Software Testing for HPC

Nick Johnson, EPCC

Reusing this material

Motivation

- How do you know your program works?
  - Do you get the right result?
  - Do all the pieces work?
- How can you demonstrate it is correct to others?
  - How do you know someone else’s code is correct?
- Just write some code and see if it compiles?
  - We can do better
- Test our code
  - Makes our lives easier
  - Saves us time
  - Improves the quality of our software
  - Lets us know when we have finished

What is software?

- Starts with a customer with a problem
  - This generates requirements
- Then you produce the solution
  - Design
  - Code
  - Installation
  - Documentation
  - …
- Software is everything you deliver
  - All of it is testable

What is testing?

- A procedure or quantifiable way to check the correctness and other metrics of a piece of software.

- Testing is a process to verify
  - Software does what you expect
  - How well it does it

- Applied throughout the development of a piece of software

Benefits of Testing

- Testing improves
  - Quality of software
    - Reduce the number of errors
    - Find those errors quicker
  - Confidence
    - In the software itself
    - To make changes
  - Design
    - Makes you think about the interfaces and purpose of different parts of the software

What software testing is NOT

- NOT just a bug hunt
  - We rushed the design and development, so we're testing to find all the bad bugs before someone else does!
- NOT something we do at the end of a project
  - The earlier you find a bug, the easier it is to fix
  - Test early, and test often!
- NOT expensive
  - Do it continuously, from the start
  - Bugs you find at the design stage are cheaper to fix than once you start coding
  - Find errors when the problem area is fresh in your mind

archer

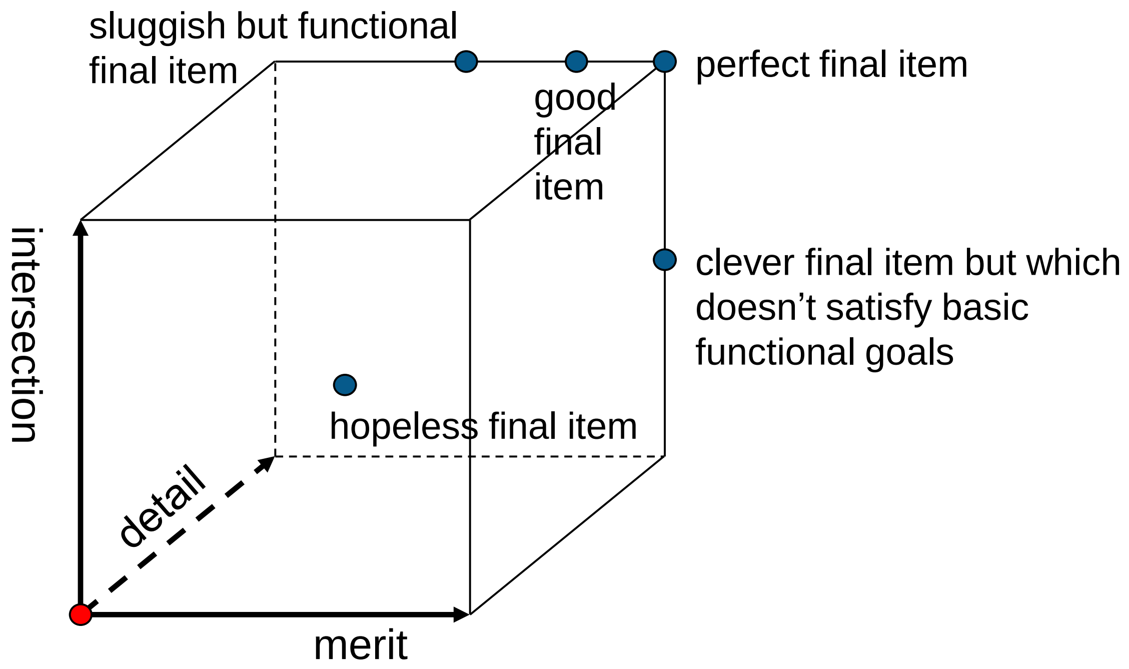
epcc

THE UNIVERSITY OF EDINBURGH

Quantify

- Ultimately, testing is needed to quantify whether your software satisfies the “Big Three” design goals…

- Detail
  - Completeness
- Intersection
  - Correct behaviour
- Merit
  - Performance


sluggish but functional final item
perfect final item
good final item
clever final item but which doesn’t satisfy basic functional goals
hopeless final item
intersection
detail
merit

Quality

- Testing helps ensures that we have a quality product
  - A quality product is one that meets the end users expectations
- What is bad software?
  - Any product that does not live up to its expectations
- Testing is an important part of any quality assurance process

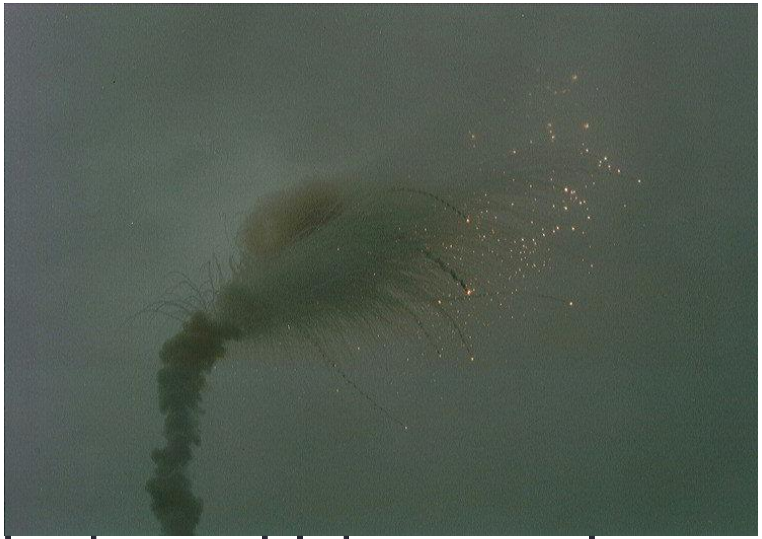
Bad software 1

- Patriot Missile System
  - Highly publicised

- Highly ineffective
  - time used to calculate target velocity
  - system converted time from internal clock to an integer
  - cumulative error in conversion
  - periodic re-boot to minimise error

- No-one ever thought to test this!

