Object Oriented Testing and Test-Driven Development

Based on notes from James Gain

(jgain@cs.uct.ac.za)

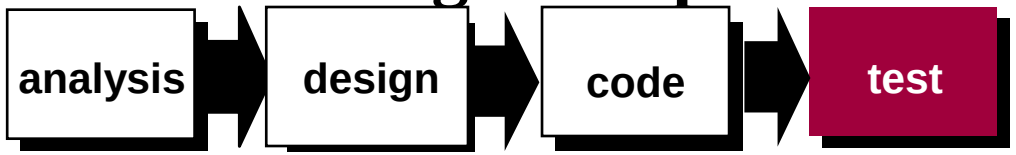
Larman, chapter 21

and notes on Larman from George Blank of NJIT

plus Glenn Blank's elaborations and expansions

Objectives

- To discuss when testing takes place in the life cycle



  - Test-driven development advocates early testing!
- To cover the strategies and tools associated with object oriented testing
  - Analysis and Design Testing
  - Class Tests
  - Integration Tests
  - Validation Tests
  - System Tests
- To discuss test plans and execution for projects

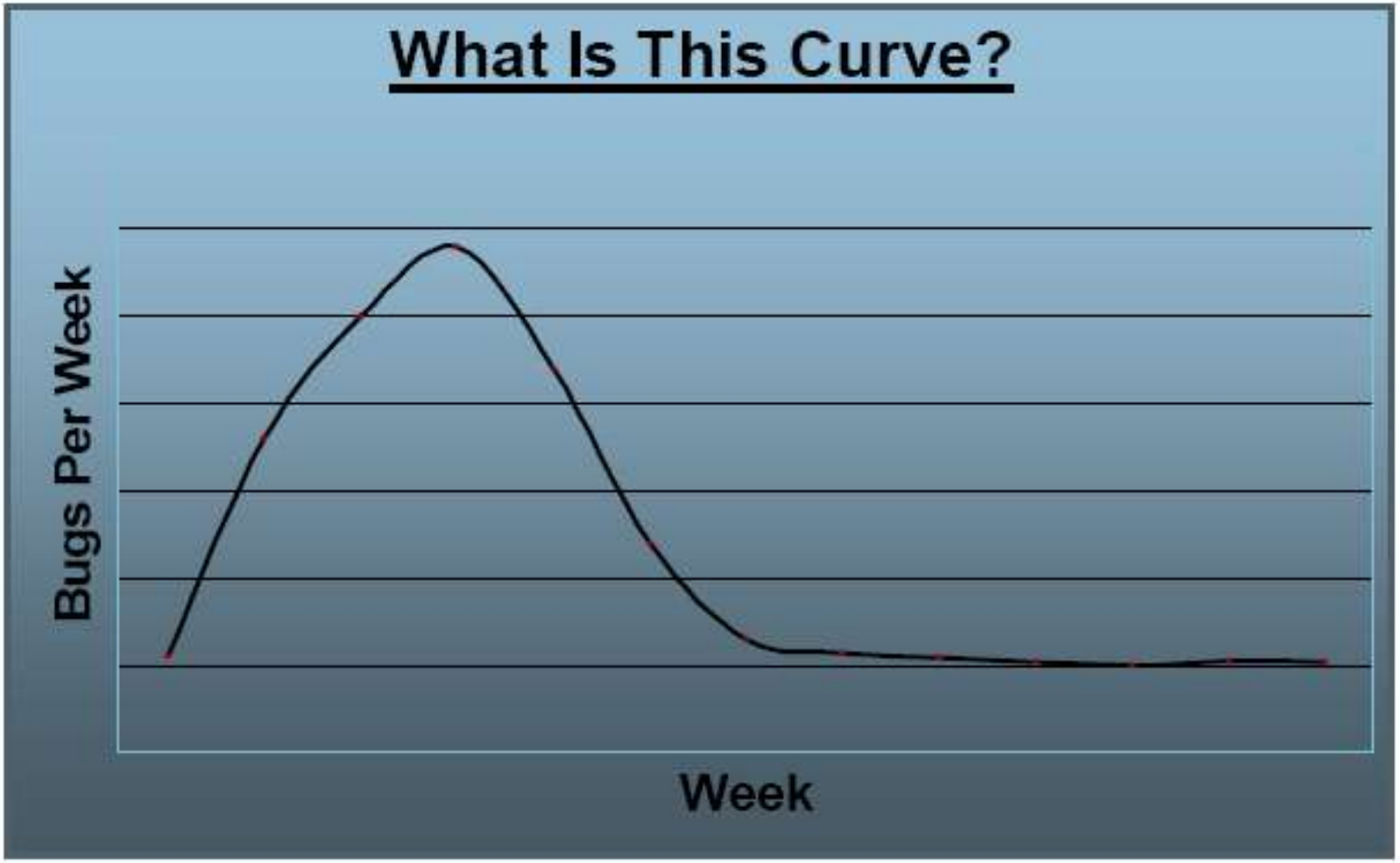
Object-Oriented Testing

- ***When should testing begin?***
- **Analysis and Design:**
  - **Testing begins by evaluating the OOA and OOD models**
  - ***How do we test OOA models (requirements and use cases)?***
  - ***How do we test OOD models (class and sequence diagrams)?***
  - **Structured walk-throughs, prototypes**
  - **Formal reviews of correctness, completeness and consistency**
- **Programming:**
  - ***How does OO make testing different from procedural programming?***
  - **Concept of a 'unit' broadens due to class encapsulation**
  - **Integration focuses on *classes* and their context of a use case scenario or their execution across a thread**
  - **Validation may still use conventional black box methods**

Test-driven programming

- eXtreme Programming (XP) advocates writing tests for units before writing actual code for units
- Why might this practice be a good idea?
- Constrains code to design: *How so?*
  - Design -> Test -> Code … in small iterations
- Promotes validation and reliability: *Why?*
  - Always rerun all tests (easier with automated testing) before integrating new code in a release
