

---

# CS1632, LECTURE 14: INTERACTING WITH STAKEHOLDERS

BILL LABOON



# TESTING THEORY

- So far in this class, we've stuck to the "hard science" part of testing: equivalence classes and boundary values, manual vs automated testing, unit testing syntax.
- All very important!

# THE "SOFT SCIENCES" ASPECTS OF TESTING

- As a tester, you're often going to be at odds with development, management, and other stakeholders.
- More than a developer, you're going to have to be prepared for conflict resolution with management.
- More than a manager, you'll be discussing edge cases in requirements with project analysts.
- More than an analyst, you'll be dealing with developers on holes in their code.

# PEOPLE SKILLS

- Software development as a whole is much more social than many non-developers think.
- The best software in the world will not be successful if nobody knows about it.
- Communication (of one form or another) is vital.

# WHAT IS A STAKEHOLDER?

A person or group who has a direct interest in the completion and/or execution of the system under development.

# KINDS OF STAKEHOLDERS

- **Customers** - The people who pay for the system.
- **Users** - The people who will actually use the system (not always the same as the customer!)
- **Project Management** - The people who are actively managing the project to completion.
- **Upper Management** - The people who care about the financial ROI (return on investment) of the development of the product.

# KINDS OF STAKEHOLDERS

- **Developers** - The people who write and maintain the software.
- **Testers** - The people who test the software for defects and quality assurance.
- **Support staff** - Administrators of the system and help desk.
- **Assessors** - The people who oversee the legal and regulatory aspects of the system.

# NOTE

Although I am specifically talking about software, the term "stakeholder" and the rest of the terminology I'm using is pretty standard across corporate project management.



# NOTE

A stakeholder is DIRECTLY, not INDIRECTLY, impacted by the completion and execution of the system.

True, the electrical worker is kind of impacted if more electricity is used to run your software, but they are not a stakeholder.

The janitor may have more pizza boxes to clean up if developers stay late working on the product, but they are not a stakeholder.

# INCLUDING RELEVANT STAKEHOLDERS

- Try to include all relevant stakeholders in the development process
- Software does not get developed in a vacuum.
- Even if you are developing it yourself, it's for a reason, even if it's just to learn a new technology.

## SPEAK THE LANGUAGE OF THE STAKEHOLDERS

- Software developers speak in terms of technology.
- Testers will speak in terms of quality.
- Project managers in terms of deadlines.
- Upper management in terms of financials.
- Care enough to know and understand what they care about. Ideally, you should care about it, as well!

# ENGAGE EARLY, ENGAGE OFTEN

- "Engagement" is more than just talking.
- Give the stakeholder a chance to let you know their opinions. Avoid coloring them with your own.
- Get their feedback early on in the process and as often as possible.

## BE EXPLICIT ABOUT PROBLEM, BENEFITS, DRAWBACKS

- Be as clear as possible when giving status updates.
- It is always better to let stakeholders know of problems early.
- If you're ever wondering whether or not you should communicate something, do it.

# IDENTIFY AND MANAGE RISK

- Be upfront with the fact that there are certain factors beyond your - or sometimes anyone's - control.
- You can *mitigate* but never *avoid* risk.

# EXAMPLES

- **Upper Management** - How does this impact the financials of the product?
- **Product Management** - How does this impact the scope or timeline of the project?
- **Users** - How does this impact the functionality important to me?
- **Customers** - How does this impact what I'm getting or how much I'm paying?

# BE SUCCINCT, BUT PREPARED TO DIVE DOWN

- Keep in mind how much time and energy the person to whom you are reporting has for this topic.
- However, be prepared for questions.



# A TEMPLATE FOR DEALING WITH MANAGEMENT

- **Red** = This system or subsystem has extremely serious issues which will almost certainly impact the timeline or financials unless they are remediated.
- **Yellow** = There are some warning flags, but they can be dealt with. Some outside help may be necessary.
- **Green** = Everything is A-OK, or there are only minor issues.