Bad Software 2

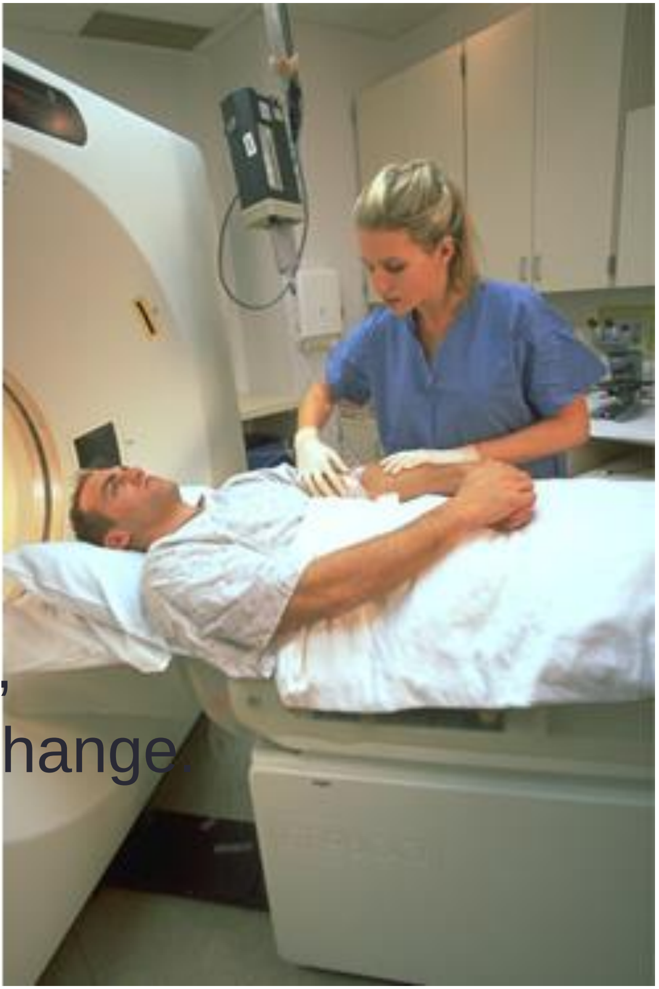


- Ariane 5 Explosion
  - Code from Ariane 4 re-used
- Faster engines in Ariane 5 triggered a bug which caused buffer overflows
- Oops!!
  - No comprehensive testing of old code in the new platform
- Result – A very big bang

Bad Software 3

- Therac-25: Medical Linac
- Two modes: “Electron” and “X-ray”
- Defect in control sequence
  - user entered “X” by mistake
  - quickly corrected sequence, entering “E”
  - ran sequence
  - original sequence ran, not corrected
- Because user corrected error quickly,
- the system did not update the change.

- Several deaths occurred.



All Defects Have a Cost

- The cost of smaller defects can still add up

- How much time does it take to find and fix a bug?

Time to test and find a bug	1/2 hour
5 people reading bug report	1/4 hr/person
2 people reproducing bug	1/4 hr/person
1 person to fix	1 hour
Testing fix	1/4 hour
Commit/review changes	1/4 hour

  - 2 1/2 hours maybe?
  - What happens if you find 10 bugs per day?

- When caught by customers, easily add on an extra hour for measurable costs, plus harm to reputation.

- The earlier in the development process you find a defect, the less expensive it is to fix

Bad Software

- Poor Quality Software is
  - Hard to maintain
  - Hard to change
  - Embarrassing
  - Gives results that are less than rigorous
  - Costly
  - At worst, fatal.

Summary

- Software testing includes the processes that:
  - Find out how our software behaves
  - Give us confidence that it does what we designed it to do
  - Establish the quality of the product
  - Tells us when to stop!
    - In Scientific work, we should know the expected outcome a-priori and how much tolerance from this value we will accept.

- Testing allows us to reduce the number of costly defects in our software

- Testing should be done from the beginning

Up next…

- Introduction to types of software testing and when to do them
    - Will look at how Unit Testing fit into the rest of the software testing hierarchy
- How to manage your tests
    - Project organisation
    - Managing test runs
    - Management of defects and bug fixes

Introduction to Types of Testing

- Testing should occur throughout the software development process
- Tests can be applied to
  - Individual components
  - Groups of components
  - The entire system
- Lets look at types of tests applied to production quality software

