Testing and Debugging
Lecture 9

Waterford Institute of Technology

March 29, 2014

John Fitzgerald

Software complexity

Douglas Crockford: Javascript expert

- "Computer programs most complex things humans make"

An apt definition

- Software is applied thought

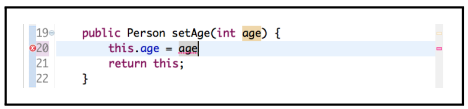
How complex are thoughts?

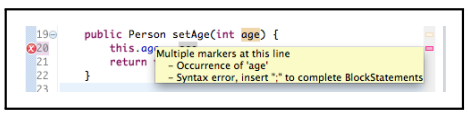
- How to impose order and uniformity to thoughts?
- How to avoid spaghetti code?

Program errors

Syntax

- **Syntax errors**
  - Relate to structure and grammar of code.
  - Usually easily identified.
  - Error messages generated by compiler



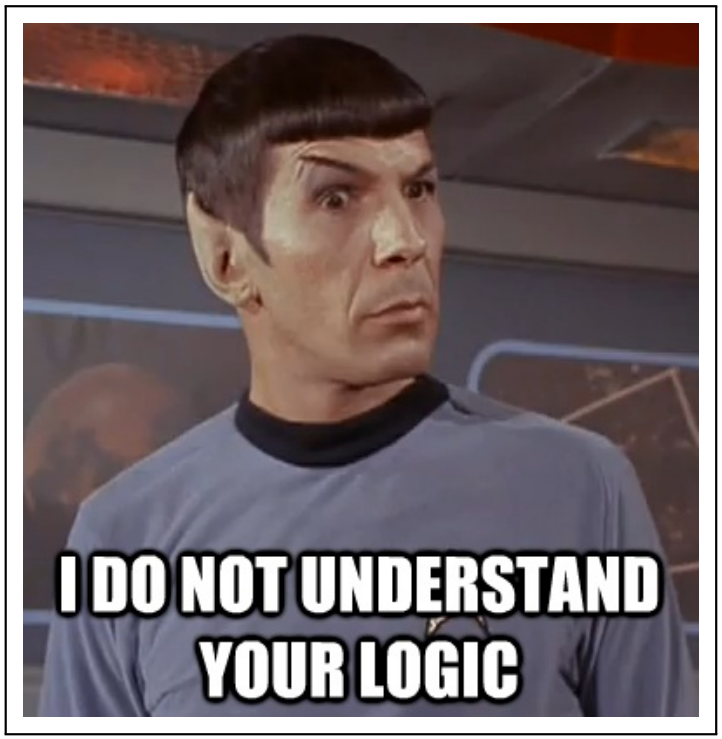



```
public Person setAge(int age) {
    this.age = age
    return this;
}
```

Program errors

Logic

- **Logical errors**
  - Compiles without error
  - Runs error-free until catastrophic failure
  - Or runs to completion but with incorrect results
  - Generally much more difficult to detect.



Logical errors

Unfortunately logical errors rule rather than exception

- Producing programs free logical errors extremely difficult
- Much non-trivial commercial software shipped with bugs
- Detection of logical bugs cannot be automated
- Some industries have better track record than others
  - Aerospace
  - Military

Ariane 5: A logical error

Software failures

Some examples software failures:

- Therac radiation therapy machine (c1985)
  - Massive overdose (100 times)
  - Three people died
- Mars orbiter disentegration (1998)
  - Ground computer used imperial units
  - Satellite used metric
  - Burned up
  - Loss: 300 million dollars
- London Ambulance Service (1992)
  - Newly commissioned system rapidly slowed to a crawl
  - System abandoned
  - Reputed cause: memory leak.
- Ariane 5 (1996)
  - Rocket downed by floating point error
  - Loss: 500 million dollars

Software failures

- Mariner 1 (1962)
  - NASA's first spacecraft
  - Off course minutes into flight
  - Safety officer hit auto-destruct
  - Cause: incorrectly transcribed math symbol.
- Year 2000 bug: a disaster averted
  - Disaster
    - Decision to continue producing not-fit-for-purpose code.
  - House of Commons report
    - Estimated repair cost: $400 billion
    - A cost to clients but revenue to software industry
- British NHS IT system (2013)
  - Patient record system
  - Project launched 2002
  - Now abandoned
  - Loss: €10 billion and climbing

Software quality

General product quality

- Fit for purpose
  - Possess appropriate functionality
- Merchantable quality
  - High quality processes and standards

Software quality

- Meet customers' needs
  - Easy to use
  - Correct results
  - Doesn't crash
  - Easy to debug and extend

