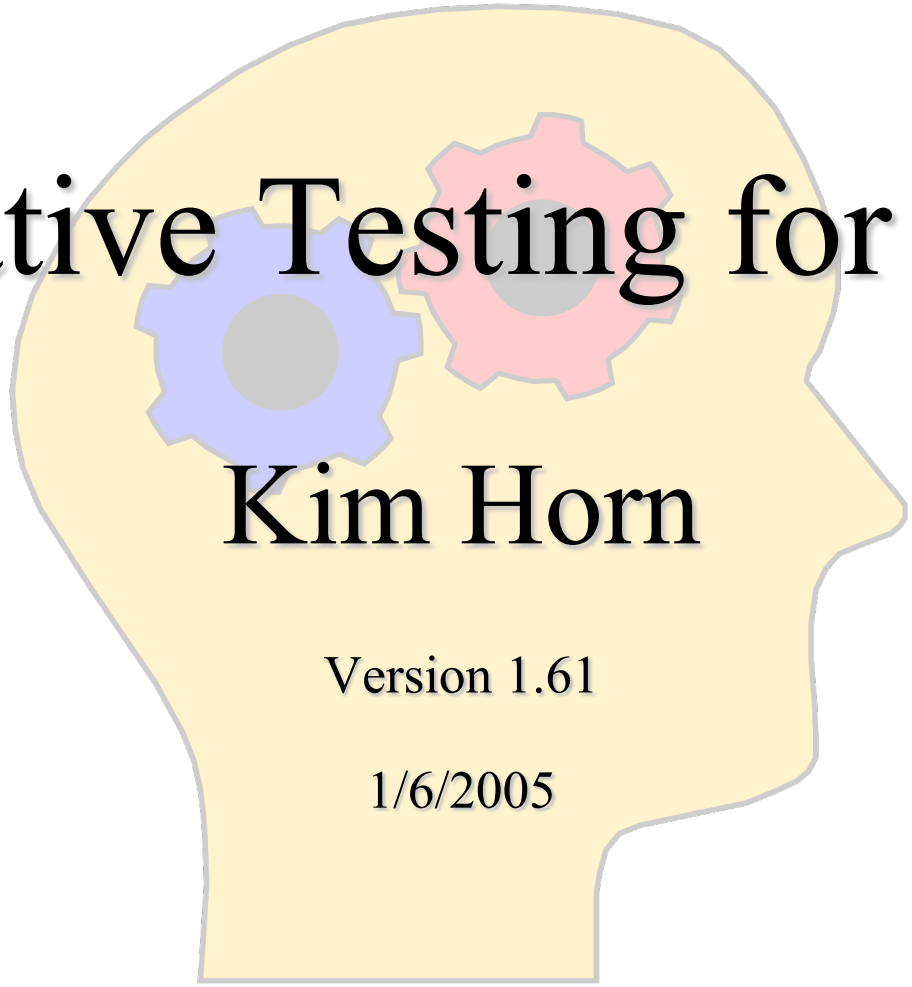
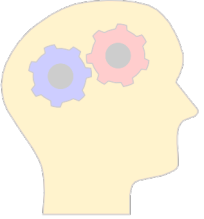
Iterative Testing for Java



Kim Horn

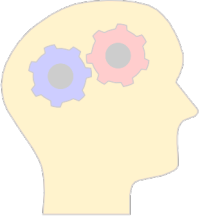
Version 1.61

1/6/2005



Agenda

- Scientific Method
- Iterative Processes
- Types of Testing
  - Unit
  - Integration
  - System
  - Functional
  - Stress
  - UAT
- Other types of test
- Platforms (Environments)



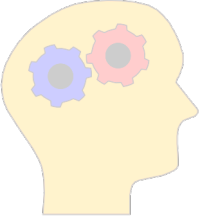
The Scientific Method

Science advances via a series of failures with a cycle:

1. A theory is proposed
2. An experiment tests the theory by attempting to falsify the theory.
3. If the experiment fails the theory is amended
4. Go back to 2

In the same way software systems are theories on requirements and are tested with experiments (test cases)