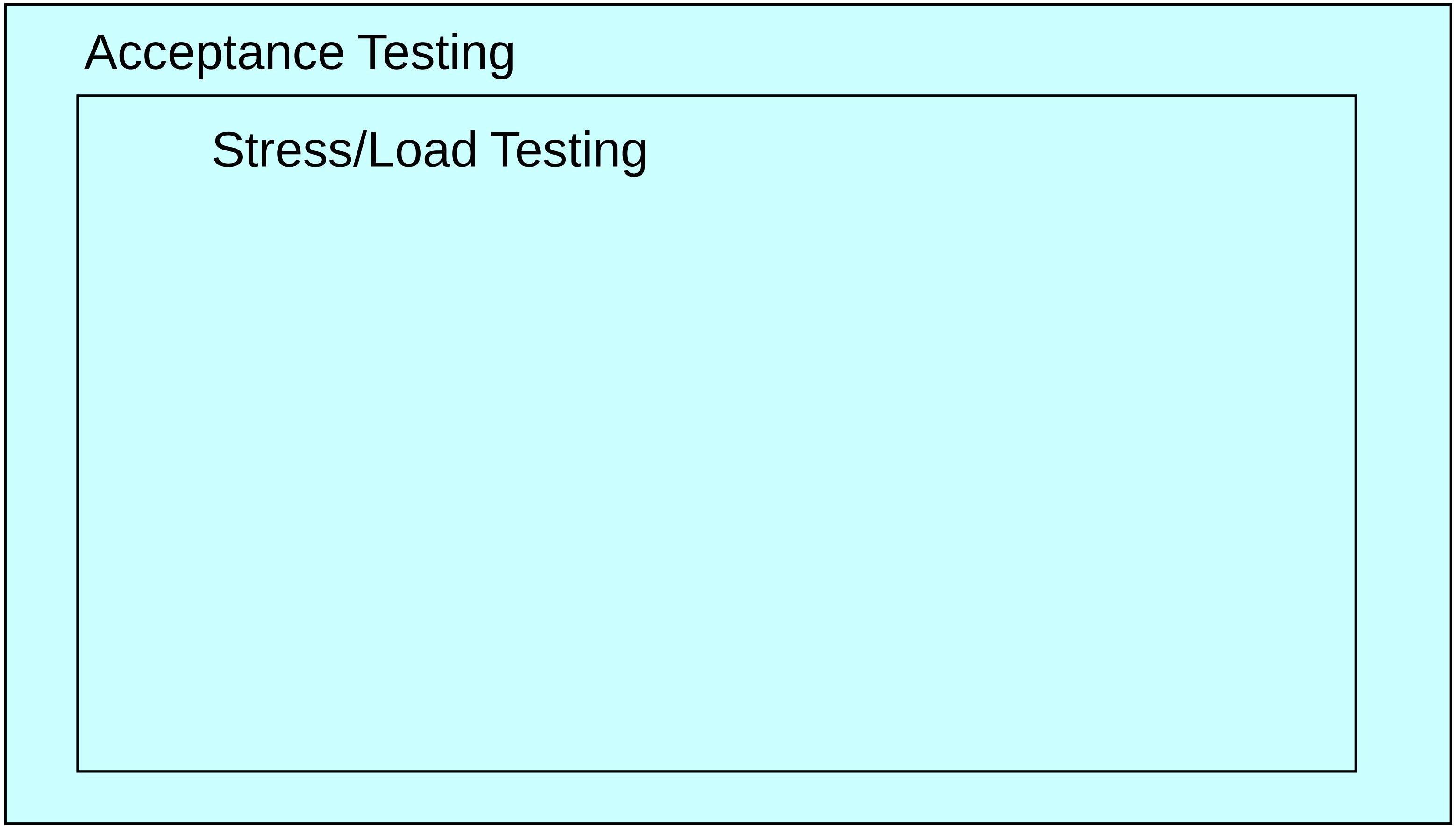
Acceptance Testing

Acceptance Testing

Acceptance Testing

- Tests that the software does what the user wants
- They should come from the requirements
- They are MUST HAVES
- They are the first part of the software you should design
- I.e., does result X fall within N% (or N absolute) of the expected value from book work etc.

Stress/Load Testing



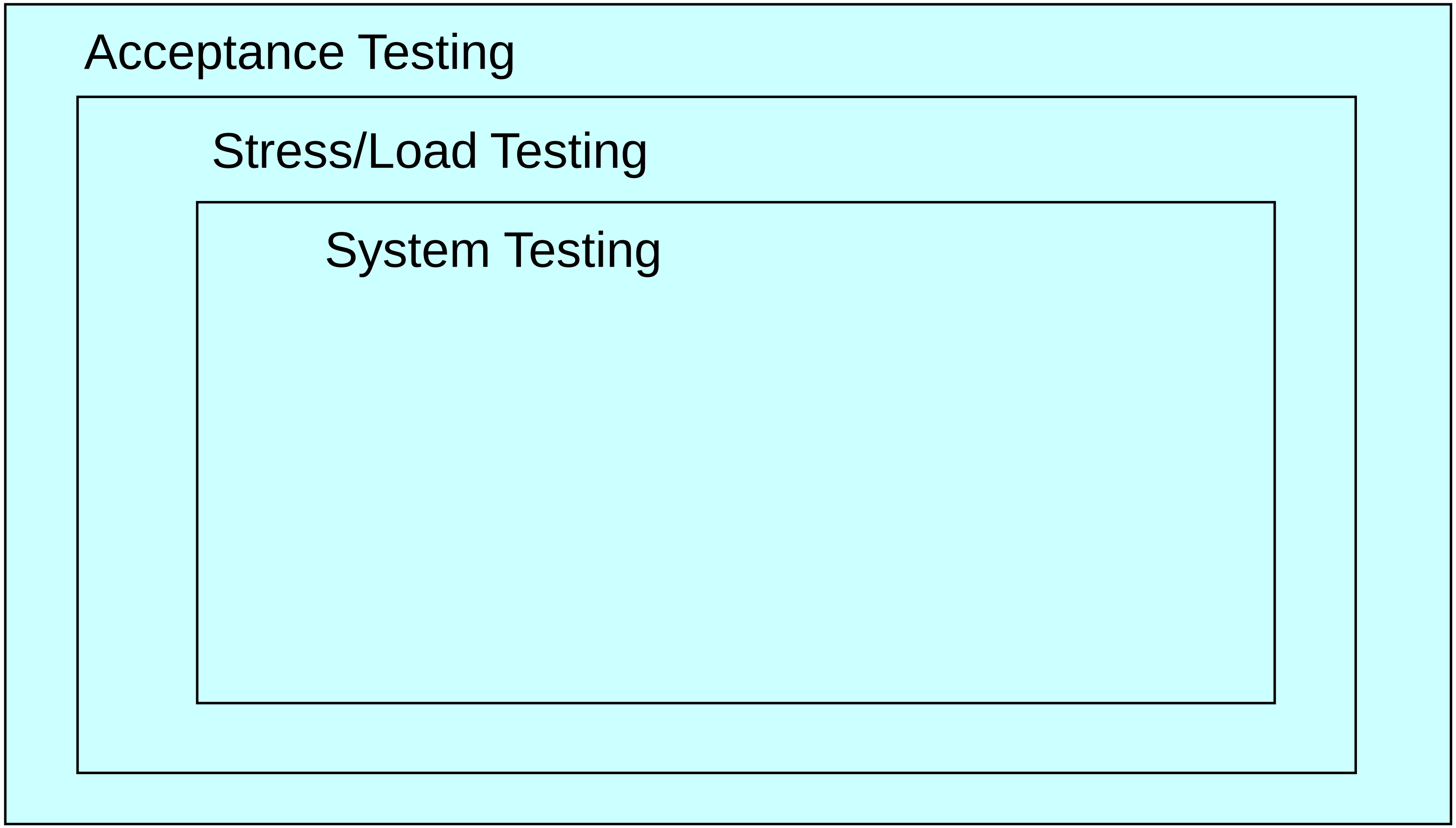
Acceptance Testing

Stress/Load Testing

Stress/Load Testing

- Tests how robust the system is
- How does the system cope under heavy loads?
  - Or with large data sizes
- Sometimes to the point of failure
  - To test error handling
- Tests realistic input in realistic conditions
- Often code is developed using smaller computing resources