- Increases confidence to change code: *Why?*
  - Changes shouldn't break old code if you can test old code
  - Creed of XP: "embrace change"

The Bug Curve



Criteria for Completion of Testing

- When are we done testing? (Are we there yet?)
- How to answer this question is still a research question

1. One view: testing is never done… the burden simply shifts from the developer to the customer
2. Or: testing is done when you run out of time or money
3. Or use a statistical model:
   - Assume that errors decay logarithmically with testing time
   - Measure the number of errors in a unit period
   - Fit these measurements to a logarithmic curve
   - Can then say: "with our experimentally valid statistical model we have done sufficient testing to say that with 95% confidence the probability of 1000 CPU hours of failure free operation is at least 0.995"

YAHOO!


OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
S. Adams E-mail: SCOTTADAMS@AOL.COM
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!

Strategic Issues

- **Issues for a successful software testing strategy:**
  - **Specify product requirements long before testing commences**
    For example: portability, maintainability, usability
    Do so in a manner that is **unambiguous** and **quantifiable**
  - **Understand the users of the software, with use cases**
  - **Develop a testing plan that emphasizes "rapid cycle testing"**
    Get quick feedback from a series of small incremental tests
  - **Build robust software that is designed to test itself**
    Use assertions, exception handling and automated testing tools (Junit).
  - **Conduct formal technical reviews to assess test strategy and test cases -** "Who watches the watchers?"

