# CSE 403

Testing, part II (Quality Assurance), bugs, and code reviews

# Announcements

- Projects
  - You've reviewed the last release -- know what's going right and wrong
  - Have a plan for release 2, due 11/19
  - You are executing
- Cost-benefit analysis
  - Your projects -- what are you going to do?
  - Your life -- "You can't have it all," S. Wienbar
  - Your midterm
  - Your homework assignments

# Homework assignment

- CODE: modify sort routine to allow for the return of a sorted list of all, valid, or invalid URLs.
- DESIGN your URL validation, including:
  - Specifying a definition for a valid URL
  - Defining a canonical form
  - Designing a system that decides if a URL is valid or not
  - Defining a canonicalizer
  - Defining a comparator operator for URLs (<, >, ==)

# Homework assignment

- Requirements: What is the specification of a valid URL?
- Design: Sorting and validating in the same module?
  - Good or bad?
  - Should you have a separate validator of a list of URLs?
  - What are the implications?
- Implementation and testing: Foreshadowing of what's next

# Release notes

- A description of what you just released
  - High level, thematic description
  - Major functionality
  - Exceptions of what might not be working
  - How to access the release
- Approximately a "README"
- "We're proud to announce release 1.4 of..."
- Largely for external consumption
- About 1 page

# Tile to the Top release notes

## Release Notes (November 5th)

New Page | Edit Page | Page History

Tile to the Top is located at: tiletothetop.herokuapp.com

### High Level Functionality

The major feature for this release was basic game functionality. This means that if a user visits our page, they are able to successfully play one full game of Tile To The Top. In addition, we have made it so that a user can play another game after they have finished the first one without reloading the page. For this to be the case, a slew of features needed to be implemented from the front end to the back end.

### Major Functionality

Front End Features:

- Definition Zone
- Tile Area / Tiles
- Tile Slot Area / Workspace
- Tile Moving / Locking
- End of Game Recognition / Correctness Checking for each word
- Score box - see notes in "Exceptions"

Back End Features:

- Dictionary (database)
- Webpage hosting
- Random Word Fetching Service

Bonus Features:

- Click and Type (Front End)

# Change My Mood release notes

## Client Side:

1. [FR] Added button xml and styled the app to match mock up. (hunlan & youngmin)
2. [FR] Added code to call and retrieve data from server. (hunlan)
3. [FR] Added Facebook sign in by Facebook SDK (hyunjoon)
4. [FR] About Us page with member's names and email addresses (hyunjoon)
5. [FR] 2 menu options (without sign in / with sign in) (hyunjoon)

## Server Side:

1. [FR] Added api method and backend logic to get media content. (hunlan)
2. [FR] Added test for get media content. (hunlan)
3. [FR] Added api method and backend logic to rank media content. (hunlan)
4. [FR] Added test for rank media content. (hunlan)
5. [FR] Added api method and backend logic to load database with up to 50 pictures (hunlan & Garrett)
6. [SR] Added api method and backend logic to filter database media content. (hunlan).
7. [SR] Setup cron job on GAE to talk to load and filter database periodically. (hunlan)

## Bugs/ Problems:

1. Testing for filter was not implemented yet because its a second release feature.
2. Improvements for search quality when filling up our database with media content
3. Facebook sign in should be reimplemented with sign out function

# The Grinf*ck

(tip of the day)

**Grinfock**: To tell someone, typically a superior, what they want to hear to get her/him “off your back” or because the truth is damaging.

(I may have gotton the spelling wrong.)