Testing

Exhaustive testing not feasible

- One 64 bit variable
  - One nanosecond per test
  - 500 years required

Divide and conquer

- Different tests at different development stages
  - Unit test within a class
  - Modular test across classes
  - Integration test: check interfaces between modules
    - Consistent assumptions.
    - Correctly communicate.
  - Regression test

Customer acceptance test

Unit Testing

Unit test

- Test to determine if a unit of code behaves as designed
- Unit can be considered smallest testable part of program
- In our labs we test to the level of a method

```
/**
* Unit test
* Checks making appointment
* @return returns true if test succeeds else false
*/
public boolean twoHourAppointment() {
        Day day = new Day(1);
        Appointment appointm1 = new Appointment("Course board meet", 2);
        return day.makeAppointment(16, appointm1);
  }
```

Positive Unit Testing

A positive unit test

- Expected to return true on success

```
/**
* This is a positive unit test
* It returns true on success
* Makes appointment at 4 p.m. day 1 for a lab
*/
public boolean oneHourAppointment() {
    Day day = new Day(1);
    Appointment appointm1 = new Appointment("Lab", 1);
    return day.makeAppointment(16, appointm1);
  }
```

Negative Unit Testing

A negative unit test

- Expected to return false on success
- Tends to be overlooked but important to include

```
/**
* Attempted double booking
* This is a negative unit test
* We expect the method to return a false
* Returning true would indicate a bug
*/
public boolean doubleBooking()
{
    Day day = new Day(1);
    Appointment appointm1 = new Appointment("Java lecture", 1);
    Appointment appointm2 = new Appointment("Java lab", 1);
    day.makeAppointment(10, appointm1);//make booking at 10
    return day.makeAppointment(10, appointm2);//try 2nd booking at 10
}
```

Negative Unit Testing

Test *doubleBooking*

- Attempt to double book should not succeed
- *boolean false* expected when *doubleBooking* invoked
- Use JUnit *assertsEqual* to test for *false*

```
@Test
public void negativeTests()
{
    assertEquals(false, diaryTester.doubleBooking());
}
```

Regression Testing

Unit tests

- Performed on smallest testable portions
- Number unit tests grows as development progresses

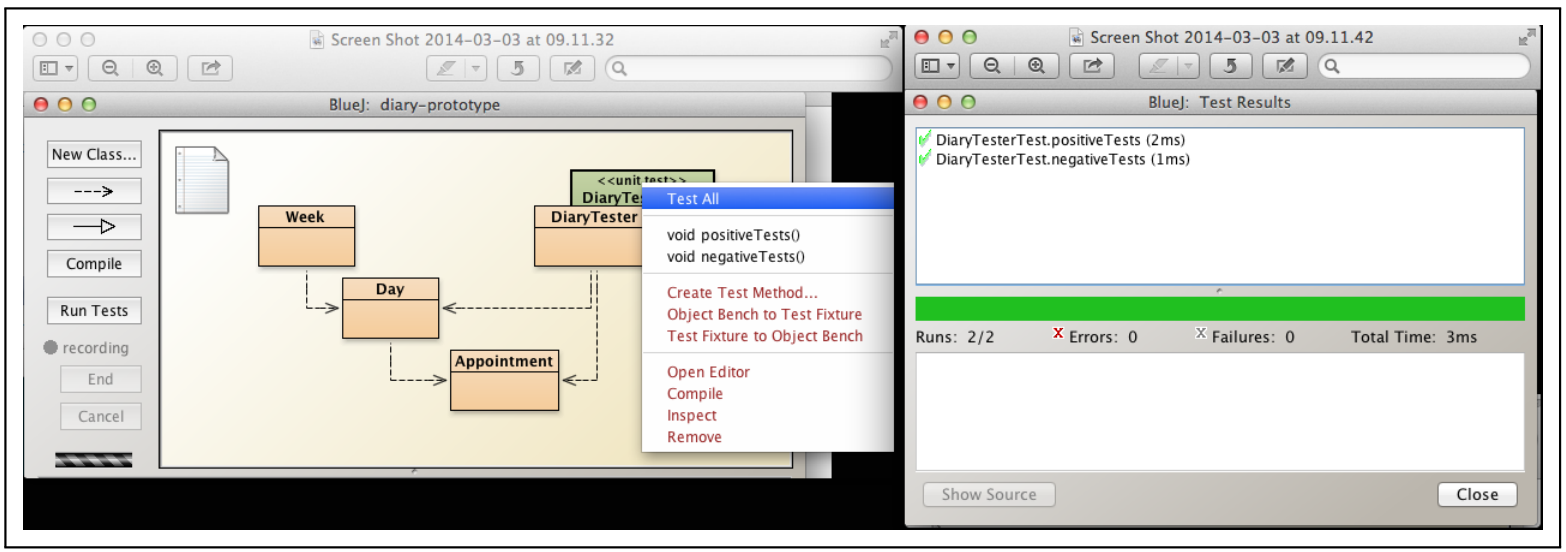
Regression testing

- Re-running suite of unit tests
  - As development progresses (daily)
  - At key milestones
  - Before shipping
  - Following bug fixes