# EXAMPLE

**Login/Security System** One minor SQL injection issue found, currently being worked.

**Database Storage** Latency on sharded Item databases is very high. Will require some research to fix.

**User Interface** No non-trivial issues. On schedule.

**System Performance** All KPIs in nominal range.


**System Administration** Some shell scripts left to write and automate, but on schedule.

# EXAMPLE

- **Login / Security System** - Numerous flaws. Outsiders have achieved superuser access thirteen times last week.
- **Database System** - System does not scale past 100 bytes of storage.
- **User Interface** - Choice of vi or Emacs keybindings popular among Computer Science students, but not for our target audience of first and second graders.
- **System Performance** - Web page currently takes 20 minutes to render, and no known way to remove the 27 MB of JavaScript currently being loaded.
- **System Administration** - Currently all tasks are being done manually by one person, who has repeatedly threatened to quit and has refused to document any processes.

# YOU ARE IN CHARGE OF TESTING A NEW FUNCTION

- **SYS-ACOUNT** - The subsystem shall accept text and return the number of A's in the text.



Any questions?

# QUESTIONS

1. Is the string ASCII, UTF-8, EBCDIC, other?
2. Does this count capital and lowercase?
3. What about a's with accent marks?
4. Are there performance requirements?
5. How does the text get in (network, Unix pipe, prompt)?
6. What's the return type? (int, BigInt)
7. Does it need to be distributed?
8. What size data is going in?
9. What architectures does it need to run on?
10. Is the data coming from a trusted source?
11. What to do in the case of exceptions?
12. Do I need to worry about thread safety?

# MORE COMMUNICATION IS BETTER

You should develop a good relationship with project managers, customers or users (if appropriate), and developers.

# ON BEING THE BEARER OF BAD NEWS

- Be prepared to give bad news without being a pessimist.
- Testers often see a system at its worst. Don't let this color your view of the product.
- Be respectful when clarifying requirements. Don't belittle people not being clear... it's difficult work.





CS1632, Lecture 14.5:  
Testing Strategy and the  
Process of Quality

Bill Laboon

# Putting It All Together

---

- So far, discussed elements in isolation
  - Unit testing
  - Systems testing
  - Performance and Non-Functional Testing
  - Combinatorial Testing
  - Property-based Testing

# Just Like In Programming, Scale Is Where It Gets Difficult!

---

// Understanding how to unit test is simple:

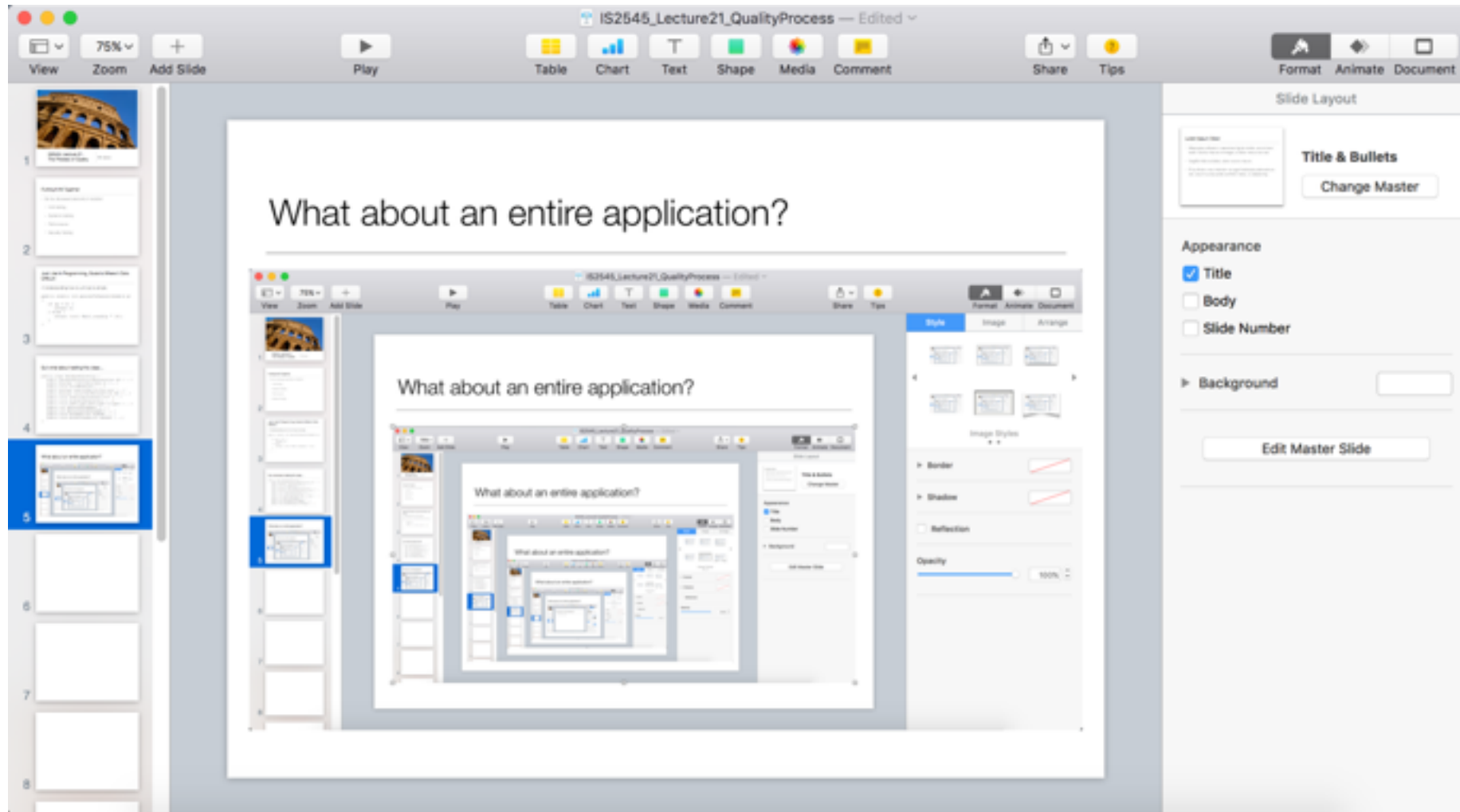
```
def pounds_to_ounces(p)
  if (p < 0)
    0
  else
    (p * 16).round
  end
end
```

# But what about testing this class...

---

```
class DatabaseConverter
  def initialize(db_connection) . . .
  def insert_data(data) . . .
  def force_reshard . . .
  def execute_sql(sql_string) . . .
  def select_db(db_connection) . . .
  def revert_last_transaction . . .
  def unrevert_reversion . . .
  def add_trigger(table, trigger) . . .
  def current_db_num . . .
  def current_db_threads() . . .
  def set_db_num(new_num) . . .
  def set_db_threads(new_num_threads) . . .
end
```

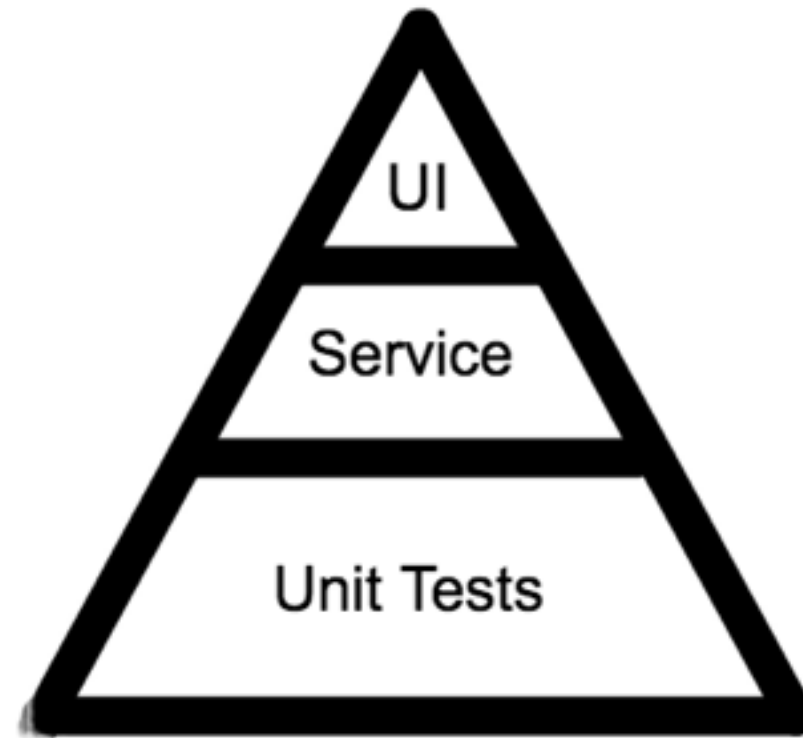
# What about an entire application?