Workflows and Phases



Core Workflows

Requirements

Architecture

Realization

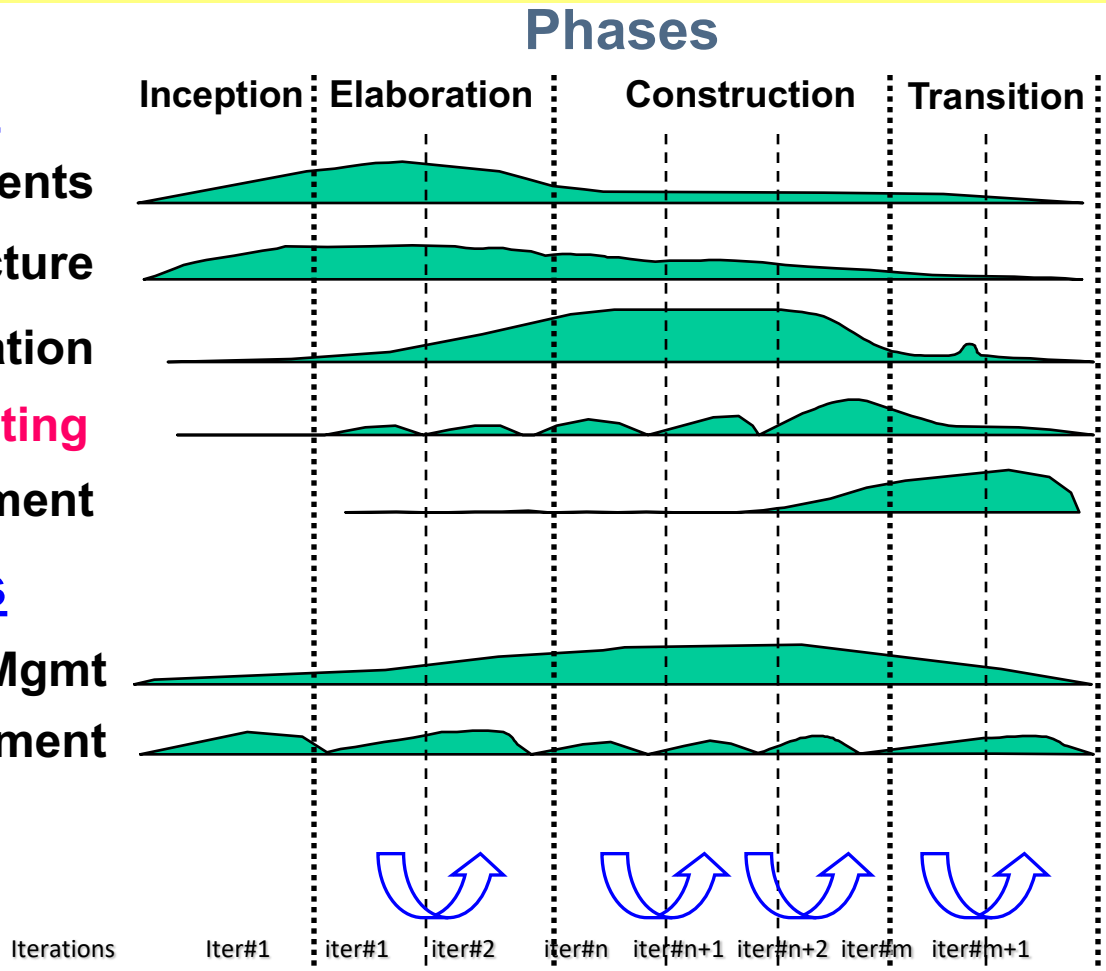
Continuous Testing → Testing

Deployment

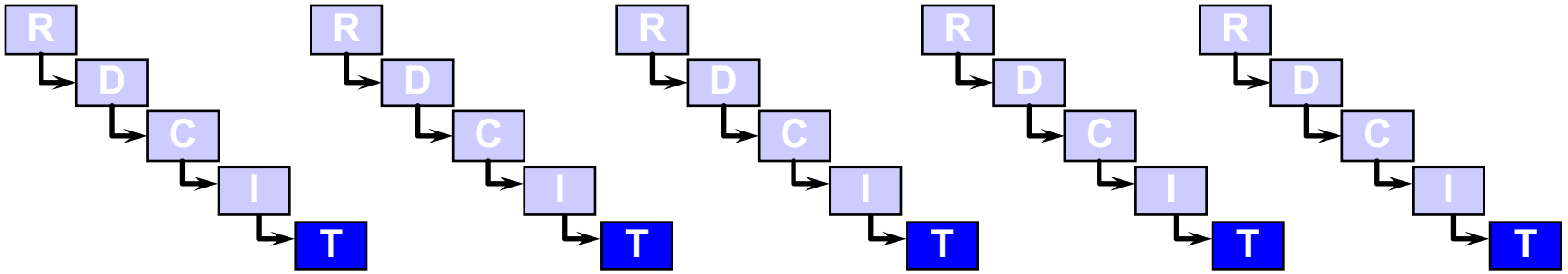
Supporting Workflows

Configuration & Change Mgmt

Project Management

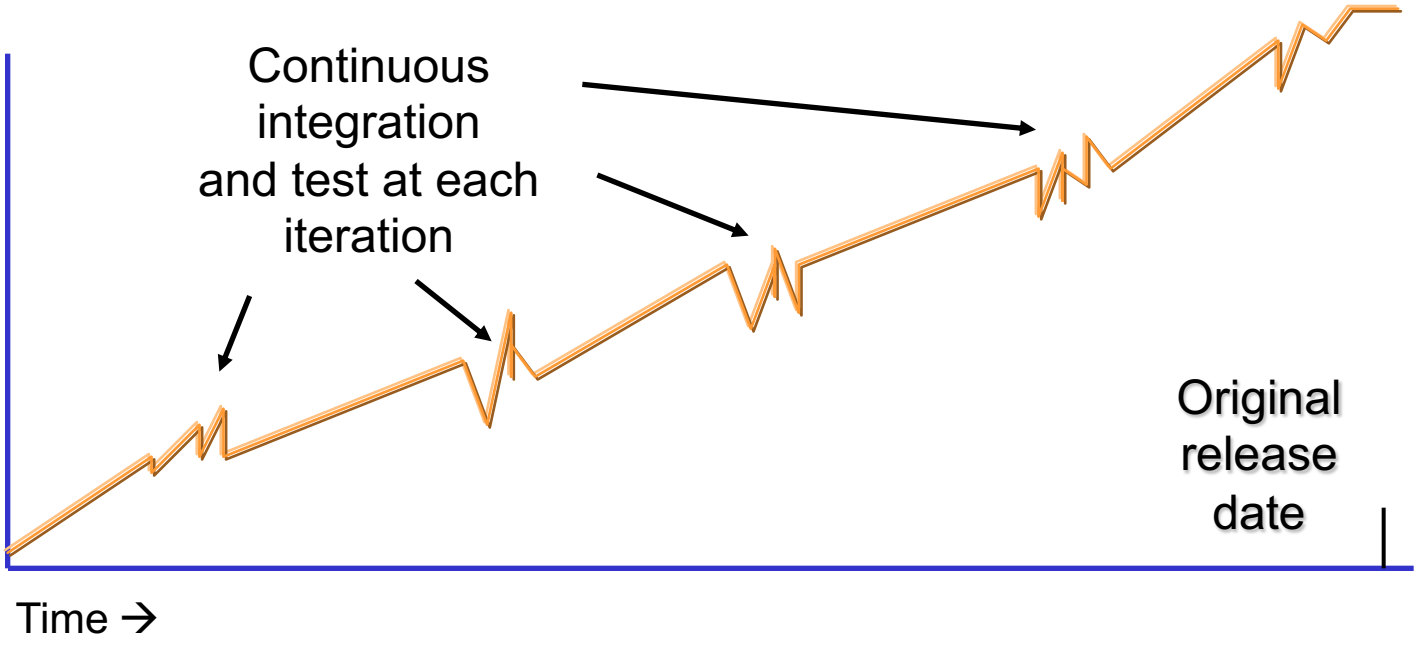


Iterative = Continuous Testing



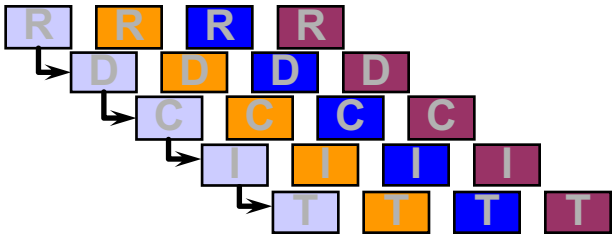
Progress

- Running Software
- Requirements Delivered
- **Tests Passed**



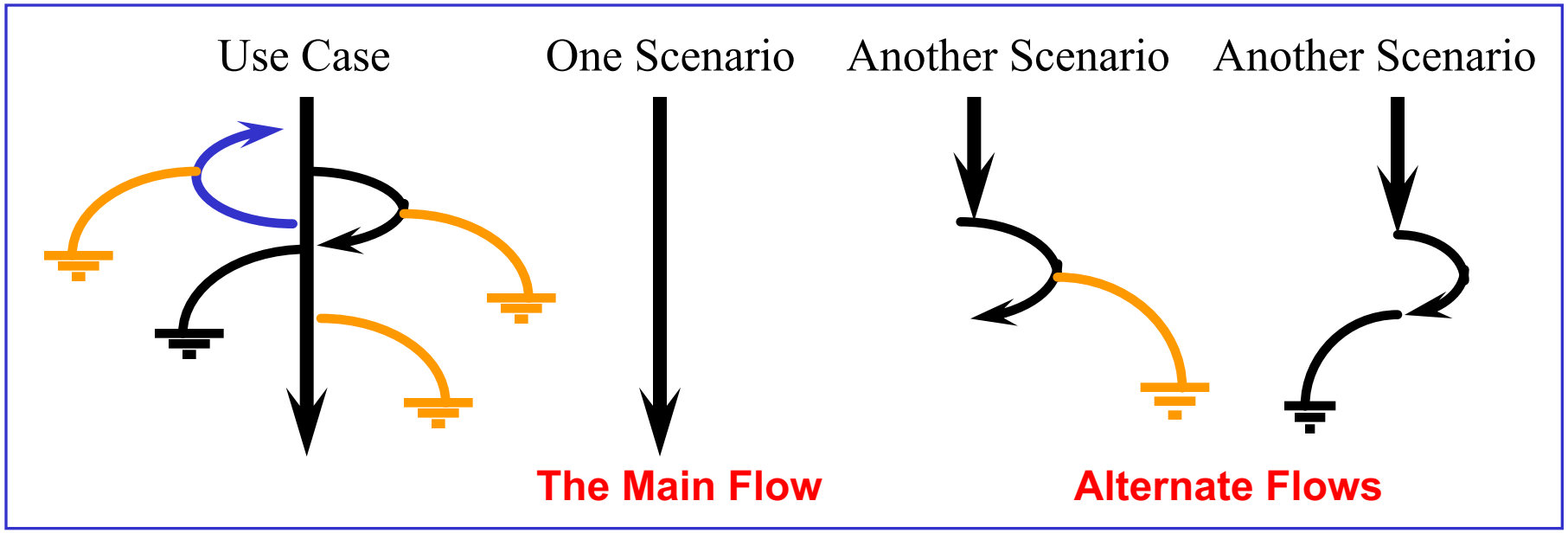
Verification = Progress

- Instead of complex measures of *earned value* based on intermediate artefacts (documents), that the business does not want, we substitute a simple *objective* measure of progress: **Running software in which a number of requirements have been implemented and verified (tested).**
- These completed requirements represent the real value to the business that the project has earned.
- Continual integration and test, means there are no surprises.
- Objective measure characteristics:
  - Incrementally increasing amounts of verified functionality;
  - Continuously improving product quality;
  - Continuous risk reduction;
  - Reducing levels of change; and
  - Increasingly accurate estimates.