System Testing



Acceptance Testing

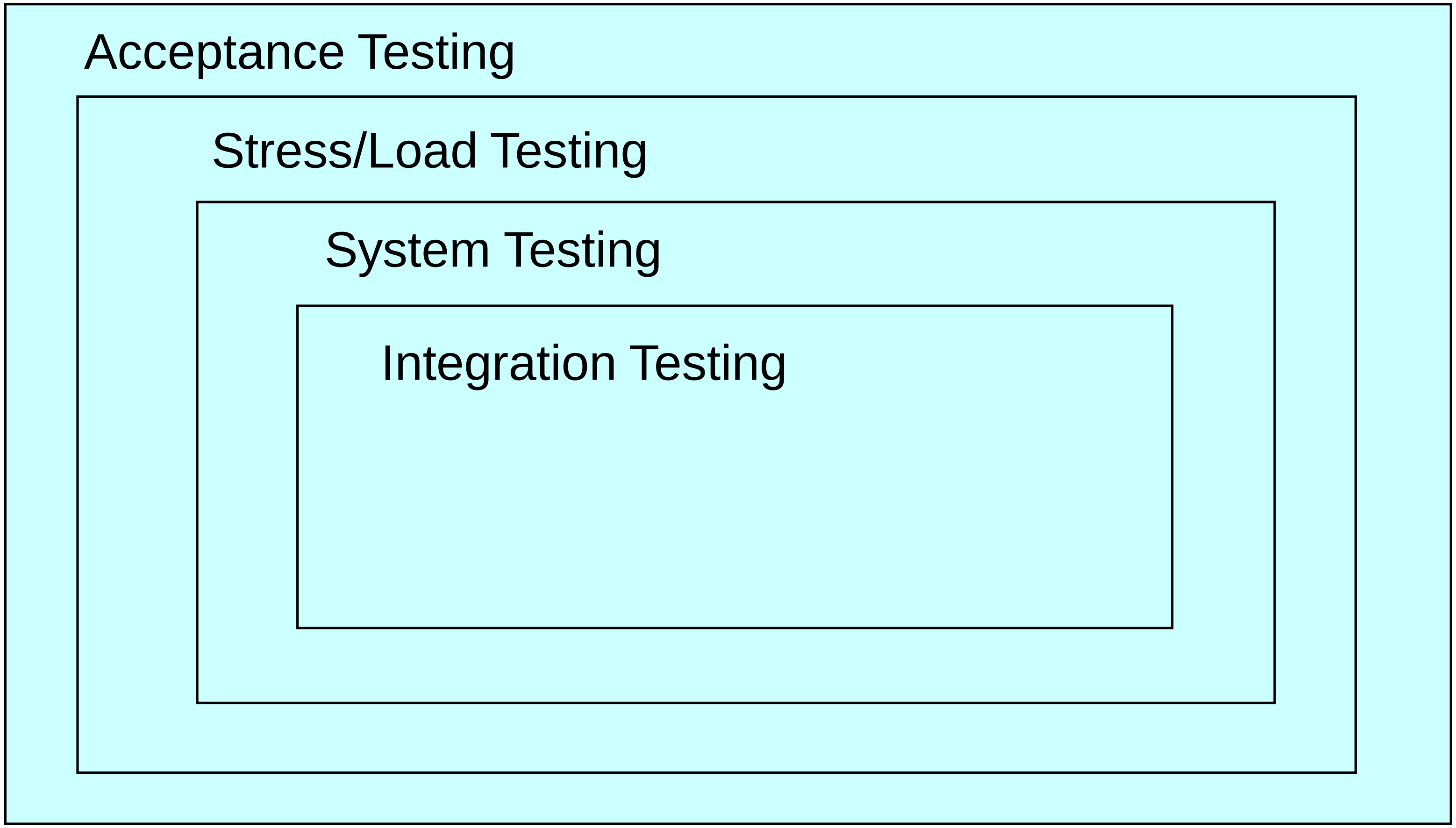
Stress/Load Testing

System Testing

System Testing

- Tests the entire system
  - The system being everything you’ve done so far
  - Used as a milestone check
  - Gives you confidence you’re on target
- Systems tests should be designed at the same time as the code is designed
  - Think of them as mini-acceptance tests
- Systems tests should ideally be done by a third-party (and involve an end-user)
  - Involves destructive testing (looking for bugs)
  - Constructive testing (suggesting improvements)
- The tester and the developer have to work together to get the most from system testing
  - It’s easy for both sides to blame the other for problems

