Testing

Elena Punskaya, op205@cam.ac.uk

It is All About Trust

Defects destroy the trust required for effective software development. The customers need to be able to trust the software. The managers need to be able to trust reports of progress. The programmers need to be able to trust each other. Defects destroy this trust.

Kent Beck and Cynthia Andres, Extreme Programming Explained: Embrace Change

Main Questions:

- **What is Testing?**
- **What to Test?**
- **How to Test?**
- **When to Test?**
- **Who should Test?**
- **What are the Tools for the Job?**

What is Testing?

- **Testing is double checking**
  - If you add a column of numbers one way, there are many errors that could cause your sum to be wrong. Add the numbers two different ways, say from top and then from the bottom and the same answer is likely to be the right answer.

> **Finding bugs is somewhat like fishing with a net. We use fine, small nets (unit tests) to catch the minnows, and big, coarse nets (integration tests) to catch the killer sharks.**
>
> *Andrew Hunt and David Thomas, The Pragmatic Programmer*

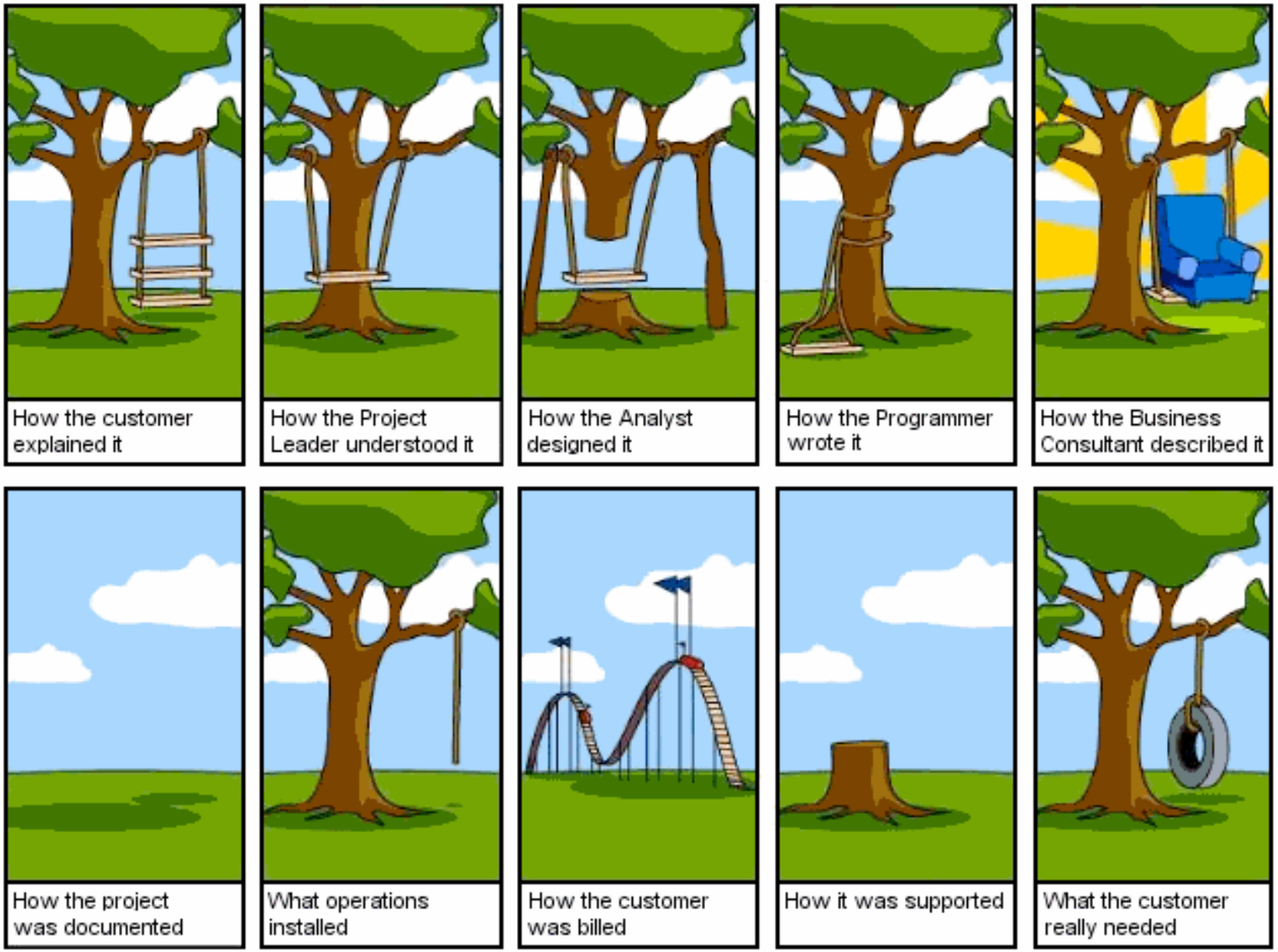
- **Many types of tests exist, what follows is just a starting point!**
- **Many ways to categorise the tests, depending on one's perspective**

What to Test?