More:
http://www.bothsidesofthetable.com/2010/03/28/dont-be-a-grin-fucker/

# Points in Review

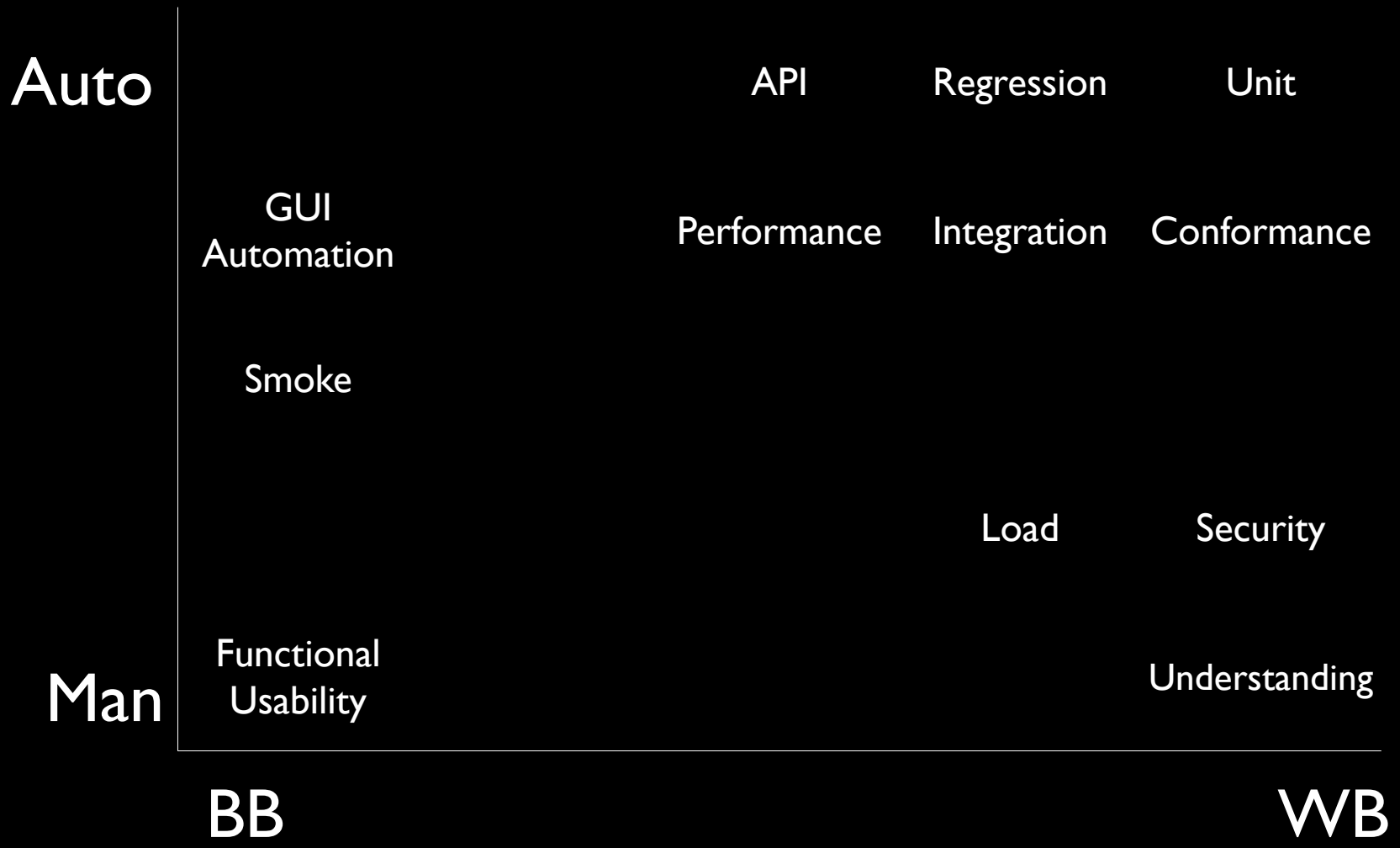
# Purpose

- To find bugs
- To enable and respond to change
- To understand and monitor performance
- To verify conformance with specifications
- To understand the functionality of code

# Classifying Testing

- There are many testing techniques -- a taxonomy is useful
- White box vs. Black box
- Automated vs. Manual

# Test Techniques



# Unit Tests

- Ensure completeness and correctness of implementation
- Cover all public APIs and significant internal APIs
- Begin writing tests as soon as basic structure and some APIs are known
- Regression tests: bugs as a source of test cases

# Functional Testing

- Humans (finally!)
- Written scripts and checklists
- Levels of detail: smoke, iteration, alpha, beta, release acceptance

# Summary

- Some testing is the developer's responsibility
- Automation makes powerful testing relatively simple

# More on testing (and bugs and code reviews)

# Types of testing

- Unit testing
- Regression testing (different definition that David talked about)
- Coverage testing
- Monkey testing
- Code review (is this testing?)
- Performance testing, load testing, scalability testing (maybe later)