# The Testing Pyramid

---

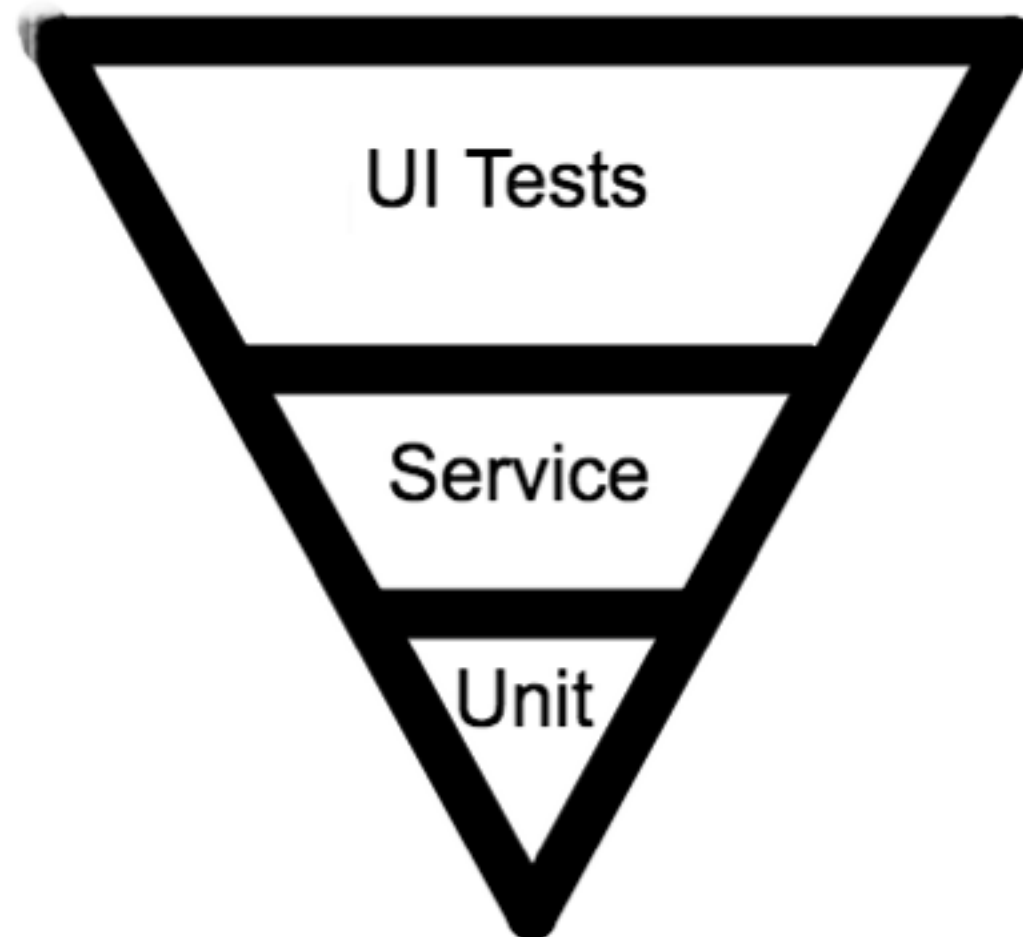
- 10% UI Tests
  - Tests which check the whole system end-to-end
- 20% Service Tests
- 70% Unit Tests