Let us follow the process:

- **Unit test**
  - is code that exercises a small unit (module) of software separate from other units of the application
  - is the foundation of all other types of testing (if parts don't work by themselves they won't work together)
- **Integration (component) testing**
  - ok, all parts seem to be working but how do they interact with each other, does the entire subsystem work?
  - with good OO design in place should be straightforward to detect the issues
  - potentially can be the largest source of bugs in a system
- **Validation and verification**
  - ok, I put something together, this seems to be working as my users wanted but is this what they needed? are functional requirements met?

Validation and Verification?

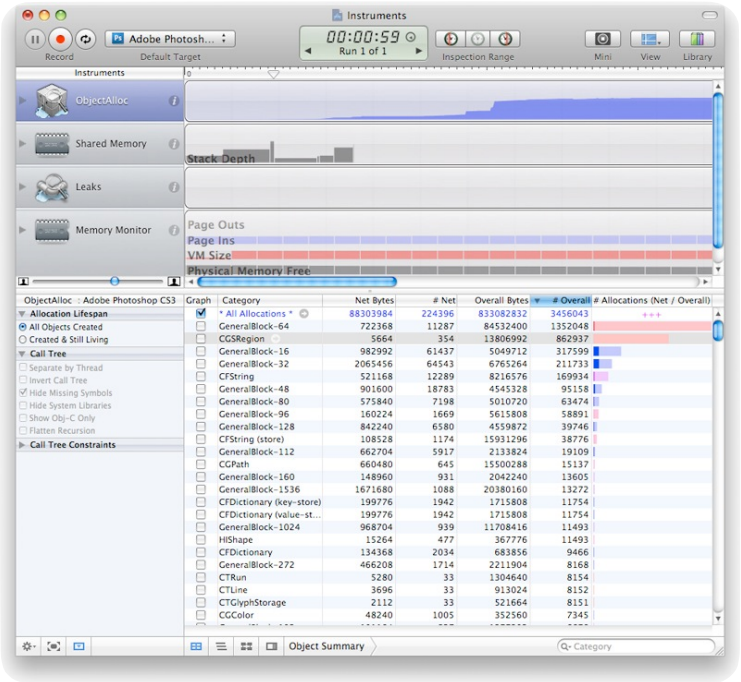


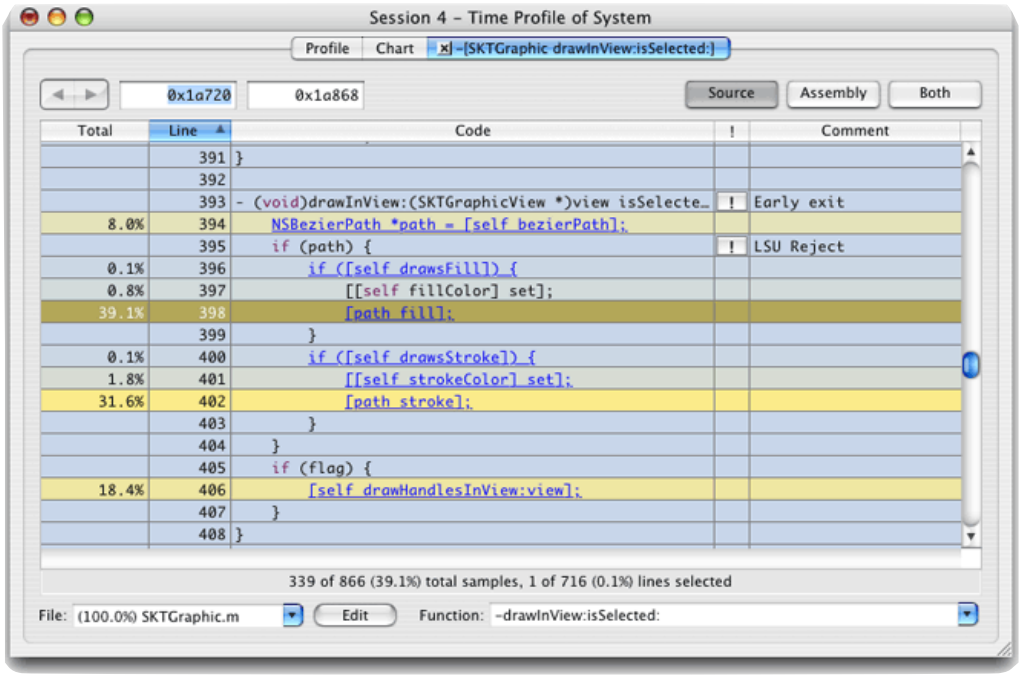
What to Test?