Integration Testing



Acceptance Testing

Stress/Load Testing

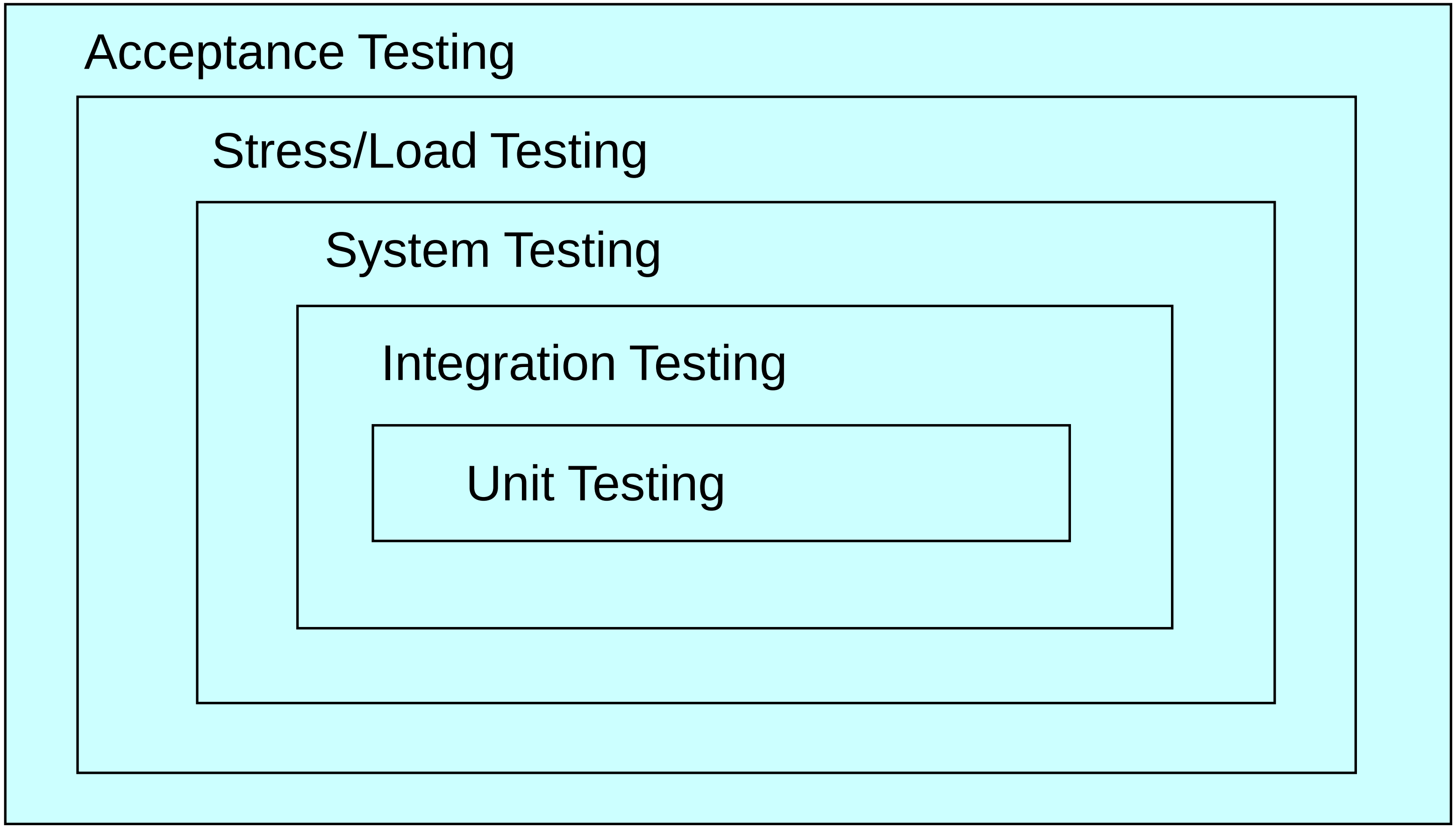
System Testing

Integration Testing

Integration Testing

- Multiple software modules are tested together to check that they integrate properly
- Designed to test the "glue"
  - Checks that interfaces are being used correctly
  - Tests assumptions made by developers of different modules
- Communication throughout the development process reduces the pain when you come to do integration testing
- Very important phase of testing
  - It is easy for things to fall between the gaps in modules

Unit Testing

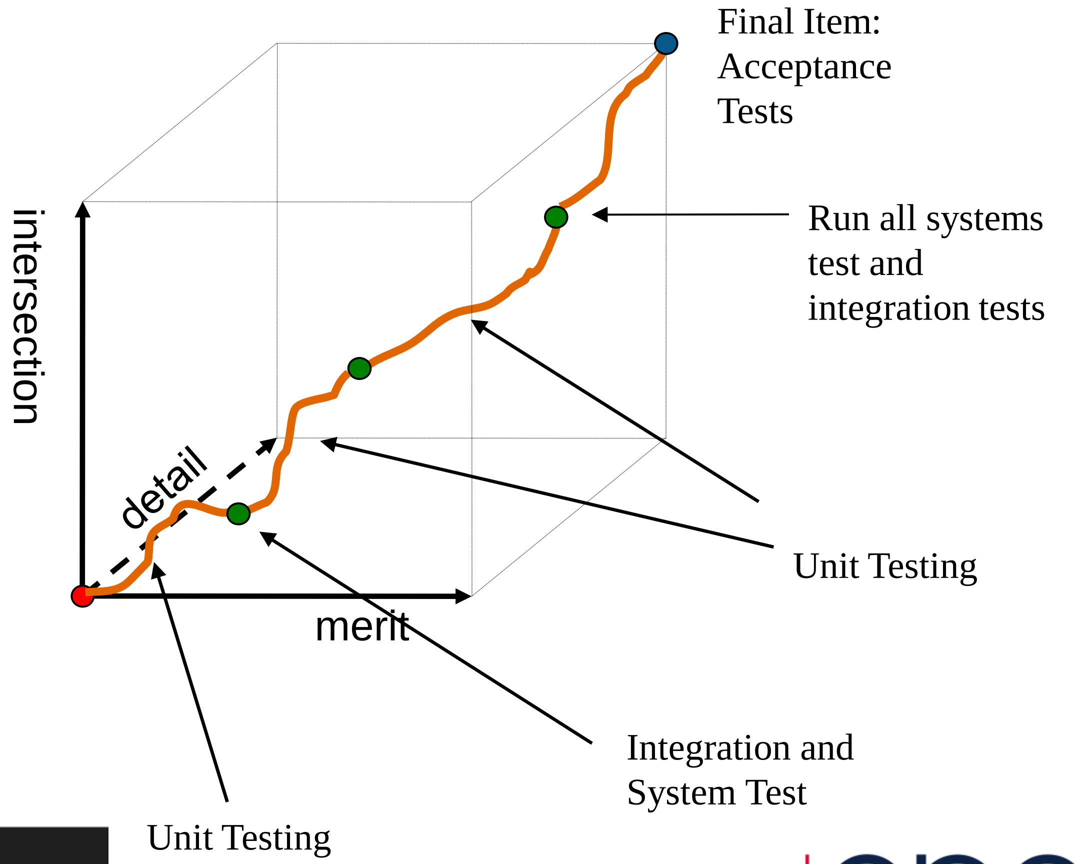


Unit Testing

- Tests individual modules and small units of code
- Verifies the low level behaviour of the software
- More on this later…

Testing and the design cube

- What testing do we use and where?