Testing Analysis and Design

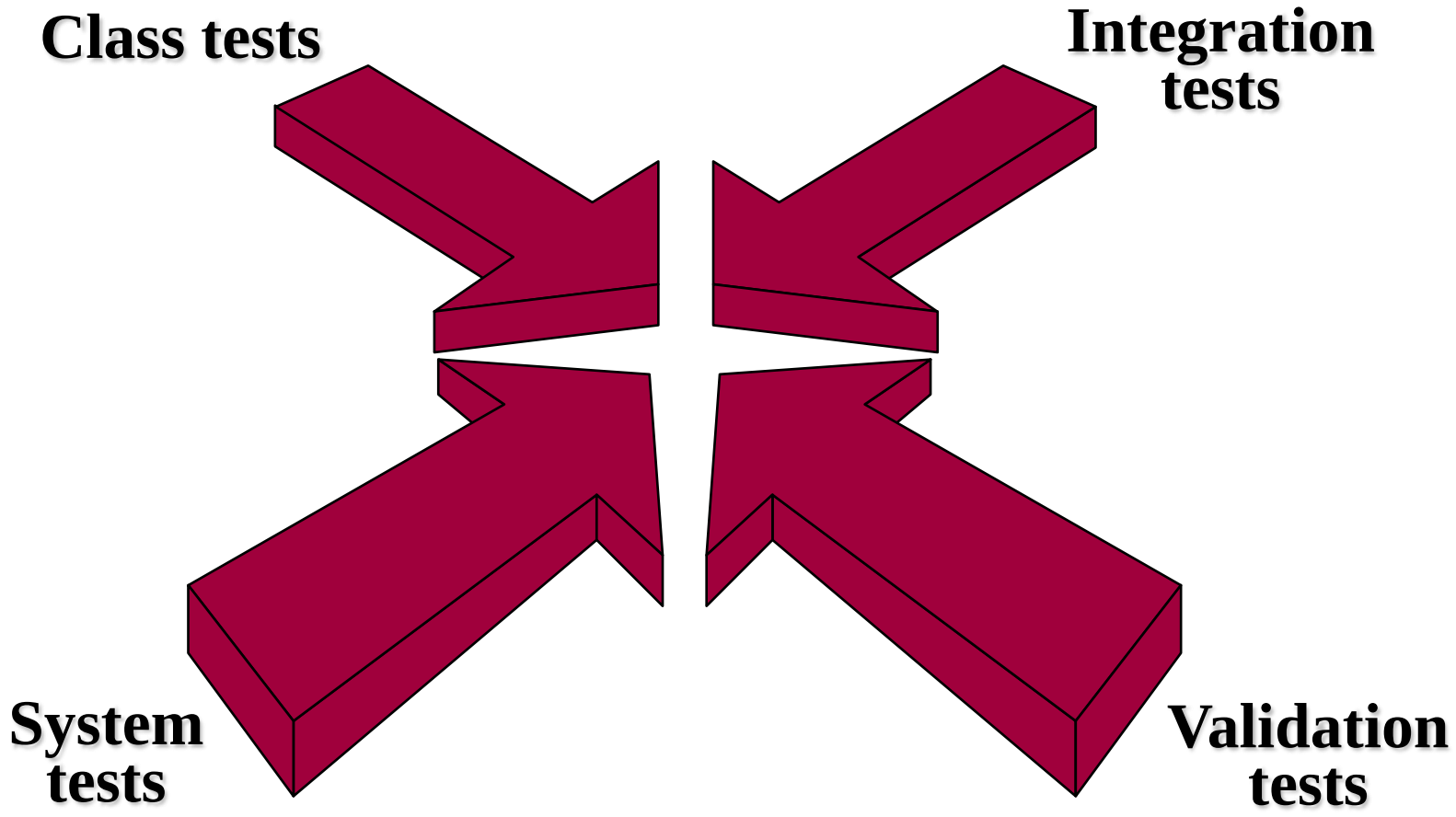
- **Syntactic correctness:**
  - **Are UML and ADT notation used correctly?**
- **Semantic correctness:**
  - **Does the model reflect the real world problem?**
  - **Is UML used as intended by its designers?**
  - **Is the ADT design complete (capturing all the classes and operations in UML diagram) and understandable?**
- **Testing for consistency:**
  - **An inconsistent model has representations in one part that are not reflected in other portions of the model**

Testing the Class Model

1. **Revisit the Use Cases, CRC cards and UML class model. Check that all collaborations are properly represented. Inspect the description of each CRC index card to make sure a delegated responsibility is part of the collaborator's definition.**
   - **Example: in a point of sale system.**
   - **A *read credit card* responsibility of a *credit sale* class is accomplished if satisfied by a *credit card* collaborator**
2. **Invert connections to ensure that each collaborator asked for a service is receiving requests from a reasonable source**
   - **Example: a credit card being asked for a purchase amount**