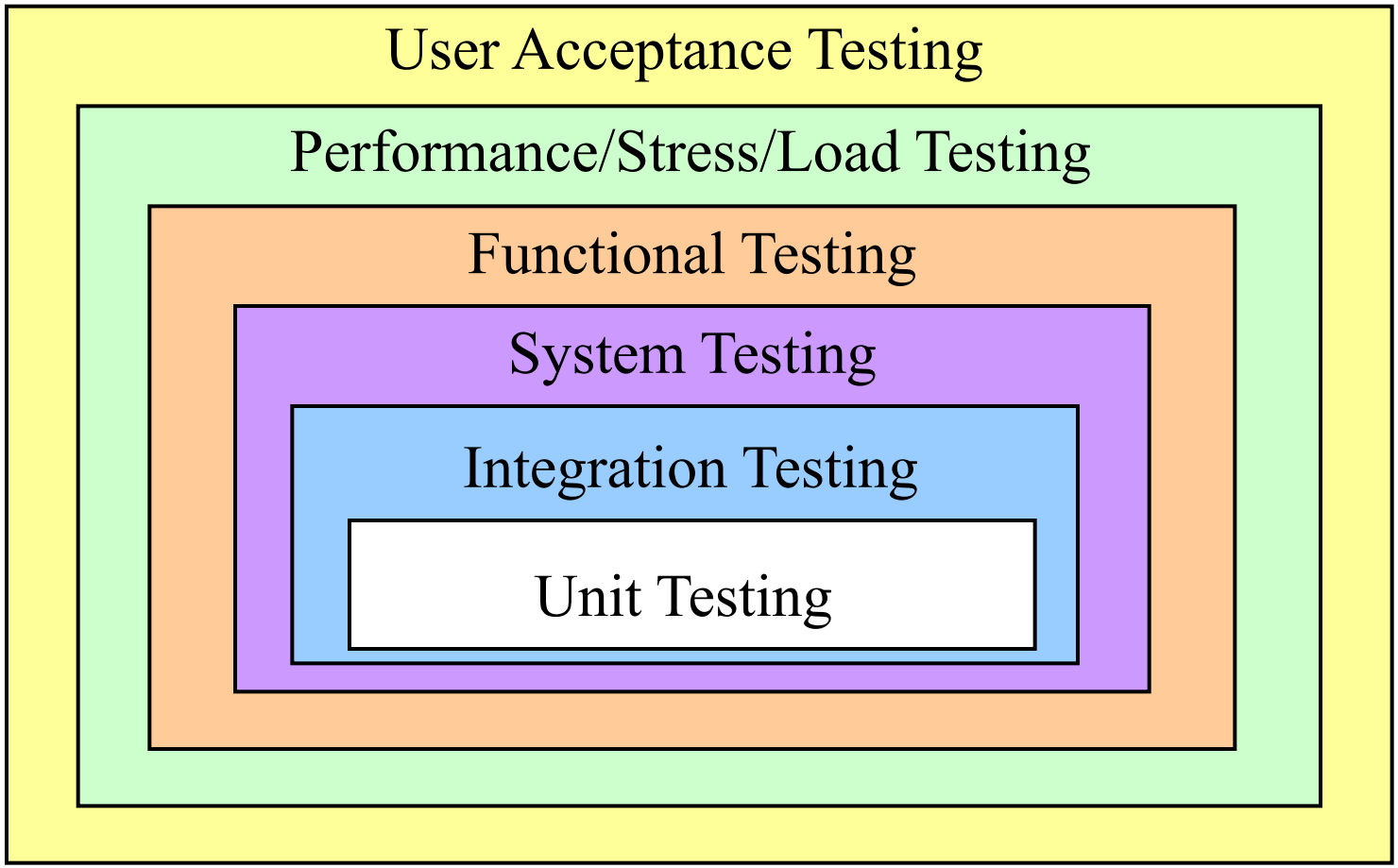


Units of Iteration

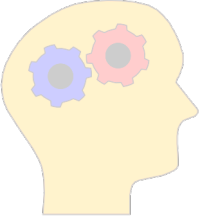
- Scenarios not Uses Cases form the units of iteration.
- A scenario is an instance of a use case, the main flow or one of its alternate flows.
- Scenarios are concrete examples. e.g. “Bill logs into the system and buys item X…..then pays with Visa card”



6 Main Types Of Testing

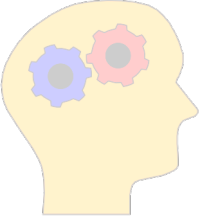


Time extends outward from centre



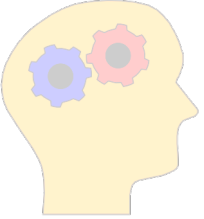
Unit Testing

- Test code from inside: White Box Testing.
- Done by developers.
- 3 Types of Unit Testing:
  - **Logic Unit Test:** test the code logic, usually method by method. e.g. test my code method.
  - **Integration Unit Tests:** test interaction between components, e.g. test my code method with data base.
  - **Functional Unit Tests:** extend boundaries out further to a full stimulus-response, e.g. web page, my code and data base.
    - These do not test full workflow as per true functional test.
- Mock objects are used to control the test boundaries. These are stand-in objects that return results you want; they stop interactions happening.
- **Open Source Tools**: jUnit, jMock, StrutsTestCase, etc….
- **Test Coverage Tools**: Clover
  - Reports on how much of jUnit code is tested ?