Final Item:
Acceptance
Tests

Run all systems
test and
integration tests

Unit Testing

Integration and
System Test

Unit Testing

Summary of types of testing

- Different types of testing
  - Unit
  - Integration
  - System
  - Stress/Load
  - Acceptance
- Applied throughout the development process
  - Continuous unit testing
  - Iterations of integration and system testing
  - Final stress and acceptance tests before delivering the final software

Re-motivation

- Why test?
  - Increases the quality of the software you produce
  - Saves you time
    - Find bugs soon after they are introduced
  - Gives you confidence in the code you develop
    - Easier to add code and re-factor
  - Encourages good design
    - Makes you think more about the code as you write it

Tools & Techniques available

- Junit, CPPUnit, *Unit
- Test Driven Development is a proven technique now
- Your fellow participants
  - Test your designs by describing them to someone else
  - The act of explaining things to others makes you realise gaps in your own knowledge

Software Testing

- Should be carried out up-front and defined at design
  - Test early
  - NEVER at the project end only

- Used to keep us on track
  - Test often
  - Use tools to help automate the process
    - E.g., Ant, Make

Brief Summary

- Quality is key to all software components
  - Code
  - Documentation
  - Installation – often over looked
  - Functionality
  - Usability
- Testing is a valuable tool that can help increase the quality of your software
- Gives you confidence in your code
  - Very useful if you have to extend or change it

Comfort Break

- If needed…

Agenda

- Organization of tests
- Managing test runs
- Continued maintenance

Why bother?

- Larger projects require (some) management

More code or More people

Issues with testing

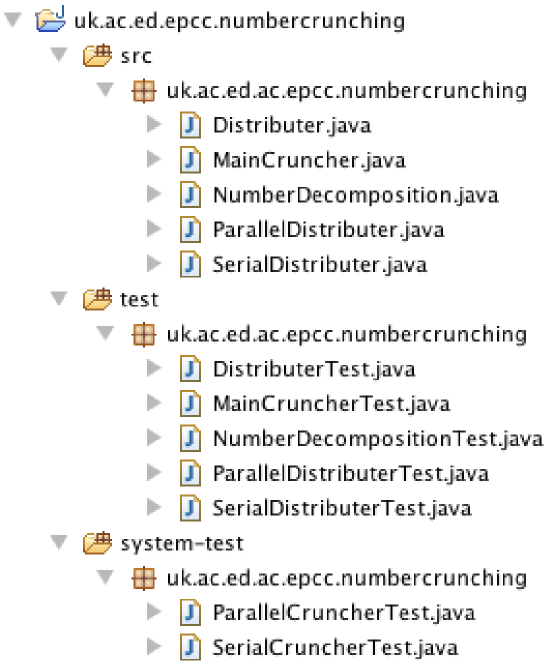
- Lots of tests
  - Developer-led testing
  - System test
  - Etc.
- How do you decide what to run and when?
- Lots of data
  - Each run will produce lots of results!
- Turn these into pass/fail statistics
  - How do you use these metrics?

What is Test Management

- Test management is about organizing your project
- Things to consider
  - Project structure
  - Test framework
  - Nightly builds
  - Code freeze
  - Code coverage tools
  - Test teams
  - Change management

Project structure

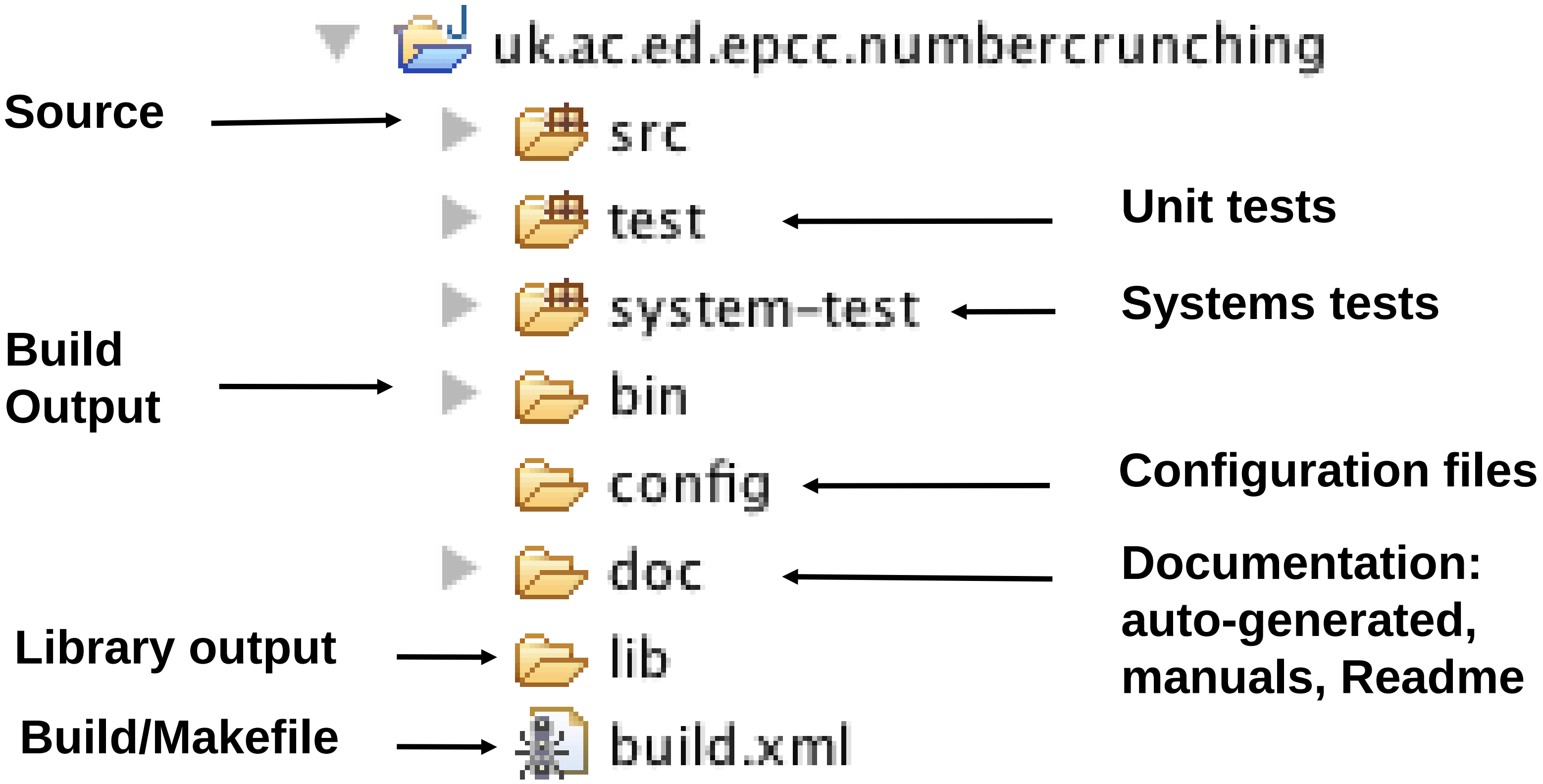
- First of all, you need to get organised
- Good project organisation
  - A must
  - Use make, ant, JUnit, CppUnit etc
  - IDEs good at this
- Make dependencies obvious
- Make it easy to file test results



```
uk.ac.ed.epcc.numbercrunching
  src
    uk.ac.ed.ac.epcc.numbercrunching
      Distributer.java
      MainCruncher.java
      NumberDecomposition.java
      ParallelDistributer.java
      SerialDistributer.java
  test
    uk.ac.ed.ac.epcc.numbercrunching
      DistributerTest.java
      MainCruncherTest.java
      NumberDecompositionTest.java
      ParallelDistributerTest.java
      SerialDistributerTest.java
  system-test
    uk.ac.ed.ac.epcc.numbercrunching
      ParallelCruncherTest.java
      SerialCruncherTest.java
```