JUnit Testing

JUnit

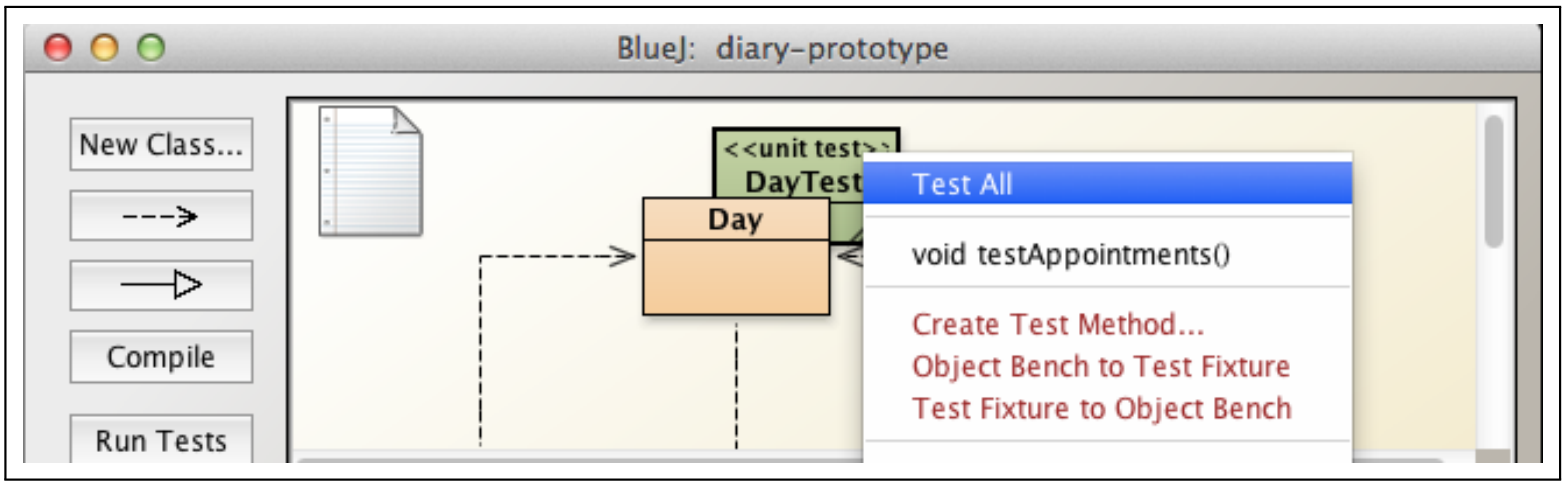
- Unit testing framework
- A framework can be considered
  - Reusable set of libraries or classes
  - Provider of some declared functionality
- JUnit widely used across different languages
- BlueJ has built-in JUnit



Automated Regression Testing

BlueJ framework includes JUnit

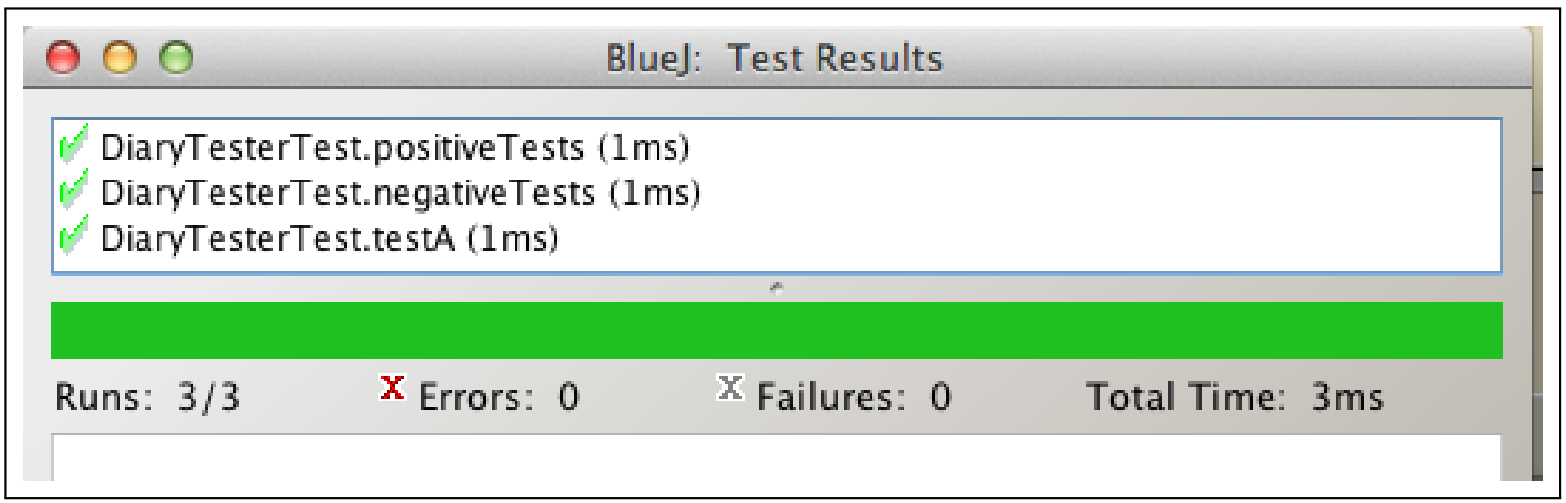
- For any class under development
  - Easy to create associated JUnit class
  - Write test methods in test class
  - Invoke individual test methods
  - Invoke testAll method