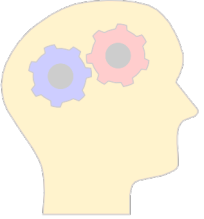
Test Driven Development (TDD)

- **Rule:** No code is written until a test fails.
- **Process:**
  - **1:** Write a test.
  - **2:** Check the test fails.
  - **3:** Write/refactor your code. Only enough code to pass test.
  - **4**: Check if the test passes. Finished.
  - **5**: If not: refactor code, go to 4.
- **Benefits of unit testing and TDD:**
  - Tests help understand goal of code.
  - Focus developer to reduce over-development.
  - Permit refactoring - we can change code and know the effects.
  - Prevent regression – limit need for debugging.
  - Improve test coverage, over functional tests. Functional tests (Black Box) cover about 70% of an applications code.



TDD & Scientific Method

- TDD works by providing a test that fails, demonstrating the system (theory) needs amendment.
- New code is written to make the test pass.
- A new bug or system crash is the environment falsifying our theory, it requires amendment.
- Quality Systems are built scientifically, both functional and non-functional tests are required.

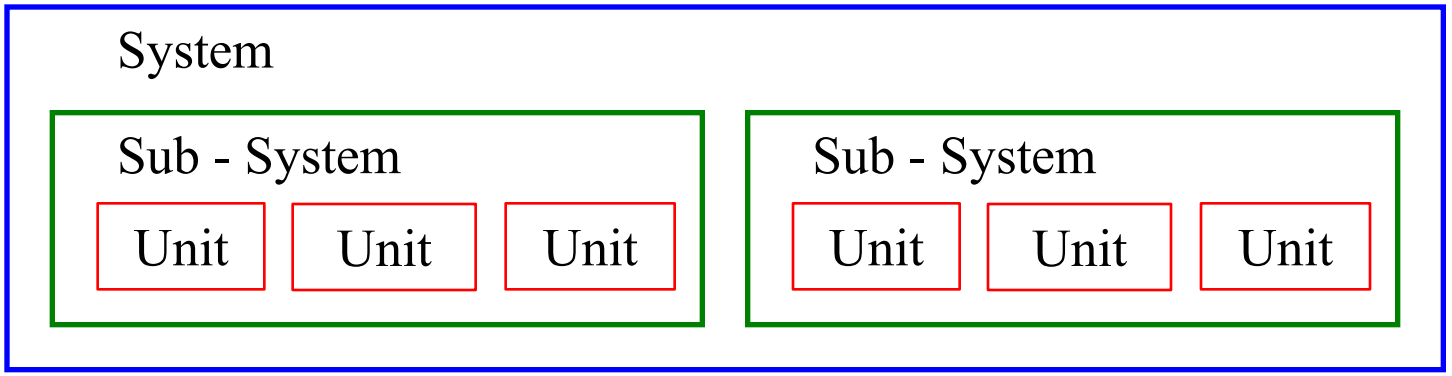


Integration Testing

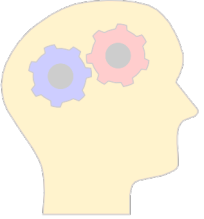
- What happens when we combine the units into more of the workflow; now lets look at the interactions.
  Can combine units of code together and they break – unit testing is NOT enough.
- 3 Types of Integration Testing:
  - **How objects interact:** test one of more objects/methods together
  - **How services interact:** test within a target container (server), interact with data bases, resources or legacy.
  - **How subsystems interact:** join the layers together. Do requests work from front end to back end.
- When all components are fully integrated a **System Test** can be done.
- **Open Source Tools:** jUnit, jMock, StrutsTestCase, etc….
- **When:** May happen multiple times in an iteration, for each build.

- **Best Practice**: Continuous Integration: the software development process, of frequent automated integration builds
  - significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