- **Resource exhaustion, errors, recovery**
  - ok, now the system behaves correctly under ideal condition, what about real-world?
  - could it run out of
    - memory?
    - disk space?
    - CPU bandwidth?
    - video resolution?
    - network bandwidth?
    - etc.
  - if it has to fail, will it fail gracefully?
- **Performance testing, stress testing or testing under load**
  - ok, it can work under real world conditions but what if the number of users/transactions/ connections increases? is it scalable?
  - you might need to simulate the load realistically
- **Usability testing**
  - ok, seems to work in real world but would it with real users under real conditions? this new tool “fits our hands” but does it “fit other users’ hands”
  - human factor plays important role, is it intuitively clear how to use?
  - failure to meet usability requirement is the same as dividing by zero
- **Test are also software - don’t forget to test them! :-)**

Profiling Tools

- **Profiling tools allow to analyse software performance**
- **In particular, tracking Memory Allocation throughout runtime allows to spot memory leaks and inefficient memory use (requiring too much memory space without a real need)**
- **Tracking time required execution of program methods helps to identify "bottlenecks" of performance that could be caused either by ineffective programming techniques or could benefit from optimisation**





XCode profiling tools, https://developer.apple.com/library/ios/#documentation/Performance/Conceptual/PerformanceOverview/InitialEvaluation/InitialEvaluation.html#//apple_ref/doc/uid/TP40001410-CH206-SW8

Usability Testing

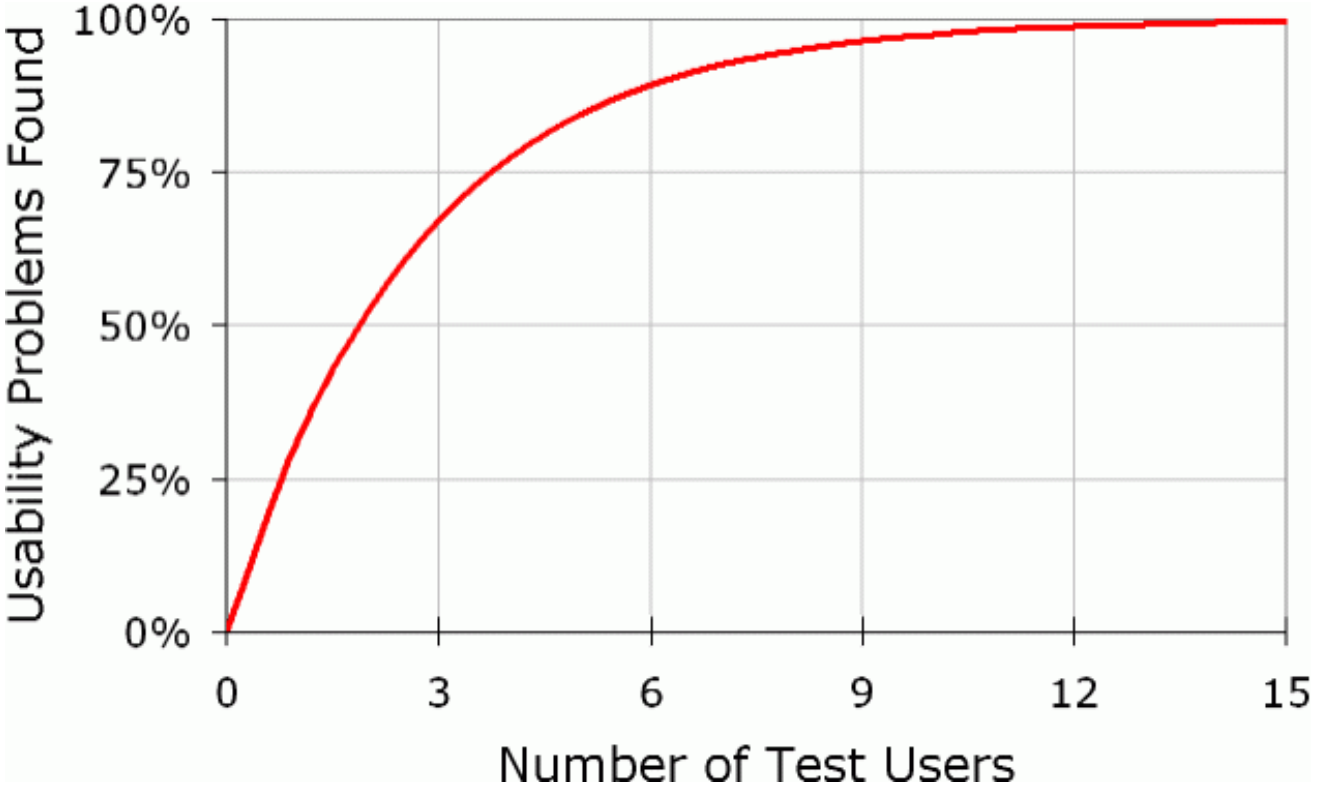
- Usability Testing ~~is~~ should be one of the most critical parts of software development: designing experiences!
- Consider web search – a user wants most relevant results in the fastest way, relevance could be measured by analysing user clicks on the result link, but what about speed? Is 1s fast or 0.3s or 0.03s? Is it about the actual speed or about speed's perception by the user?
- Usability studies help to answer those questions by using two types of user research: quantitative (data points) and qualitative (perception)
- Quantitative results allow to draw conclusions approximating them on to the whole target user set, i.e. "launching a new banking website will cause no more than 0.1% of online banking users to call the support line"
- Qualitative results allow to capture how users feel about their experience with the product/ service
- Good usability studies will use both approaches