BlueJ JUnit Automated Regression Test

Run all tests example

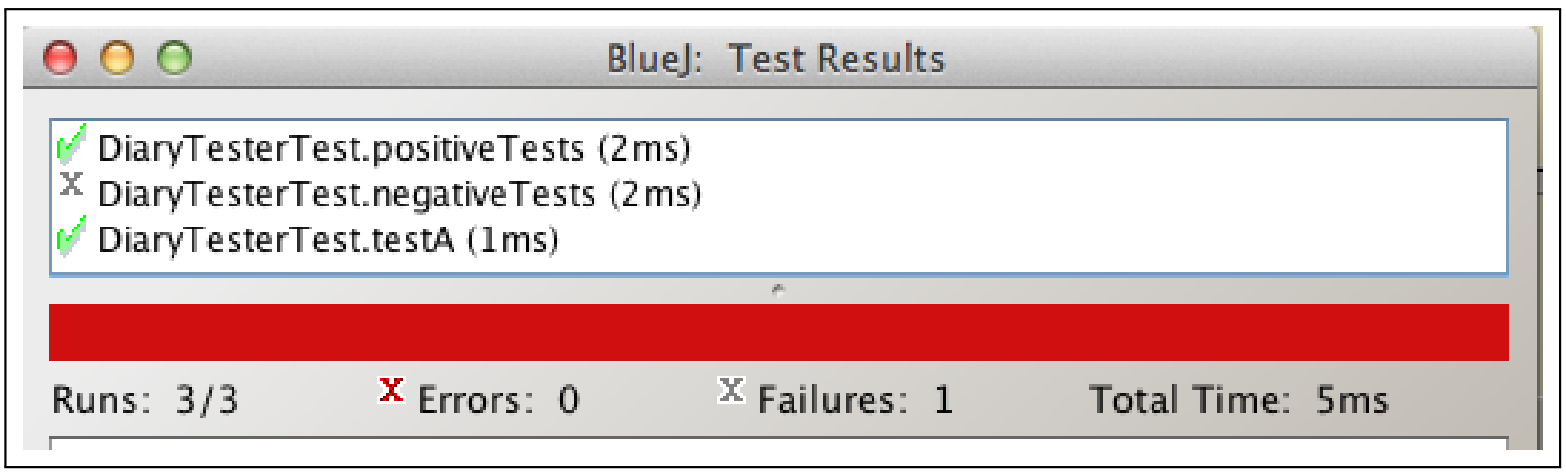
- Three methods invoked
  - positiveTests
  - negativeTests
  - individual testA
- Green progress bar indicates overall success



BlueJ JUnit Automated Regression Test

Run all tests example

- Individual and overall results published
  - positiveTests succeeded
  - negativeTests failed
  - individual testA succeeded
- Red progress bar indicates overall failure



Referenced Material

1. Epic failures: 11 infamous software bugs

`http://www.computerworld.com/s/article/9183580/Epic_failures_11_infamous_software_bugs`

[Accessed 2014-03-104]

2. The top 10 IT disasters of all time

`http://www.zdnet.com/the-top-10-it-disasters-of-all-time-3039290976/`

[Accessed 2014-03-104]

3. Wiener, L.R. (1993) Digital Woes. Perseus Books Groups.

4. Software Engineering: Ariane 5

`http://www.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html`

[Accessed 2014-03-05]

Referenced Material (continued)

5. edX BerkeleyX: CS169.1x Software as a Service

`https://courses.edx.org/courses/BerkeleyX/CS.CS169.1x/3T2013/8e8cf6e05c8f43749fbac0938f4acbaa/`

[Accessed 2014-03-05]

6. Abandoned NHS IT System has cost €10 billion so far

`http://www.theguardian.com/society/2013/sep/18/nhs-records-system-10bn`

[Accessed 2014-03-06]