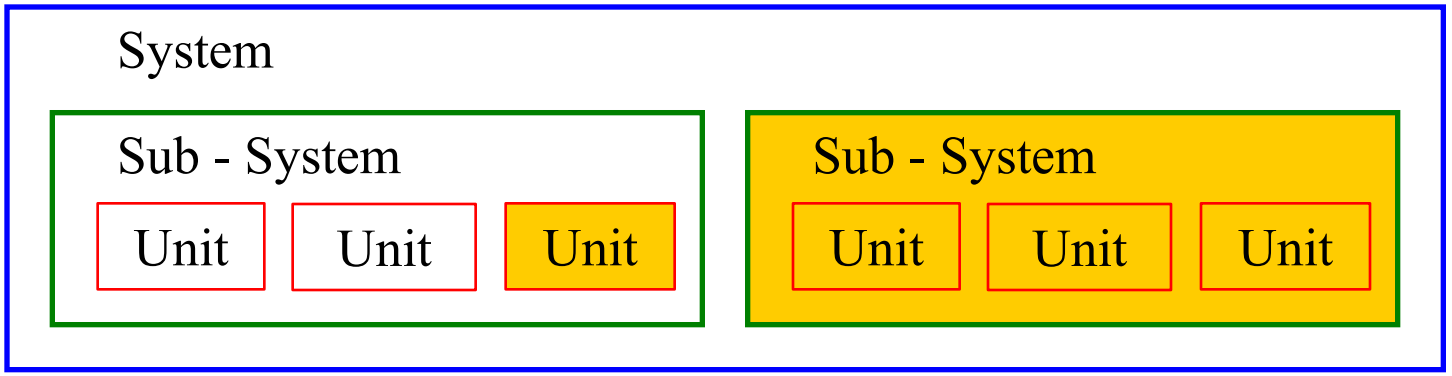
System Testing



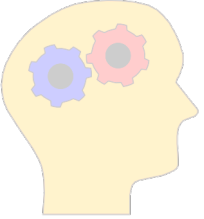
- When enough bits are integrated we get a sub-system. Eventually we get the completed system.
- Black Box testing.
- **When:** At the end of each iteration.
- **Open Source Tools:** jUnit, jMock, StrutsTestCase, etc….



Stubbing and Mocking



- Difficult to anticipate the results external systems may provide.
- Certain parts of system may be available only as real services and cannot be used for testing.
- Stubs can be used to simulate and thus isolate parts from testing.
- **Mocking is more fine grained isolation;** such as individual method calls and Mocks don't implement any logic.
- A mock object is an object that your code will be collaborate with and deliver defined results.

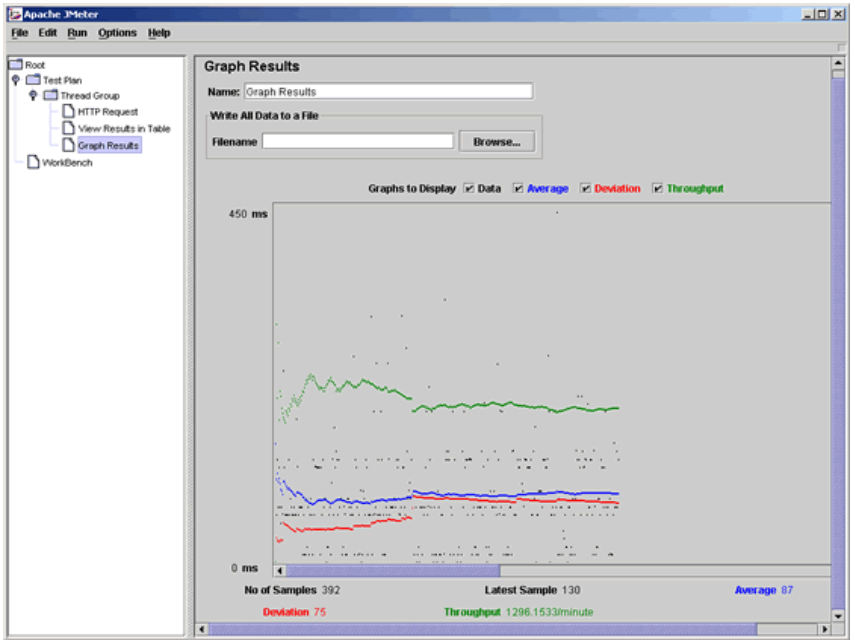


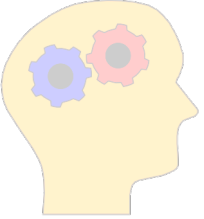
Functional Testing

- When we test the interactions are the result what we expect – usually not ?
- Now we are testing the scenarios and use cases; the workflows.
- Black Box testing, we only see the boundaries or APIs.
- 3 Types of Functional Testing:
  - **Framework:** test the framework API.
  - **GUI:** test all aspects/features of GUI. May consequently go all the way to back end. Use tools (robots) to run the GUI's, Web Pages; simulate user.
  - **Subsystems:** A layered system provides API for higher layers. Check the contracts are enforced.
- Open Source Tools: jUnit, StrutsTestCase, etc….
- ***TDD keeps your code clean – but does the code do what the stakeholder wanted. Does it meet the requirements ?***

Performance/Stress/Load Testing

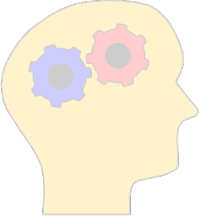
- It is important to have an application that functions correctly but can it meet the SLAs (number of users, response times).
- How does it perform a use case/scenario with many users at once, under load.
- Stress test environment is as close as possible to production environment.
- **Failure Test:** load all the way to failure. What fails ?
- **Open Source Tools:** jMeter
- Profilers: look at bottlenecks.
- **JUnit** can be extended with **JUnitPerf** to perform tests in specified time.





Performance Testing

- Goal: is not to find bugs, but to eliminate bottlenecks and establish a baseline for future regression testing;
- Can be used to establish a “benchmark”;
- A carefully controlled process of measurement and analysis. Ideally, the software under test is already stable enough so that this process can proceed smoothly;
- A clearly defined (and measurable) set of expectations is essential (e.g. SLAs).
  - expected load in terms of concurrent users or HTTP connection;
  - acceptable response time;
- Mostly done with Black Box; load the application by running concurrent users across use cases;
- If it fails then look inside (white box);

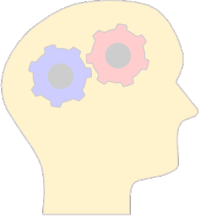


Performance Testing

Once you know where you want to be, you can start on your way there by constantly increasing the load on the system while looking for bottlenecks.