Usability Study Design

- **Method**
  - define user interactions that are going to be tested, how the user is instructed and how the system is implemented (real product/ prototype / simulator)
  - define how the user feedback is captured: interviews, scoring cards
  - define how user interactions are observed and analysed: video and screen capture, eye tracking
- **User group (Know Your Customer!)**
  - choose the user group to provide a representative set of product/ service users
    - an app for remote access to computers (e.g. SSH client) is likely to be used by **experienced** computer users
    - a web conferencing app (e.g.WebEx) need to be easier to setup by **business** (non-technical) users
    - a call/chat client (e.g. Skype) should be easy to use by **any** computer user
  - segment target users by their characteristics: demographics: age, gender; level of experience...
- **Results presentation – a report should include**
  - conclusions based on observations of users by the specialists to identifying e.g. a common source of confusion
  - qualitative user feedback
  - quantitative results: rankings/scoring by user groups (charts/tables)

Number of Test Users

- **Jacob Nielsen advocated that most effective approach is multiple usability studies with 5 users at a time**



http://www.useit.com/alertbox/20000319.html

- **The chart above applies to the number of users of the same kind, in reality, if the target user set is diverse, so it'd be better to have 5 users for each user segment**

Number of Test Users

- **Laura Faulkner in "Beyond the five-user assumption: Benefits of increased sample sizes in usability testing" showed that 5 users do not necessarily deliver 85% problem discovery but a set of 20 was certainly very reliable**
- **In practice, the optimum number can depend on the diversity of target users and complexity of the product**