- **Have you tested your analysis and design?**
  - **If not, who will do it?**

Testing OO Code



[1] Class (Unit) Testing

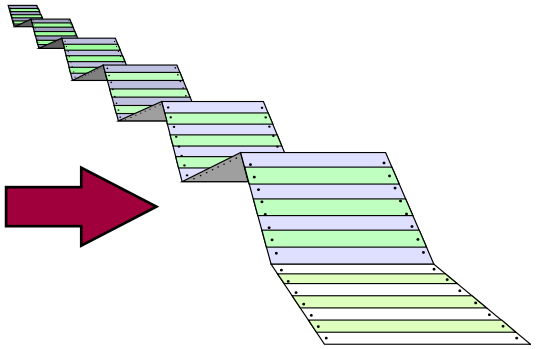
- **Smallest testable unit is the encapsulated class**
- **Test each operation as part of a class hierarchy because its class hierarchy defines its context of use**
- **Approach:**
  - **Test each method (and constructor) within a class**
  - **Test the state behavior (attributes) of the class between methods**
- ***How is class testing different from conventional testing?***
- **Conventional testing focuses on input-process-output, whereas class testing focuses on each method, then designing sequences of methods to exercise states of a class**
- **But white-box testing can still be applied**

Class Testing Process

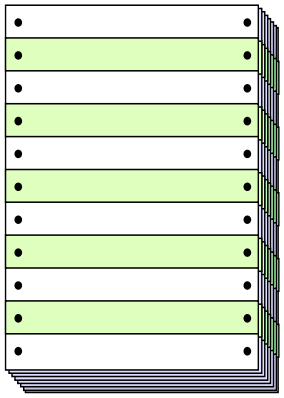
software engineer

How to test?

class to be tested



results



test cases

Why a loop?

Class Test Case Design

1. **Identify each test case uniquely**
   - Associate test case explicitly with the class and/or method to be tested
2. **State the purpose of the test**
3. **Each test case should contain:**
   a. **A list of messages and operations that will be exercised as a consequence of the test**
   b. **A list of exceptions that may occur as the object is tested**
   c. **A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)**
   d. **Supplementary information that will aid in understanding or implementing the test**

– **Automated unit testing tools facilitate these requirements**

Challenges of Class Testing

- **Encapsulation:**
  - Difficult to obtain a snapshot of a class without building extra methods which display the classes' state
- **Inheritance and polymorphism:**
  - Each new context of use (subclass) requires re-testing because a method may be implemented differently (polymorphism).
  - Other unaltered methods within the subclass may use the redefined method and need to be tested
- **White box tests:**
  - Basis path, condition, data flow and loop tests can all apply to individual methods, but don't test interactions between methods

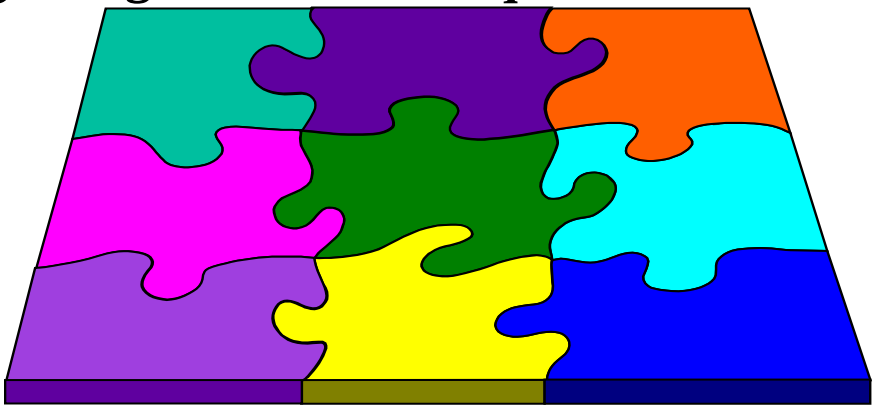
Random Class Testing

1. Identify methods applicable to a class
2. Define constraints on their use – e.g. the class must always be initialized first
3. Identify a minimum test sequence – an operation sequence that defines the minimum life history of the class
4. Generate a variety of random (but valid) test sequences – this exercises more complex class instance life histories

- Example:
  1. An account class in a banking application has *open*, *setup*, *deposit*, *withdraw*, *balance*, *summarize* and *close* methods
  2. The account must be opened first and closed on completion
  3. *Open – setup – deposit – withdraw – close*
  4. *Open – setup – deposit –* [deposit | withdraw | balance | summarize] – withdraw – close*. Generate random test sequences using this template