These bottlenecks can exist at multiple levels:

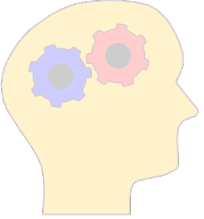
- **App level,** developers can use profilers to spot inefficiencies in their code (for example poor search algorithms)at the database level, developers and DBAs can use database-specific profilers and query optimizers
- **OS level**, system engineers can use utilities such as top, vmstat, iostat (on Unix-type systems) and PerfMon (on Windows) to monitor hardware resources such as CPU, memory, swap, disk I/O; specialized kernel monitoring software can also be used
- **Network level,** network engineers can use packet sniffers such as tcpdump, network protocol analyzers such as ethereal, and various utilities such as netstat, MRTG, ntop, mii-tool



Performance Testing

If the system still doesn't meet its expected performance goals, then the cycle **"run load test->measure performance->tune system"** is repeated until the system achieves the expected levels. A wide array of tuning procedures is available:

- Tune the JVM (e.g. garbage collection)
- Use Web cache mechanisms, such as the one provided by Squid
- Publish highly-requested Web pages statically, so that they don't hit the database
- Scale the Web server farm horizontally via load balancing
- Scale the database servers horizontally and split them into read/write servers and read-only servers, then load balance the read-only servers
- Scale the Web and database servers vertically, by adding more hardware resources (CPU, RAM, disks)
- Increase the available network bandwidth



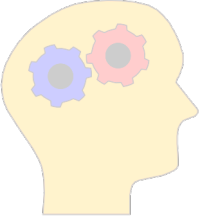
Load Testing

Load testing is usually defined as the process of exercising the system under test by feeding it the largest tasks it can operate with. Load testing is sometimes called volume testing, or longevity/endurance testing. Goals:

- expose bugs that do not surface in cursory testing, such as memory management bugs, memory leaks, buffer overflows, etc.
- ensure that the application meets the performance baseline established during performance testing. This is done by running regression tests against the application at a specified maximum load.

Examples of volume testing: Send 1000 concurrent users to a web system 100 times exercising a known use case.

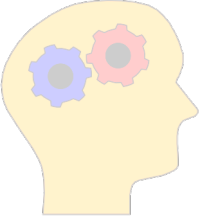
Examples of longevity/endurance testing: Check a client-server application by running the client in a loop against the server over an extended period of time



Load Testing

Although performance testing and load testing can seem similar, their goals are different.

- Performance testing uses load testing techniques and tools for measurement and benchmarking purposes and uses various load levels.
- Load testing operates at a predefined load level, usually the highest load that the system can accept while still functioning properly.
- Load testing does not aim to break the system by overwhelming it, but instead tries to keep the system constantly humming like a well-oiled machine



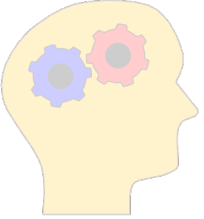
Failure or Stress Testing

Stress testing tries to break the system under test by overwhelming its resources or by taking resources away from it.

Purpose is to make sure that the system fails and recovers gracefully: systemic quality called recoverability.

Examples:

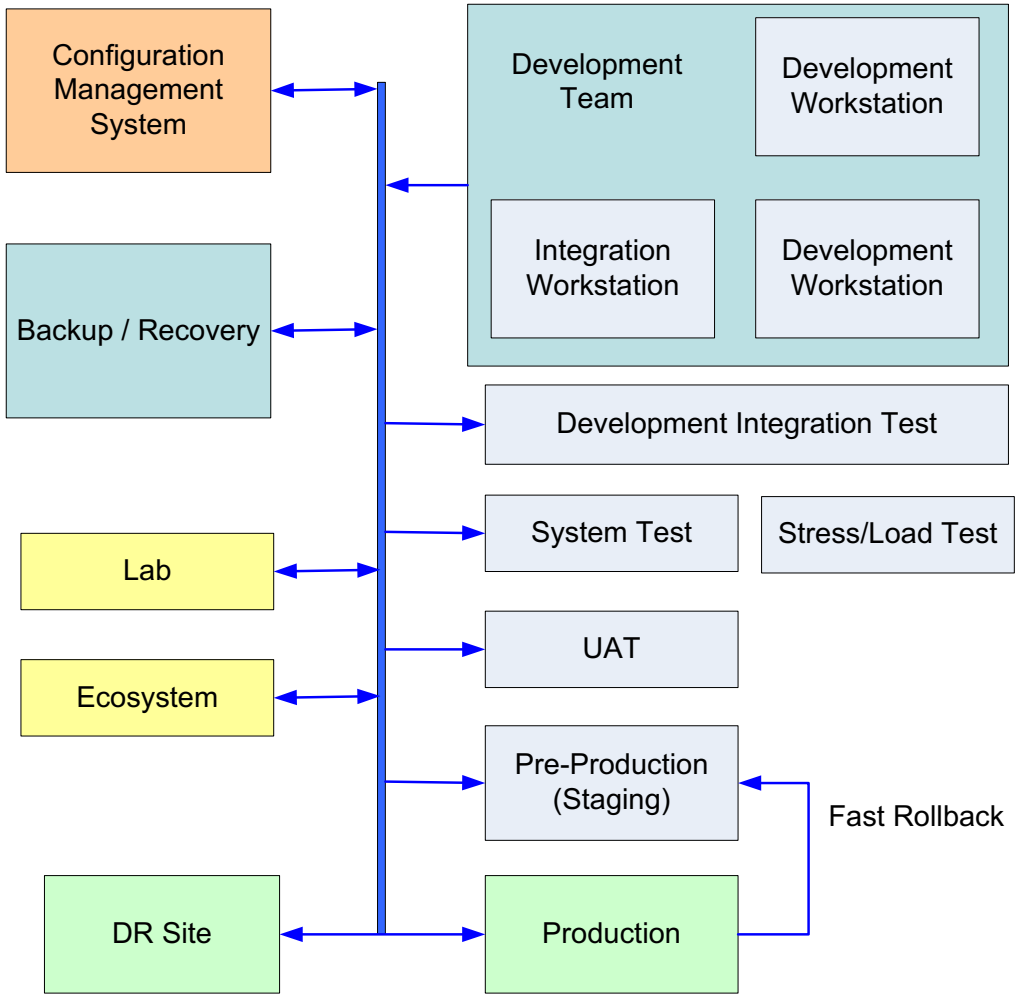
- double the baseline number for concurrent users/HTTP connections
- randomly shut down and restart ports on the network switches/routers that connect the servers (via SNMP commands for example)
- take the database offline, then restart it
- rebuild a RAID array while the system is running
- run processes that consume resources (CPU, memory, disk, network) on the Web and database servers