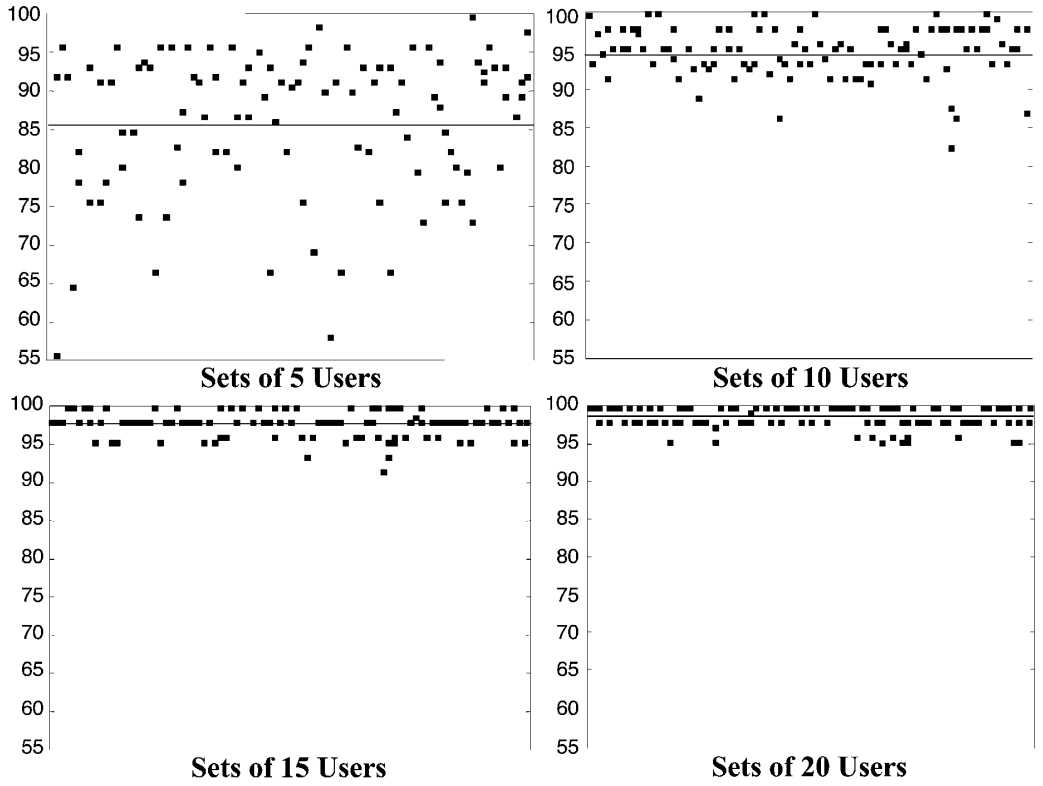
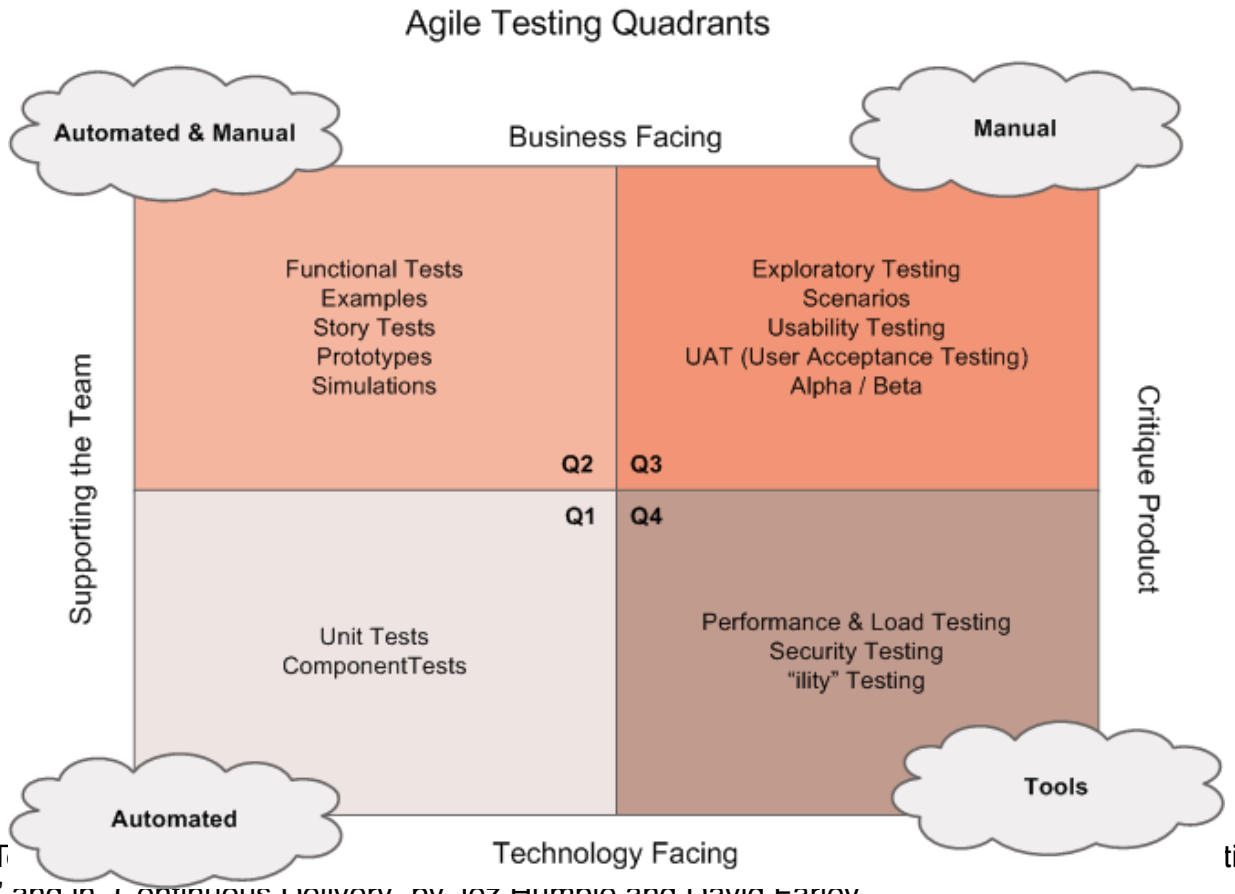