[2] Integration Testing

- **OO does not have a hierarchical control structure so conventional top-down and bottom-up integration tests have little meaning**
- **Integration applied three different incremental strategies:**
  - **Thread-based testing: integrates classes required to respond to one input or event**
  - **Use-based testing: integrates classes required by one use case**
  - **Cluster testing: integrates classes required to demonstrate one collaboration**



- **What integration testing strategies will you use?**

Random Integration Testing

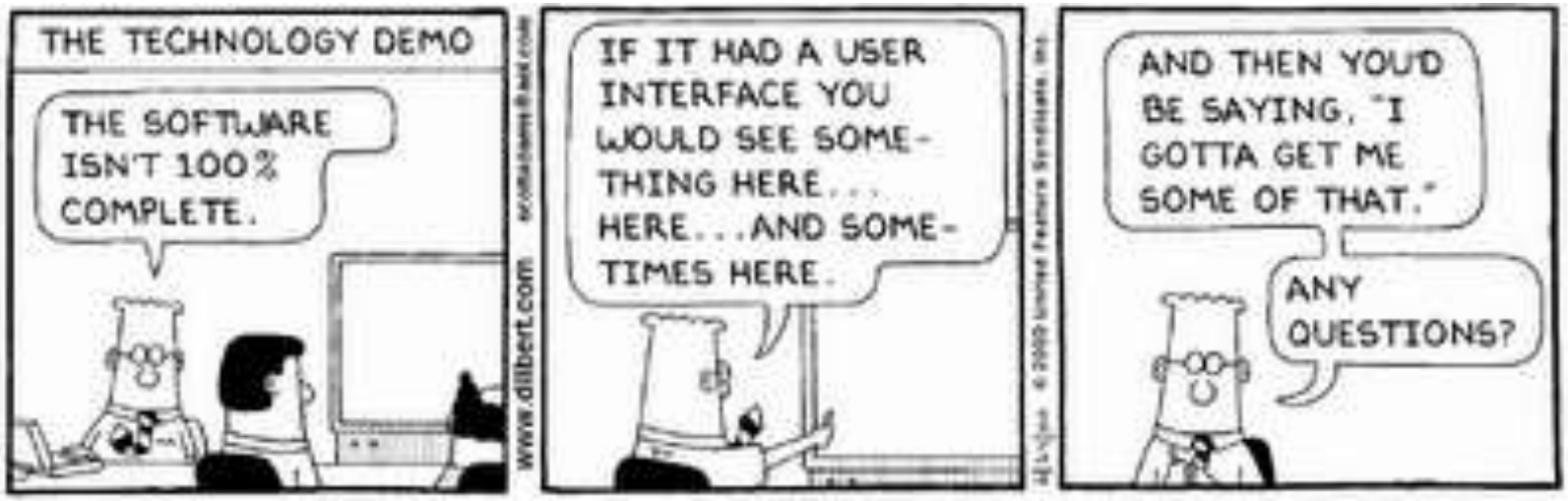
Multiple Class Random Testing

1. For each client class, use the list of class methods to generate a series of random test sequences. Methods will send messages to other server classes.
2. For each message that is generated, determine the collaborating class and the corresponding method in the server object.
3. For each method in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits
4. For each of the messages, determine the next level of methods that are invoked and incorporate these into the test sequence

[3] Validation Testing

- Are we building the right product?
- Validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- Focus on user-visible actions and user-recognizable outputs
- Details of class connections disappear at this level
- Apply:
  - Use-case scenarios from the software requirements spec
  - Black-box testing to create a deficiency list
  - Acceptance tests through alpha (at developer's site) and beta (at customer's site) testing with actual customers
- How will you validate your term product?