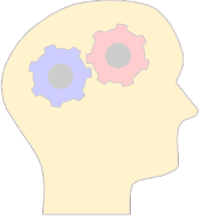


Acceptance Testing

- An application may do what its meant to do, perform well, but will the users accept it ?
- Does it meet the customers (business) needs.
- Usually ***independent*** of other tests; done by customer.
- ***Independence*** is important as it removes biases. *Developers and TDD are myopic, they do not see the big picture.*
- Which tests:
  - Superset of all previous tests.
  - Whatever the users wants.
  - Other qualities:
    - Usability
    - Look and feel
- **Open Source Tools:** Fit and Fitnesse

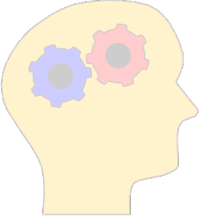
Platforms (Environments)





Other tests

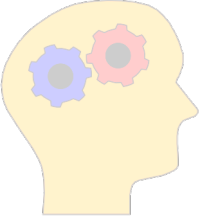
- **Configuration tests:** test different configurations of system.
- **Installation tests:** test the system runs correctly when installed on customer platform.
- **Security tests:** test how secure the system is.
- **Smoke tests:** preliminary to further testing, which should reveal simple failures severe enough to reject a prospective software release.
- **Disaster** recovery testing.



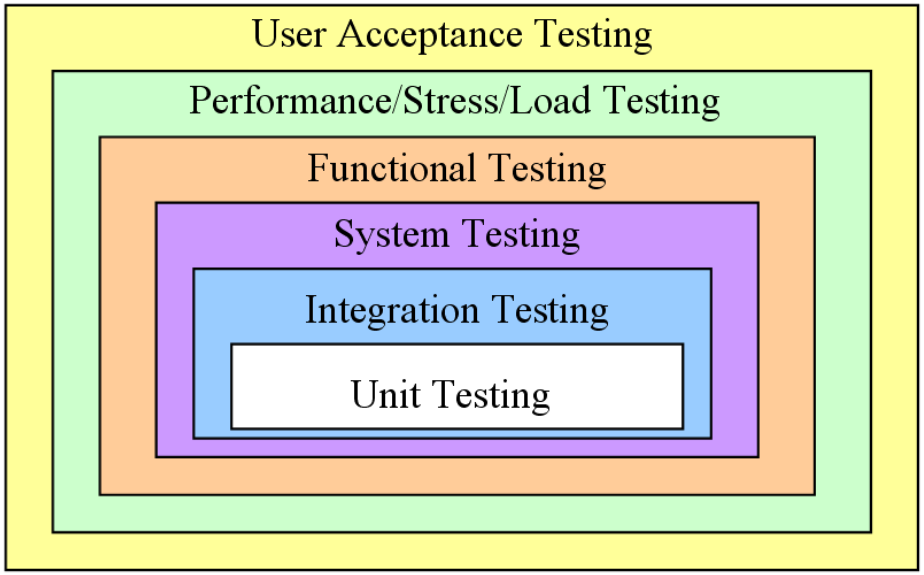
Other Open Source Tools

There are over 200 tools out there, some are:

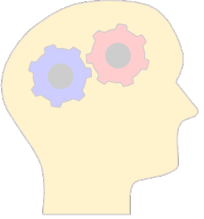
- JMock, EasyMock, DynaMocks…many other mocks
- HttpUnit
- ServletUnit
- CruiseControl
- jWebUnit
- Cactus
- Struts:
  - StrutsTestCase
  - MockStrutsTestCase
  - CactusStrutsTestCase
- AntEater, AntHill
- Maven
- Fit and Fitness



Summary



- Use cases from the core unit of Iterative methodologies.
- Use case provide the basis for test cases using scenarios.
- Progress is based on verified tests passed
- Unit testing is at the core of Iterative approaches.
- Done well it reduces rework and unwarranted testing down the line.
- However alone it is not enough. The surrounding testing approaches are all essential.



The End

Thank you.

Do you have any questions?