Figure 1. The effect of adding users on reducing variance in the percentage of known usability problems found. Each point represents a single set of randomly sampled users. The horizontal lines show the mean for each group of 100.

Tests from Different Perspective

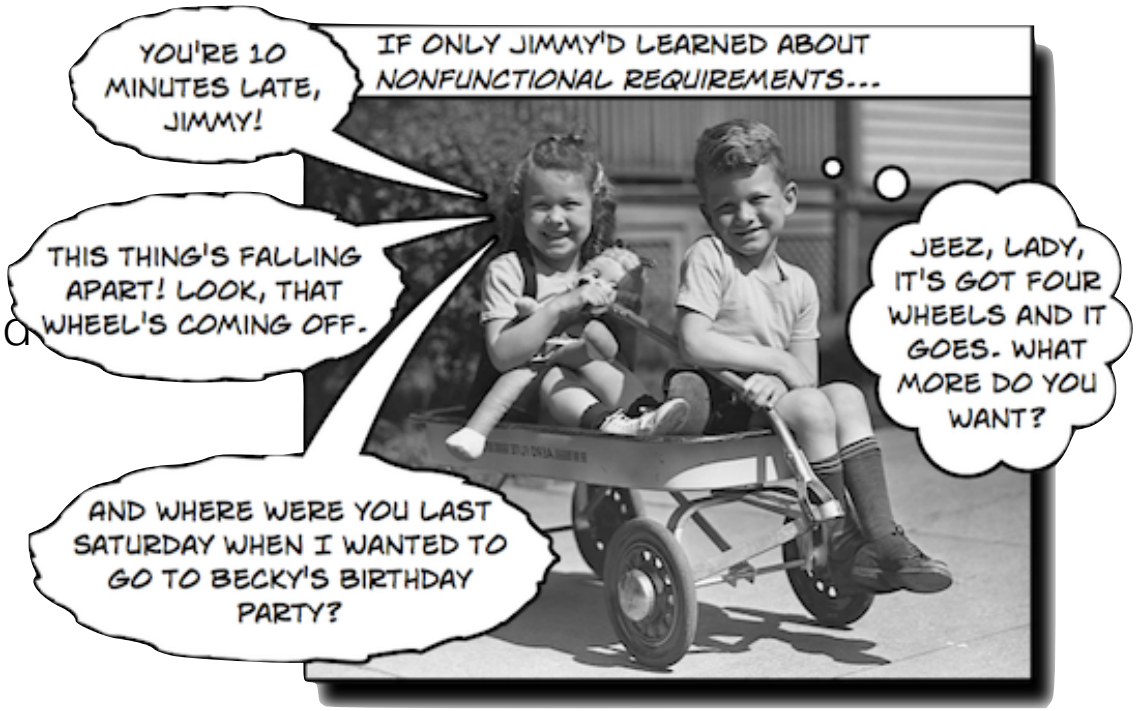
- **"Testing Quadrant" categorises the tests according to whether they are Business-Facing or Technology-Facing and whether they support development process or used to critique (review and analyse the project)**



- Brian Marick introduced T ting: A Practical Guide for Testers and Agile Teams" and in "Continuous Delivery" by Jez Humble and David Farley

