Software Testing Process Overview

CSCI E-19 SOFTWARE TESTING AND TEST-DRIVEN DEVELOPMENT

DR. ALINE YURIK

HARVARD UNIVERSITY EXTENSION SCHOOL

Testing Concepts

- What is the goal of software testing?
  - Find Problems in a Program?
  - Verify that a program does what it is supposed to do and does not do what it is not supposed to do?
- This traditional approach to testing puts developers and testers in separate camps:
  - Developers aim to write working software
  - Testers aim to find problems in software
- Test-driven development
  - The majority of testing efforts is placed on developers
  - Google and a number of other companies have adopted this approach

Defects

- Problems found during testing are referred to as:
  - Defects
  - Bugs
  - Incidents
  - These terms are interchangeable, we will use the term “defect” in this class
- What is a defect?
  - A problem in program implementation / error in the code
  - A missed requirement
  - A requirement implemented not as originally intended
- Defects are classified by severity/priority and recorded in a defect-tracking system

Retrieved from: http://qatestlab.com/knowledge-center/software-testing-glossary/defect/

Defect Information

- When recording a defect, the following information is captured in a defect tracking system:
  - Defect title
  - Detailed description
  - Severity/Priority
  - Screenshots or description of steps showing how to arrive at the problem
  - Environment defect was found in: development, system test, user acceptance test, production
  - OS and browser information
  - Release or build number
- Defect information should be sufficient for a developer to reproduce the defect without contacting the tester

Defect Severity/Priority

- These are common definitions of defect severities or priorities (terms severity and priority are often used interchangeably)
  - P1 – showstopper
    - May cause a system crash or make a large part of functionality unusable
  - P2 – critical
    - May be a serious problem, for example not being able to add an item to the shopping cart
  - P3 – major
    - May be not computing taxes on the sale correctly or not producing the correct total for the sale, or not being able to update item quantities in the shopping cart
  - P4 – minor
    - May be visual issues, such as poor alignment, misspelled words, incorrect wording

Source: http://www.dilbert.com

Class Checkpoint: What Defects Can We Tolerate?

- What are some examples of defects that may be tolerable in a content-based web site, such as a University web site? How about CNN? What types of defects may be “showstoppers” for these web sites?
- How about tolerable and showstopper defects for a financial or banking web site? Tax preparation software?
- What happens when we consider defects that may be tolerable in a medical device? An airplane or car navigation system? What is the impact of showstopper defects in these systems?

Testing Definitions and Approaches

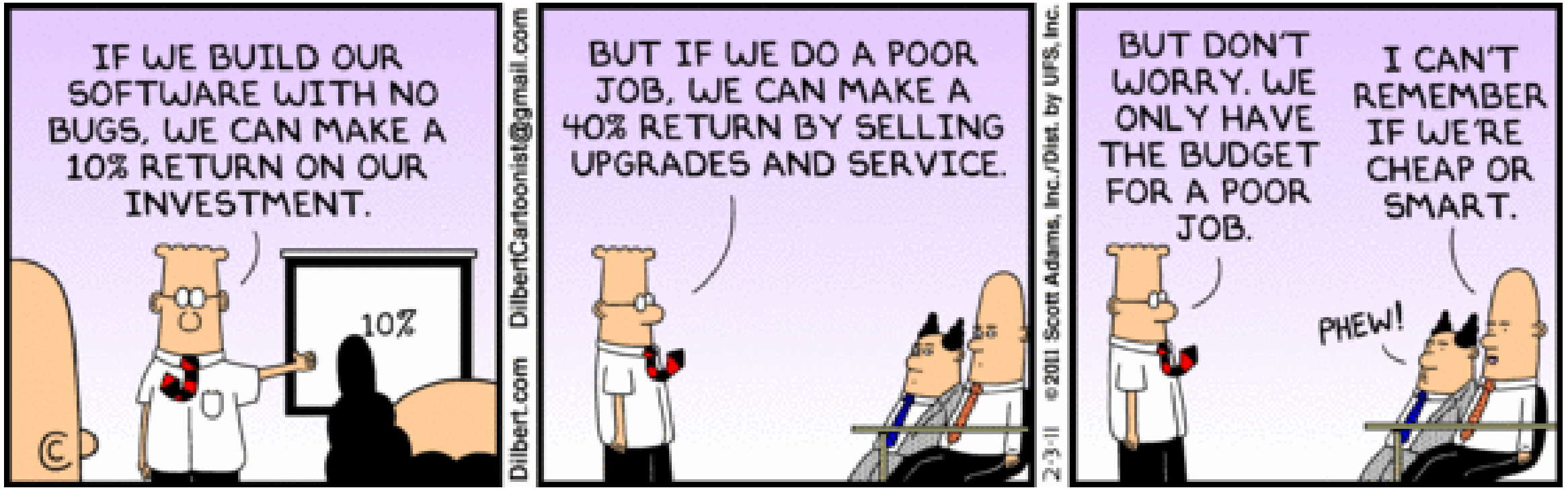
- Positive Testing
  - Evaluate program capability and determine if it works as expected per specification
- Negative Testing
  - Find defects in a program
- Testing in terms of Risk
  - Determine benefits and risks associated with release of the software
  - Risk evaluation:
    - Business, safety, security criticality of the software
    - Commercial/pubic visibility

Testing: Verification and Validation

- Verification:
  - Specific software development practices and process requirements have been fulfilled
  - “Are we building the product right?”
  - Verify that expected software development processes have been followed
- Validation
  - Software is meeting specified requirements for intended use
  - “Are we building the right product?”
  - Validate that the software is meeting its formal requirements

What Happens When Software Is Not Thoroughly Tested?

