, %	Method, %	Line, %
BuildConfig	0% (0/1)	0% (0/1)	0% (0/2)
MainActivity	0% (0/1)	0% (0/1)	0% (0/4)
Puppy	100% (1/1)	100% (5/5)	100% (14/14)
R	0% (0/14)	0% (0/1)	0% (0/47)

Test Code Quality

- Unit tests are integral part software development:
  ..
  as the rest of the project.
  - Only possible if you don’t think of tests as throw-away or beneath your coding skill.

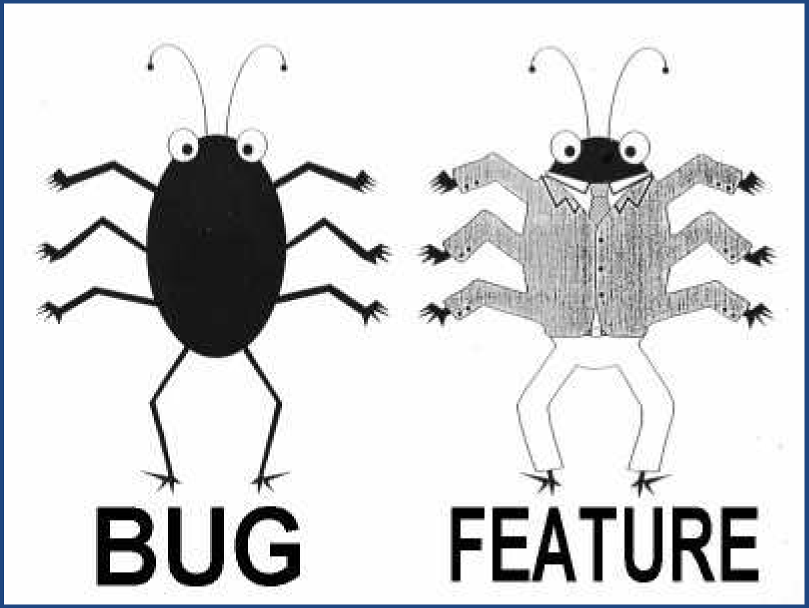
- Good code quality makes maintenance easier
  - Keeps tests current and relevant
  - Poor code makes tests obsolete fast (and useless)!
  - Unreliable tests cause developers to lose trust.

Finding Many Bugs

- If you find a function which is quite buggy, don’t debug it: ..
  - Good unit testing only finds..
  - A hacked together routine indicates poor understanding of its requirements:
    - If many bugs are discovered now,
      then many bugs will be encountered later!
- More tests cannot solve this problem:
  Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often.

  McConnel, 2004

Bug reports



Bug Report

- Submit a bug report when a defect is found.

Bug Report Component	Description
	Concise, 1 line description of problem.
	Which product had error.
	Actions to cause error. Does it always occur, or only occasionally? Create simple example to demonstrate.
	What the steps should do, vs what actually do. Ensure it is actually an error not a feature: "Working as intended"?
	Software version, OS, hardware, drivers, ...

Bug Report Example

Bug Report Component	Example
Summary	Upload crashes on MP3 file drag and drop.
Component	File upload window.
Steps to Reproduce	1. Open app to upload window. 2. Select two MP3 files in file explorer. 3. Drag into upload window. 4. Application flashes and crashes. Crash is repeatable.
Expected vs Actual result	Expected “No flashing and no crashing” (files should upload without app crashing)
Environment	ShareFiles 1.2.5, Win10, Dell XYZ, Norton 3

Inspired by an actual bug report submitted by someone I know.

Bug suggestions

- The better the bug report, the more likely the developer is to identify the problem and fix it.
- Example files:
  - For an office application, or a compiler, provide an example file which causes the problem.
- Screenshots:
  - A picture of the problem is great at definitively showing what happened.
  - Developers are often..