Business-Facing tests Supporting Development

- **Acceptance testing**
  - testing conducted by a customer to verify that the system meets the acceptance criteria of the requested application
  - ideally should be written and automated before the development starts
  - ideally should be written by the customers or users
  - for developers they answer the question: How do I know when I am done?
  - for users they answer a question: Did I get what I wanted?
  - typically should run when the system is in a production-like mode



http://www.d80.co.uk/post/2011/03/08/Non-Functional-Requirements-the-forgotten-overlooked-and-underestimated.aspx

Acceptance tests that concern the functionality of the system are known as **Functional** acceptance tests – the distinction between **Functional** and **Non-Functional** is a bit blurry and often misunderstood

Technology-facing tests Supporting Development

- **Technology-facing tests are written and maintained exclusively by developers**
- **Usually these are unit tests, component tests (integration tests) and deployment tests**
- **Deployment tests are performed whenever one deploys the application and check that it is installed correctly, configured correctly, able to contact any services required and is responding**

Business-Facing tests that Critique the Project

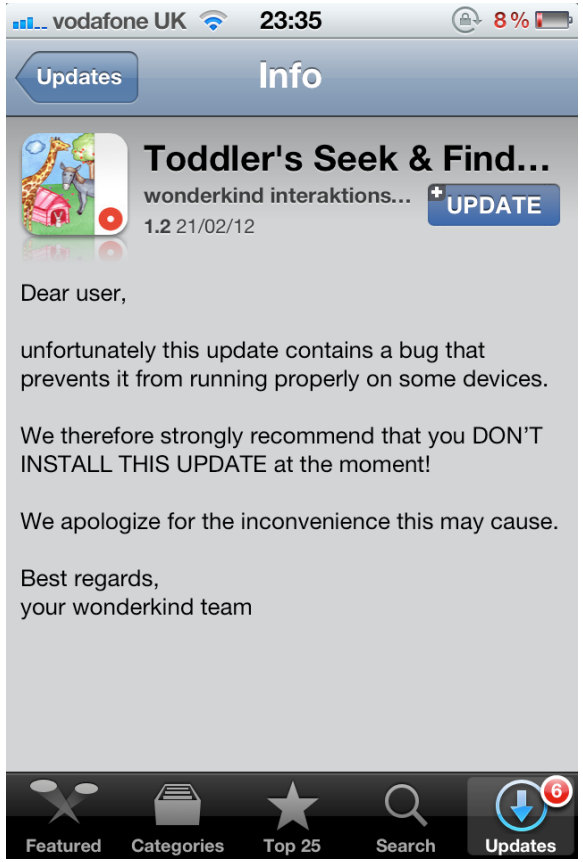
- The tests are not just about verifying that the application meets the specification but also about checking that the specification is correct
- Applications are never specified perfectly in advance so there is always room for improvement, users try things they are not supposed to try and break them, complain about usability of most commonly performed tasks, identify new features
- Showcases are particularly important - show new functionality to the customers and users as soon as possible to catch any misunderstandings early (can be a blessing and a curse - you might have a lot of suggestions!)
- Exploratory testing - “the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests” [James Bach]
- Beta testing - give your application to real user, release new features to selected groups without them even noticing

Technology-facing tests that Critique the Project

- **Acceptance tests may test functionality (functional acceptance tested) but may also test other qualities of the system such as capacity, usability, security, modifiability, availability, etc.**
- **All these are qualities other than functionality go under umbrella of nonfunctional tests although the distinction is very blurry, however, what is important is to bother testing them!**
- **It is also not necessarily fair to say that these tests are not business facing – often they are!**

How to Test?

- **Regression Tests**
  - compares the output of the previous test with the previous or known values
  - make sure that bugs fixed today don't break something else - no unpleasant surprises
  - can run regression tests to make sure the components function correctly, entire subsystem functions, performance, etc.
- **Test Data**
  - real-world - "typical data" collected; "typical" might be a surprise and may reveal misunderstandings in requirements - watch out!
  - synthetic - artificially generated data (not enough real-data, need certain statistical properties, need to stress the boundary condition)
  - real-world and synthetic - expose different types of bugs
- **GUI testing**
  - need special tools to exercise, can't automate everything



Why teams Choose to Live with Defects?