Project structure

- For example:



```
uk.ac.ed.epcc.numbercrunching
    src
    test
    system-test
    bin
    config
    doc
    lib
    build.xml
```

Source

Unit tests

Systems tests

Build Output

Configuration files

Documentation: auto-generated, manuals, Readme

Library output

Build/Makefile

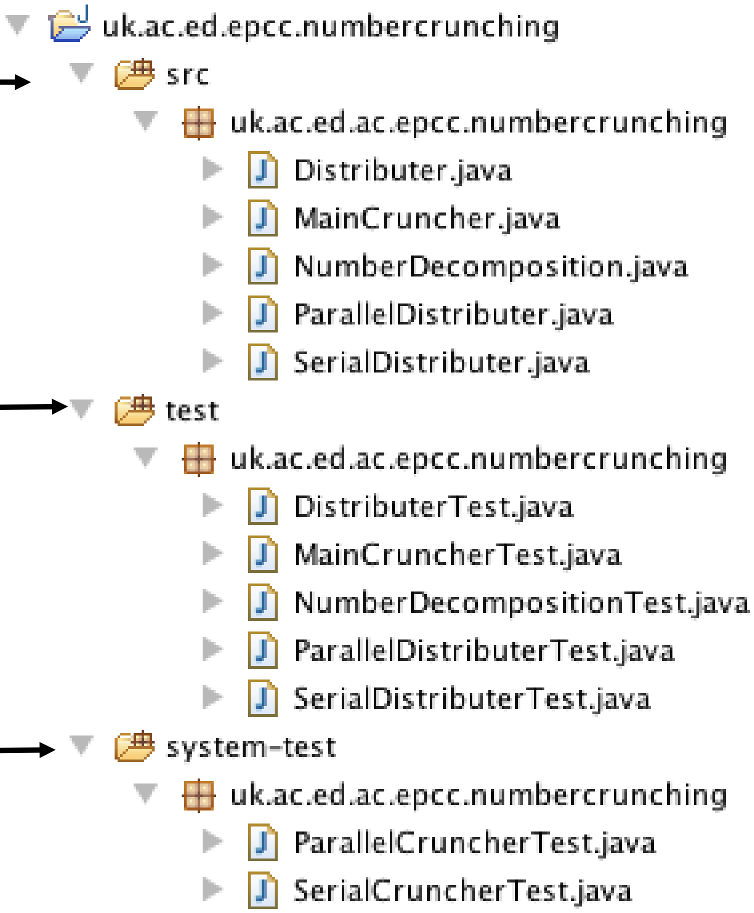
Project structure

- Expanding code folders

Source files

Unit test files
with same package/directory
structure

System test files



```
uk.ac.ed.epcc.numbercrunching
  src
    uk.ac.ed.ac.epcc.numbercrunching
      Distributer.java
      MainCruncher.java
      NumberDecomposition.java
      ParallelDistributer.java
      SerialDistributer.java
  test
    uk.ac.ed.ac.epcc.numbercrunching
      DistributerTest.java
      MainCruncherTest.java
      NumberDecompositionTest.java
      ParallelDistributerTest.java
      SerialDistributerTest.java
  system-test
    uk.ac.ed.ac.epcc.numbercrunching
      ParallelCruncherTest.java
      SerialCruncherTest.java
```

Test framework

- Organising structure and execution of the tests
- Test harness
  - Allows you to execute a predefined set of (unit) tests
  - Runs exes in bin/tests folder
  - E.g. “runAllUnitTests”
- Run test harness
  - As often as possible
  - Before you commit
  - Very important on large projects
- Automatic
  - Nightly, weekly, monthly to track regression
    - Depending on how long to run
    - Depending on frequency of commits

Nightly Builds

- Automatically building code and verifying correctness
- Starts and runs automatically at a predefined time
  - Usually at night
- Usual steps in a nightly build:
  - Checks out code from the repository
  - Builds the entire code
  - Runs all (unit) tests
  - Reports results (e.g., email, web page, etc.)

Benefits of Nightly Builds

- Gives confidence that yesterdays changes didn’t (or did) break the software
- Integrates and test components
  - Developed by different developers (important with bigger team)
  - In a “neutral” environment
  - Can be on multiple back-ends or environments
- Shows trackable progress

Test Suites

- Sub-sets of tests
  - Grouped together to test functionality
  - Testing specific aspects of the software
  - Shorter run-times
- For example: “Smoke Tests”
  - Designed to “smoke out” defects
  - Group tests that test most important functionality

Viewing test results

- Keep test results with each test
  - HTML format a good idea
- Keep a record of the last runs
  - Keep a set of metrics
- Have some sort of browsing tool
  - You can then dynamically view results
- All this is non-trivial
  - But well worth the effort for a large project
- Tools out there to help to generate results
  - JUnit, Ant, etc.

When to do test management?