Life-cycle of a bug

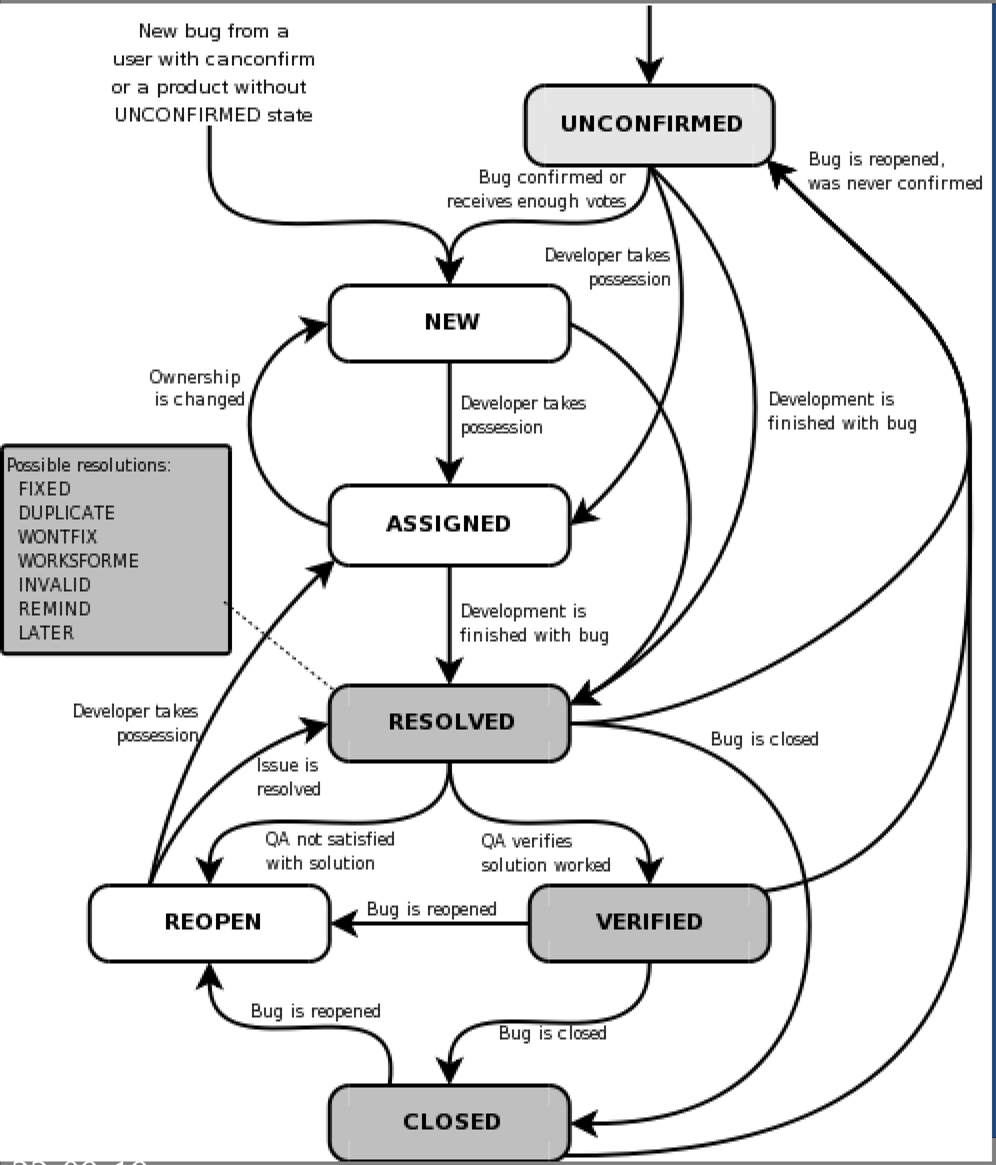

New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug confirmed or receives enough votes
Bug is reopened, was never confirmed
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
REMIND
LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Issue is resolved
Bug is closed
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

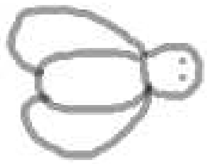
Image Source: Bugzilla – lifecycle.

- Some resolutions:
  - Fixed
  - Duplicate
  - Won't Fix
  -
  -
    - "ID-10-T"
    - "PLBKAC"
  - Enhancement / feature request

BUGS HAVE FEELINGS TOO

IF YOU FIND A BUG:
REPORT IT

BUGS DON'T LIKE
TO BE FORGOTTEN



IF YOU FIND A BUG:
GET TO KNOW THEM

BUGS LIKE TO BE
UNDERSTOOD

IF YOU FIND A BUG:
TAKE A PHOTO

BUGS LIKE TO KEEP MEMORIES
OF THE OCCASION

IF YOU FIND A BUG:
GET TO KNOW THEIR MATES

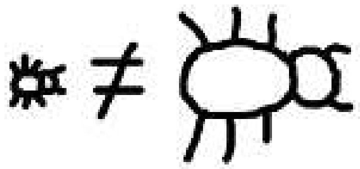
BUGS ARE SOCIALITES

IF YOU FIND A BUG:
REPORT IT QUICK

OTHERWISE BUGS SETTLE IN AND
MAKE A HOME FOR THEM SELVES

IF YOU FIND A BUG:
BE HONEST

BUGS DON'T LIKE
GOSSIPS

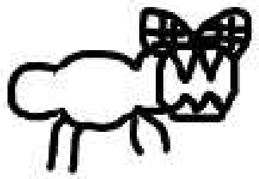


IF YOU FIND A BUG:
NOTE HOW YOU
MEET THEM

BUGS ARE ROMANTICS

IF YOU FIND A BUG:
DON'T IGNORE IT

BUGS CAN BITE IF
NOT APPRECIATED



Summary

- White-box knowledge of internals;
  Black-box uses external interface only.
- Test Types
  - Acceptance for checking features in product.
  - JUnit for detailed unit testing (white-box):
    assert...(), @Test, @Disable, assertThrows().
- Good JUnit tests
  - Equivalence Class Partition testing, Boundary value analysis, guidelines for testing.
  - High-quality test code: maintain it!
- Bug reports include
  - Description, steps to reproduce, environment info.