- **Is it possible to create BugFree software?** ***Yes in theory but how one would prove this?***
- **How much does it cost to create BugFree software?** ***Are costs astronomical?***

- **Dilemmas:**

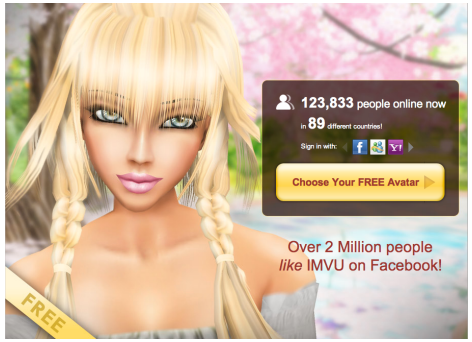
  - bugs are expensive (direct costs of fixing and indirect costs of damaged reputation and relationship, lost time) and eliminating bugs is expensive => economically viable option depends on the acceptable defects level in any particular area

  - there will always be bugs! (Unknown unknows, unexpected circumstances, new situation)

Horses for Courses

- **Not all SOFTWARE is created equal, choose an engineering approach accordingly**

Our tests suite takes nine minutes to run (distributed across 30-40 machines). Our code pushes take another six minutes. Since these two steps are pipelined that means at peak ***we're pushing a new revision of the code to the website every nine minutes****. That's 6 deploys an hour. Even at that pace we're often batching multiple commits into a single test/push cycle. On average* ***we deploy new code fifty times a day****.* [1]



This software never crashes. *It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats : the last three versions of the program – each 420,000 lines long-had just one error each.* ***The last 11 versions of this software had a total of 17 errors.*** [2]

When to Test?

- **The sooner one finds the defect the cheaper it is to fix it (catch it the minute it was created and it costs nothing)**
- **Late testing is expensive and leaves many defects**

We want to start testing as soon as we have code. Those tiny minnows have a nasty habit of becoming giant, man-eating sharks pretty fast, and catching a shark is quite a bit harder.

Andrew Hunt and David Thomas, The Pragmatic Programmer

Kent Beck and Cynthia Andres, Extreme Programming Explained: Embrace Change

- **Frequent testing reduces costs and defects (fix sooner and cheaply)**

Kent Beck and Cynthia Andres, Extreme Programming Explained: Embrace Change

Who Should Test?

- **Frequent testing (difficult to have a dedicated tester) might mean programmers themselves write the tests, i.e. the same people who make mistakes write the tests - they need another prospective! Buddy programmer? Think like a users?**
- **Consider two prospectives: programmers and users, involve the users at the very early stages**
- **Most developers hate testing - they tend to be very gentle (subconsciously) with their creations and tend to avoid the weak spots intuitively.**
- **Who likes testing???**
  - testing could be very tedious – given a 100 pages long test script most humans turn on the "zombie mode" ;)
- **Solution? Automate whenever possible (so all the tedious bits are done by the machines)!**
  - But someone still needs to define all test cases and implement them in the automated testing framework
  - Also, it is difficult to substitute manual testing for capturing unexpected user behaviour, the best testers have a six sense for breaking things :)

Automated Testing

- **The goal is to reach "coverage" – cover the most practically possible number of test cases**
- **It would typically include**
- **All unit tests – developers implement and commit those to the source code repository during the development process**
- **Regression tests – given the set of know inputs, does the system still produce the same set of outputs?**
- **UI testing – cover various screen flows, difficult to automate but increasingly more and more coverage is possible**
- **Probably enough for a "standard" app, but what about large distributed services in the "Cloud"? – Test Automation Cloud**
  - Salesforce.com: average 600 change lists per day
  - "Test automation cloud provides accurate, complete and fast feedbacks to every change to the system, on a per change list basis"
  - Hardware: 2000 Linux Virtual Machines and 1000 Selenium Virtual Machines to run all required tests

Tools for the Job

- **The test tools are growing and maturing rapidly, many contributed to Open Source by big commercial vendors and many are created and evangelised by startups, some as examples are:**
- **xUnit – a unit testing framework, ported to many languages, including JUnit (Java) and NUnit (C#)**
- **KIF – "Keep it Functional" – a new iOS UI testing framework, allows to specify user interactions for test scenarios and check app's resulting behaviour (which screen appeared etc.)**
  - http://corner.squareup.com/2011/07/ios-integration-testing.html
- **Selenium – "Automates Browsers" – allows to simulate interactions and check results with websites**
  - http://seleniumhq.org/
- **Robotium – "It's like Selenium, but for Android"**
  - http://code.google.com/p/robotium/
- **Static code analysers – checking for code inconsistency/inefficiency, security etc., e.g. FxCop (for .NET)**