- Always
- How much you do depends on
  - Number of developers
  - Duration of the project
  - Size of the code
- You will also have to balance the benefits with cost of setting up resources
- As the project grows add more aspects of test management

Pitfalls of test management

- Test tools are not a “silver bullet”
  - No substitute for thorough manual testing
  - No substitute for communication

- Be wary of bad tests
  - They will give a false sense of quality
  - Never be afraid to throw tests/data away
  - Make this part of your process

Pitfalls of test management

- Be wary of metrics
  - What does 100% pass rate mean if I’m only testing 10% of the software?

- It’s as important to know which tests fails, as much as how many
  - You can then spot weak areas
  - You can then prioritize fixes

Test Roles

- Who is responsible for what?
- Large projects will have a test team
  - Systems testing
  - Test infrastructure
  - Version management
  - Any third-party testing
  - QA
- Developers responsible for “unit testing”
  - They deliver unit tests to test team once milestones are reached

Test Roles

- Small projects
  - Independent testing not possible
  - Very hard to carry out destructive testing
- Don’t just rely on Unit Tests.
  - You will miss things

Maintenance

- Management of defects and bug-fixes
  - Large projects require change management

- Defects are variance between expected and actual
  - Not necessarily a bug, or a crash

- Certain requests for changes could be classed as a defect

Once the software is out there…

- It’s hard to anticipate all problems
  - You can’t guess every use of the product
  - You can’t test for everything
  - Users will find defects that aren’t bugs!

- Effective testing should trap most defects
  - Bugs will get through
  - You and your customers will find defects

What To Do

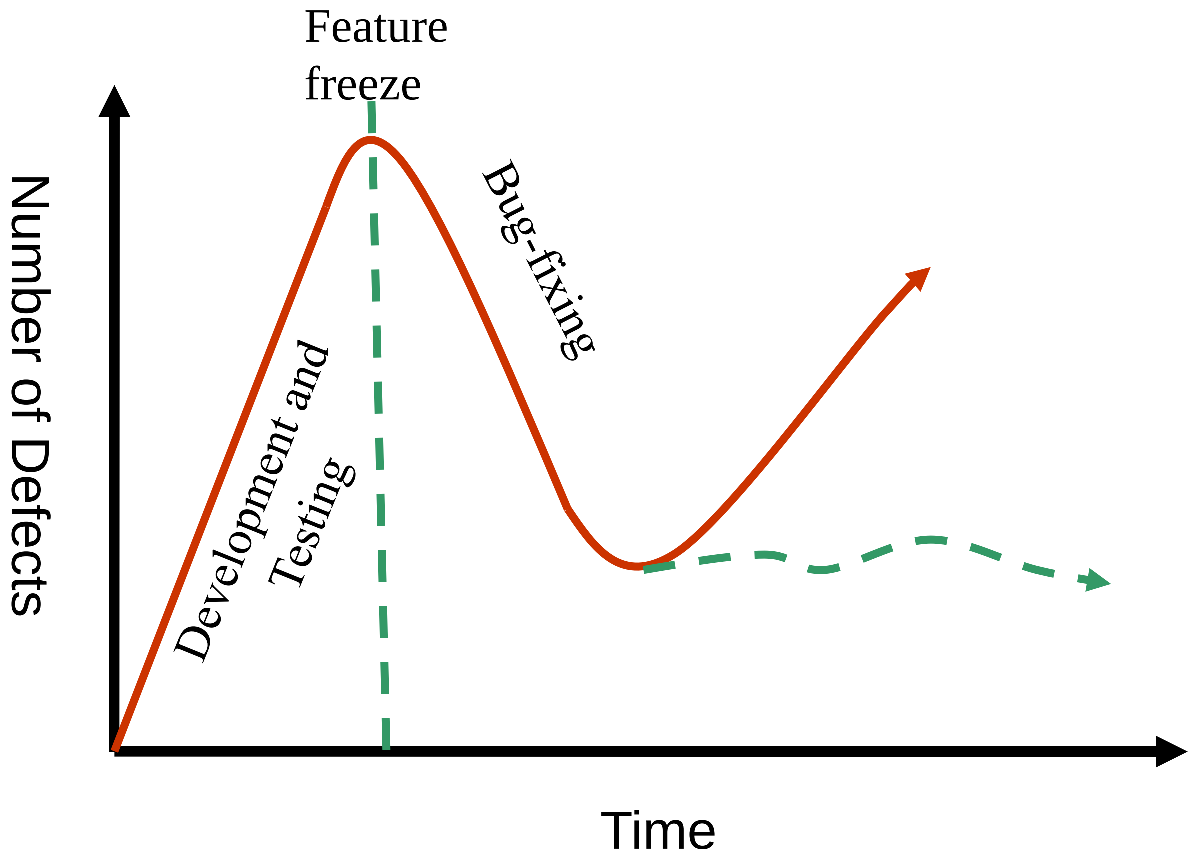
- Try and document them
  - Use some sort of bug tracking system
  - E.g. Bugzilla, GitHub, GitLab
  - Even an Excel Spreadsheet
- It’s important to prioritize fixes
- Use a debugger, add details to your “Problem Reports”
- Write a regression test

Example

- http://dashboard.cp2k.org

- http://buildbot.nektar.info

Defect creep



Key points of Defect Creep

- When fixing bugs
  - We are writing more code
  - Therefore we introduce more defects
- Defect creep: new bugs are introduced by bug fixes.
- Use change management and regression testing to prevent defect creep
  - Minimize number of new defects introduced by bug-fix work.

Minimize Change Impact

- Only make a change/fix if necessary
  - Risk management
  - Beware of quick wins!

- Never, never, never make a change without
  - Consultation and
  - Documentation

Minimize Change Impact

- Treat all changes/bug-fixes as mini-projects
  - Design, test, re-test
- Make sure you use revision control
- Consider creating a “bug-fix” stream
  - Maintains integrity of a “release”
  - Easy to spot if a fix “muddies the water”
  - Knowledge of a revision control system
- All help reduce “Defect Creep”

Summary

- Good organisation improves quality
  - Code, tests and data
- Third party testing is important
- Metrics and bad tests can be misleading
  - Have test reviews
- Don’t make hasty changes

Practical

- In the remaining time…
- Split into 4 teams
- Each time work on a testing strategy for:
  - Unit test
  - System test
  - Integration test
  - Acceptance test
- Consider:
  - When will you test?
  - What will you test?
  - Which back-end will you run on?
- Leave 15 minutes to present to other groups