# Unit testing drill down

# Let's start with a simple (famous) example

- **triangle** is a program that reads 3 integers from the command line that represent the lengths of three line segments. It returns (prints) the type of triangle that could be formed ("equilateral," "isoceles," or "scalene") from the segments or "false" if a triangle cannot be formed from the 3 segments
- %triangle 8 3 9 -> scalene
- %triangle 10 10 10 -> equilateral
- %triangle 7 1 2 -> false
- Define test cases for the triangle program

# What are some test cases?

# Here's some unit tests

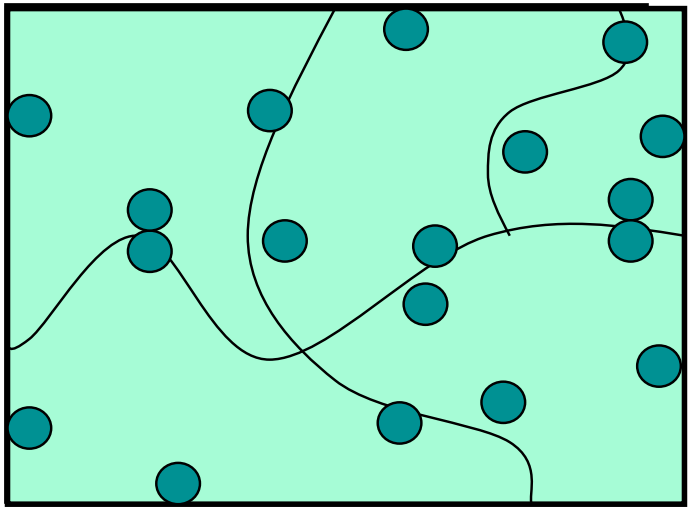
- triangle 5 5 5
- triangle 1 2 10
- triangle 7 7 2
- triangle 2 9 9
- 6 7 9
- triangle -3 5 6.5
- triangle 3 10 -1324
- triangle -3 -20 -30
- triangle 3423523523423534 2 423423523523532 5 2352352354235 4
- triangle a b c
- triangle alpha beta gamma
- triangle
- triangle 3 5 4 4

# Test cases

- How many?
- Can we test all cases?
- What tests do we want to run?
- How do we enumerate them?
- Can we partition the space?
- Tests can only reveal the presence of bugs/defects/problems -- not verify the correctness or absence of bugs

# Partition the input space to understand test cases you should write

- In theory, you only need to select one test in each partition
- In practice, this is hard -- problem specific
- I don't know how to do this!
- Edge cases are often the problem



# What's on the boundary? (i.e. what's an edge case?)

- Off by one errors
- Smallest, largest cases
- Zero, null, empty containers
- Null pointers
- Overflows in arithmetic
- Type errors
- Aliasing
- Exceeding maximum (buffer overflow)
- Circularity

# Example

Consider:

```
// returns: x < 0 => returns -x
// otherwise => returns x
int abs(x);
```

- You know nothing about the implementation
- So we are doing a black box test
- Maybe 2 partitions and an edge case { -3, 4, 0 }?
- All the tests pass
- Are we good?

# Example

Consider this buggy implementation:

```
// returns: x < 0 => returns -x
// otherwise => returns x
int abs(x) {
  if (x < -2)
    return -x;
  else
    return (x);
```

2 partitions and an one edge case { -3, 4, 0 }?

- All the tests pass
- But abs(-1) returns -1

# Back to triangle

```
import sys

def triangle(x, y, z):
    if(x<=0 or y<=0 or z<=0 or (x+y<=z or x+z<=y or y+z<=x)):
        print 'false'
    elif(x == y == z):
        print 'equilateral'
    elif(x==y or y==z or x==z):
        print 'isoscele'
    else:
        print 'scalene'

if __name__ == "__main__":
    main()
```

# Back to triangle

```
def main():
    print 'Line segments:',
str(sys.argv[1:])
    x = atoi(sys.argv[1])
    y = atoi(sys.argv[2])
    z = atoi(sys.argv[3])
    triangle(x, y, z)


if __name__ == "__main__":
    main()
```