- Undiscovered defects and incorrect/untested functionality can lead to:
  - Software or web site crashes
  - Incorrect behavior
    - financial mistakes,
    - losing transactions or data,
    - incorrect calculations or actions (can lead to engineering or medical mistakes)
  - Reduced software performance
    - Slowness or
    - Inability to handle higher volumes of transactions or users



Source: http://www.dilbert.com

Class Checkpoint: Impact of Software Failures

- Can you think of some high-profile cases of software failures and problems due defects or incomplete testing?
- Have you experienced serious/high-impact issues on a software project due to missed defects or incomplete testing?
  - Can you share some examples?
- Do you think additional or more thorough testing may have discovered these issues?
  - If not, what are some of the other safeguards that could have been used to prevent or identify these failures prior to software launch?

Balancing Testing Needs

- While it is theoretically possible to find “all defects”, we are more likely to be able to find all/most defects in a simple program
- The more complex a software system is, the harder it is to ensure that the system is mostly “bug-free”
- Complexity of testing also increases when a system:
  - is not standalone, but may be interacting with other systems,
  - has a complex architecture and many components,
  - runs on a variety of devices, operating systems and platforms
- Increased complexity of testing = higher costs and longer project delivery time
  - Systems that need to be close to “bug-free” will need a longer testing cycle and have higher testing costs

Source: http://www.dilbert.com

When Do We Feel Confident to Release a Product?

- Testing process should give enough confidence that major paths and use cases through the system function as expected
- We also look for successful testing of each of the system requirements
- Ideally the test cases include a variety of data sets and user action combinations
- We also want to try out various extreme situations – high volume of transaction, extreme data values, high user volume, security-related considerations, and any other situations that may be specific to a given project

Test-Driven Development Approach – Google Experience

- Through this term we will also be looking at test-driven development model used at Google and also at a number of other companies
- Test-Driven development blends development and testing together
  - “Develop a little and test a little” approach
  - Instead of leaving testing to QA engineers, testing is the driving force of the entire development process
- Quality and testing is considered to be a prevention activity, done during development
  - Traditionally QA is a detection activity, done after development completes

Testing Roles in Test-Driven Development: Software Engineer

- Software Engineer (SWE)
  - Traditional development role, which in addition to design and development activities also writes test code for the modules SWE works on
  - Responsible for Test-Driven Design (TDD), unit tests, and creation of system tests for his/her modules
  - Responsible for overall quality for any code he/she worked on (created, modified, fixed)
- How is this different from the traditional approach to division of work between development and QA engineers?

Testing Roles in Test-Driven Development: Software Engineer in Test

- Software Engineer in Test (SET)
  - Developer with primary focus on testability and general test infrastructure
  - Responsible for design review from quality perspective, code quality and risk analysis.
  - Responsible for code refactoring to increase testability
  - Write unit testing frameworks and automation
  - Focus on increasing quality and test coverage
- SWE and SET are partners in the codebase
- How are these responsibilities accounted for in a traditional QA organization?

Testing Roles in Test-Driven Development: Test Engineer

- Test Engineer (TE)
  - Focus on testing code from user perspective, including use cases and usage scenarios
  - Develop automation scripts and code that drives usage scenarios and mimics users
  - Lead testing work of SWEs and SETs
  - Organize quality practices
  - Responsible for test result interpretation
  - Drive test execution
  - Build end-to-end test automation
  - Also responsible for communication of risks in design and feature complexity
- What are the similarities and differences between this role and QA engineering roles in a traditional QA organization?

Release Approach

- In addition to test-driven development, Google also adopted iterative development release lifecycle
- Products are released with a minimum core set of features
- Quick subsequent iterations allow to get feedback from users and incorporate the feedback into each successive release
- Quality is a key consideration for each release
- Releases are frequent and add a small number of features in each
- Products proceed through a number of “channels” prior to being released to users

Product Code Channels

- Google uses a concept of channels to move code through various stages of build/test/fix/release process
- Canary Channel
  - Daily builds – used by engineers (dev and test) and managers working on the product
- Dev Channel
  - Weekly builds – sustained some usage and testing
  - All engineers are required to pick up Dev channel builds and use them for work and testing
- Test Channel
  - Best build of the month – passed most sustained testing and the one engineers trust most for work
  - Can be picked up by internal users and a candidate for Beta Channel
- Beta or Release Channel
  - Stable test channel builds that passed internal usage and quality test standards; available for external use

Small/Medium/Large Tests

- Google classifies tests as Small, Medium, Large, based on the scope included in the test
- Tests can be automated or manual, with the emphasis on making most tests automated
- Small Tests
  - Focus on one function or module
  - Test functionality, data corruption, error conditions
  - Usually automated
  - Written by SWEs, sometimes SET
- Medium Tests
  - Focus on interaction of two or more features
  - Usually automated
  - SETs drive development of medium tests, SWEs responsible for test code and maintenance
  - TEs can execute medium tests manually or with automation
- Large Tests
  - Cover 3 or more interacting features, represent real usage scenarios and real data sources
  - End-to-end scenarios are large tests
  - All testing roles involved in creation and execution of large tests

Class Checkpoint: Test-Driven Development (at Google)

- How is Google testing approach different from the traditional testing approach we see at other companies?
- What are some of the benefits and drawbacks of Google's approach to testing?
- Can this testing approach be reproduced at another organization, or is it unique to Google's software culture?

Resources

- John Watkins, Simon Mills, Testing IT: An Off-the-Shelf Software Testing Process, 2nd edition, 2011, Cambridge University Press, Chapter 2
- James Whittaker, Jason Arbon, Jeff Carollo, How Google Tests Software, 2012, Addison-Wesley, Chapter 1