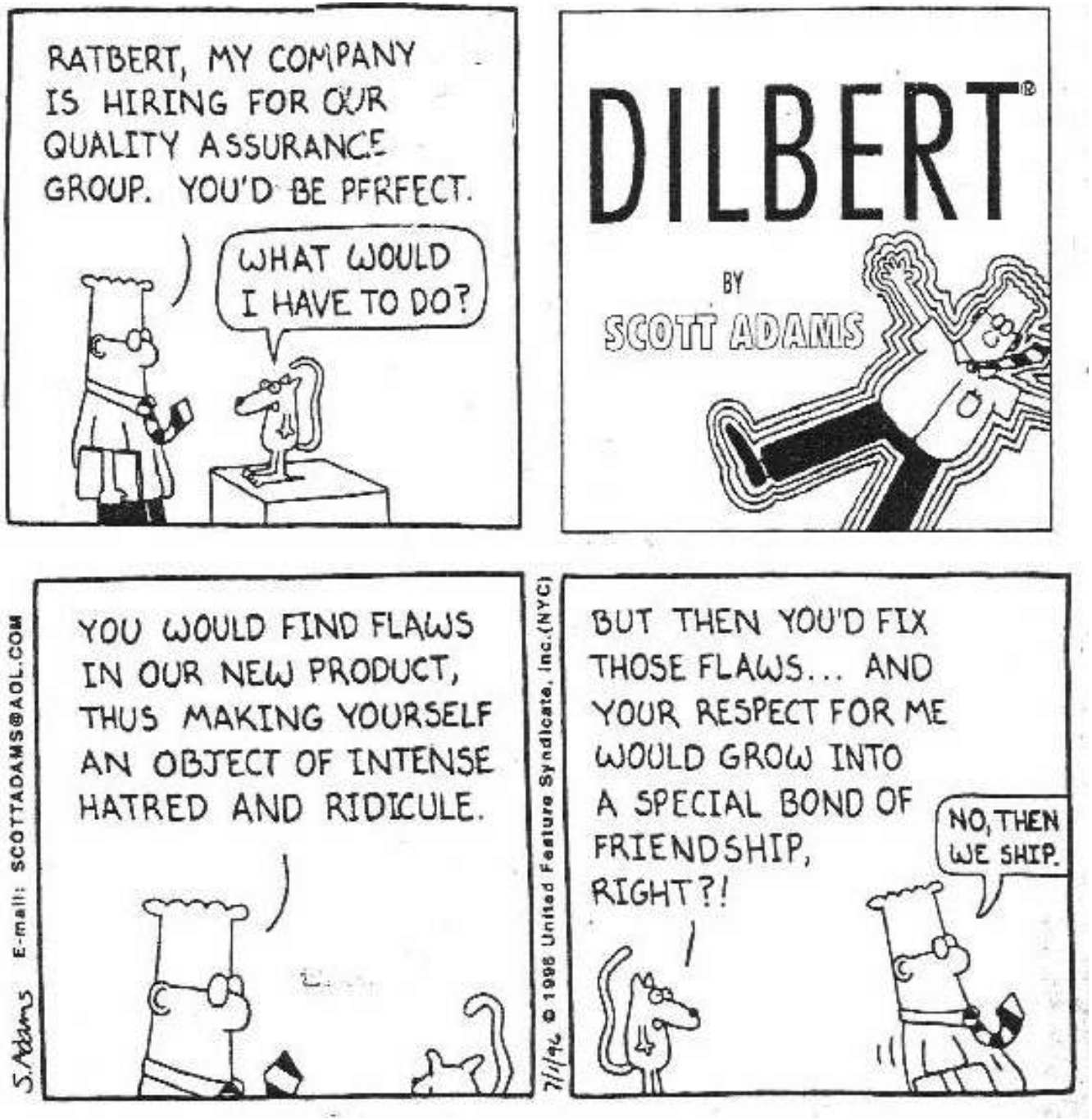
Acceptance testing, anyone?



[4] System Testing

- **Software may be part of a larger system. This often leads to "finger pointing" by other system dev teams**
- **Finger pointing defence:**
  1. **Design error-handling paths that test external information**
  2. **Conduct a series of tests that simulate bad data**
  3. **Record the results of tests to use as evidence**
- **Types of System Testing:**
  - **Recovery testing: how well and quickly does the system recover from faults**
  - **Security testing: verify that protection mechanisms built into the system will protect from unauthorized access (hackers, disgruntled employees, fraudsters)**
  - **Stress testing: place abnormal load on the system**
  - **Performance testing: investigate the run-time performance within the context of an integrated system**

SEMG



Can we do better?

Testing Summary

- Testing affects all stages of software engineering cycle
- One strategy is a bottom-up approach – class, integration, validation and system level testing
- XP advocates test-driven development: plan tests before you write any code, then test any changes
- Other techniques:
  - white box (look into technical internal details)
  - black box (view the external behaviour)
  - debugging (systematic cause elimination approach is best)