# Test harness

- We'll use pytest
- You'll use the Django test execution framework?
- Can we really test the command line parsing?
- We'll test the triangle function code
  - Assuming we want to expose this interface

# Here's the test code

```
from triangle import triangle

def test_notTriangle_allZero(capsys):
  triangle(0, 0, 0)
  out, err = capsys.readouterr()
  assert out == "false\n"

def test_notTriangle_negative_one(capsys):
  triangle(-1, 0, 0)
  out, err = capsys.readouterr()
  assert out == "false\n"

def test_notTriangle_negative_two(capsys):
  triangle(0, -2, 0)
  out, err = capsys.readouterr()
  assert out == "false\n"
```

# Write unit tests for all these

- triangle 5 5 5
- triangle 1 2 10
- triangle 7 7 2
- triangle 2 9 9
- 6 7 9
- triangle -3 5 6.5
- triangle 3 10 -1324
- triangle -3 -20 -30
- triangle 3423523523423534 2 423423523523532 5 23523523542354
- triangle  a b c
- triangle alpha beta gamma
- triangle
- triangle 3  5 4 4

# Here are the test results

```
(env)igniter:play yamamoto$ py.test TriangleTest.py
============================= test session starts ==============================
platform darwin -- Python 2.7.2 -- pytest-2.3.2
collected 31 items

TriangleTest.py ...............................

=========================== 31 passed in 0.18 seconds ==========================
(env)igniter:play yamamoto$
```

# Type errors (in Python)

- triangle function expects 3 integer arguments
- Types aren't explicitly declared
- Who should catch these errors?
- What do the following calls return?
  - triangle('a','a','a')
  - triangle ('z','z','a')
  - triangle('a','z' 'z')
  - triangle('a', 0, 2)
  - triangle ('a', 1.2)

```
import sys

def triangle(x, y, z):
    if(x<=0 or y<=0 or z<=0
or (x+y<=z or x+z<=y or y
+z<=x)):
        print 'false'
    elif(x == y == z):
        print 'equilateral'
    elif(x==y or y==z or
x==z):
        print 'isoscele'
    else:
        print 'scalene'

if __name__ == "__main__":
    main()
```

# When should you write unit tests?

- Part of the design process -- helps you define the interface?
- After the design process
- After implementing
- After integration -- make sure there aren't many gaps
- After a bug is found

# Regression Test(ing)

# Regression testing

- A regression suite is a collection of unit tests that you run to test the sanity of the system
- Typically automated
- A regression test is a unit test in the regression suite
- Slightly different than David's definition -- a unit test after a bug a found
- Run regularly (daily) and irregularly (before release, after big code changes before checking in)
- Gives confidence that things are still working or you didn't break anything
- You can't aggressively "re-factor" unless you have a good regression suite*
- *...if your refactoring changes the interfaces against which you run your test, this doesn't help -- your tests are broken

# Monkey testing

# Monkey testing

- Slang for functional/feature testing?
- Manually testing the application by clicking on buttons, filling out fields, observing results
- Walk through all the features
  - New
  - Old (regression)
- On all the platforms (**browsers**, backends, databases, etc.)
- Do it every release?
- Can this be automated?

# Code coverage

- Collection of tests so that every line of executable code gets executed
- Sanity test
- Especially important in non-statically checked/ typed languages
- Exception paths are particularly interesting
- What are the success/failure conditions?
  - Uncaught exceptions, memory leaks, type errors(?)

# Bugs

# What are bugs?

- Algorithm, coding, type, logic errors for sure
- Incomplete/Missing functionality
- Mismatches between requirements and implementation
- Failure to meet strategic goals
- Ease of use difficulties
- User interface errors

# Categorizing bugs

- Severity : How "bad" is it?
- Priority: How important is it that we fix it (and when)
- Subsystem: Which module is the bug located, if known)
- Release: Which release(s) have the bug?
- Ownership: Who is responsible for fixing it?
- Status: Open, Assigned, Fixed, Closed
- Platforms: Operating systems, platform versions, browsers

# Writing bug reports

- Use a bug tracking tool
- Identify all the category information
- Written explanation of:
  - What the bug is
  - How to trigger it
  - Screen shots if available or applicable
  - Data to support trigger if appropriate
- Search for "good bug reports"; lots of good advice on the web

# To be continued...