# Ice Cream Cone Anti-Pattern

---

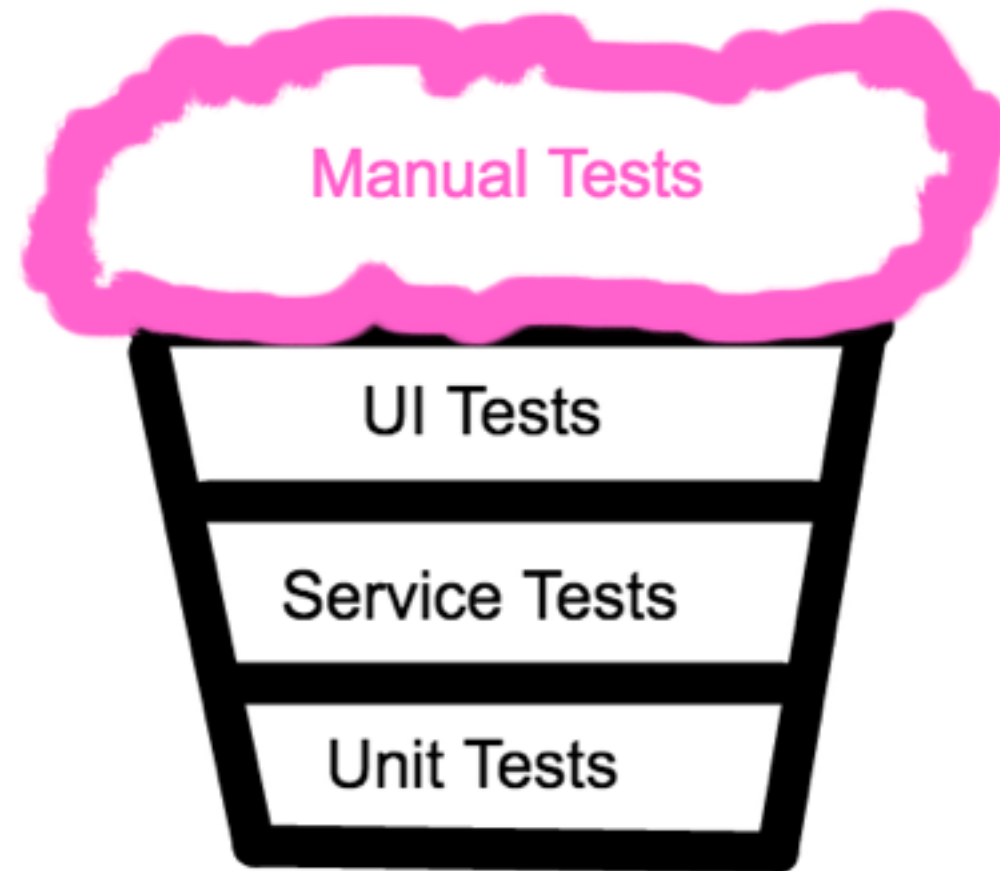
- Few Unit Tests, Some Service Tests, Many UI Tests



# Cupcake Anti-Pattern

---

- Lots of Manual Tests On Top





# Continuous Integration

---

- Avoid Big Bang integration
- Run tests automatically before merging
  - Or perhaps every time you commit/push
- Finds bugs earlier
- Ensures no failing tests in the master branch

# What About Other Kinds of Tests?

---

- These will often fit in one of the kinds of test categories
- Other kinds of tests are done on as-needed basis and the amount necessary will vary by domain
- Test coverage and weighting will vary by domain
  - Other tests/quality processes might also be necessary! Usability, formal verification, code metrics, etc.

## Example: Performance Testing

---

- Full Application performance testing: UI test
- Component testing: service test
- Performance of a function: unit test

## Example: Security Testing

---

- Penetration testing: UI test
- Testing a subsystem for injection attacks: service test
- Checking against buffer overflow in a method: unit test

## Example: Combinatorial Testing

---

- Checking OS / Browser / RAM combinations: UI tests
- Combinations of Microservices Loading: Service tests
- Checking for results of boolean args: Unit tests

# Designing a Testing Strategy

---

- Think about what are the priorities of this application
- Where would defects be worst?
- What are likely issues?
- Which have been found by similar applications?
- What tools will be useful?
- What is our team like? Do we need outside help?