

*Capability
Architects*

Testing in the SDLC



© [Capability Architects](#)



Testing In The Software Lifecycle: Agenda

It is impossible to test properly without a clear idea of what is happening during development as a whole.

- Overview of the software development lifecycle
 - The phases of software development.
 - The V-model.
 - The delivery process: Build strategy.
- Relationship between development & testing
 - The V-model of testing.
 - Levels of testing.
 - Independent testing.
- Engineering for testability
 - When is software testable?
 - Untestable software.



The Phases Of Software Development

The main phases in the standard software development lifecycle (SDLC) are:

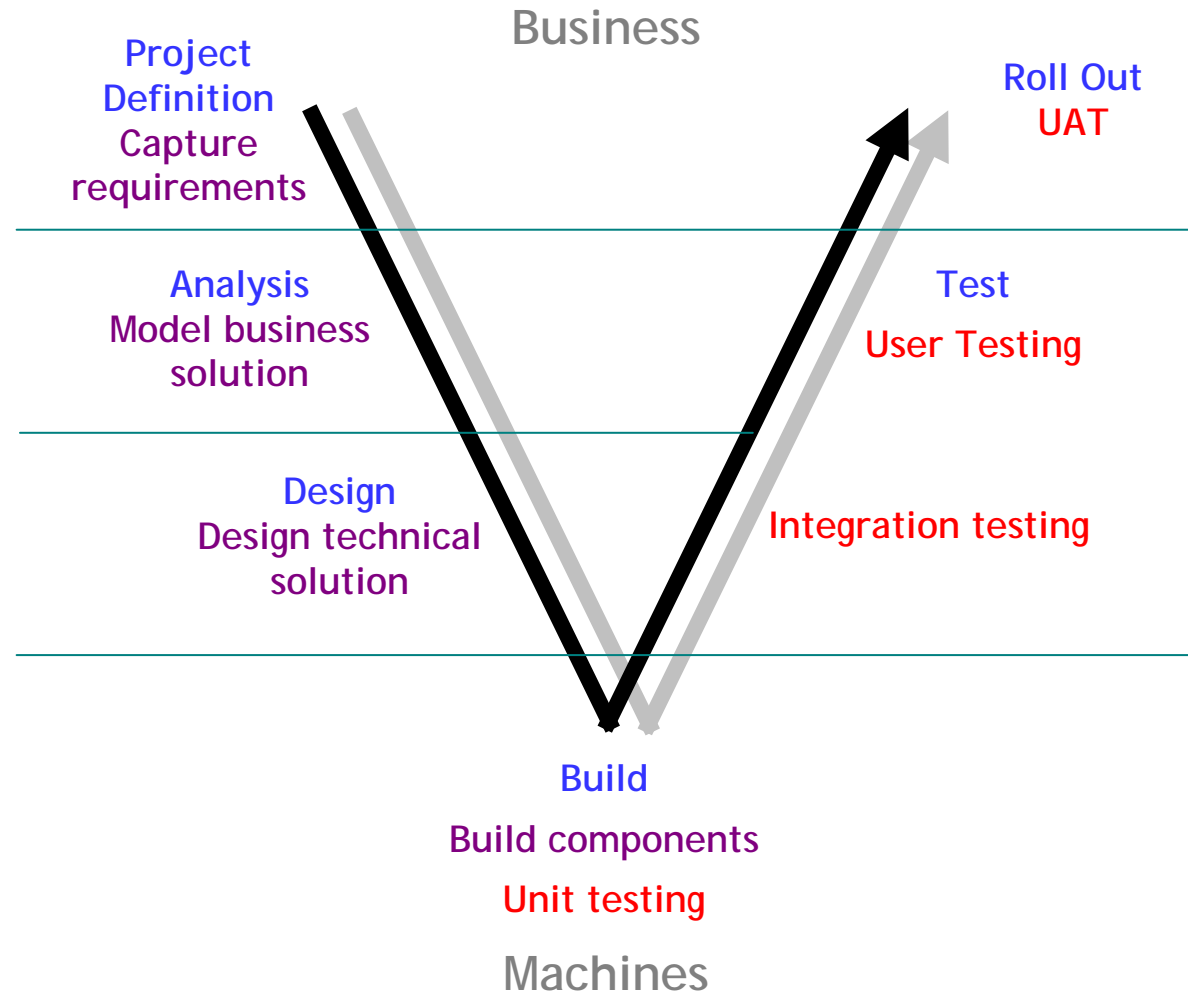
Plus others for managing, infrastructure, etc.



- Project definition
 - What the project is supposed to achieve.
 - Scope, stakeholders, strategic contribution, impact.
 - Project strategy, management team, feasibility.
- Analysis
 - Define the business solution.
 - Detailed requirements, business models, prototypes.
- Design
 - Define how the solution will work.
 - Architecture, components, interfaces, platform, data.
- Build
 - Build the solution.
 - By implementing the design.
- Test
 - Does it do the right thing? Does it do the thing right?
 - Do developers, business & users agree?
- Roll Out
 - Prepare production environment, transfer application.
 - Realise & validate business benefits.

The V-Model

The best known model of the software development process is the so-called 'V-model'



Key:
Phase
Development activity
Test Activity



The Delivery Process: Build Strategy

Nowadays, few significant projects are delivered all at once.

Instead, they appear in progressive builds.

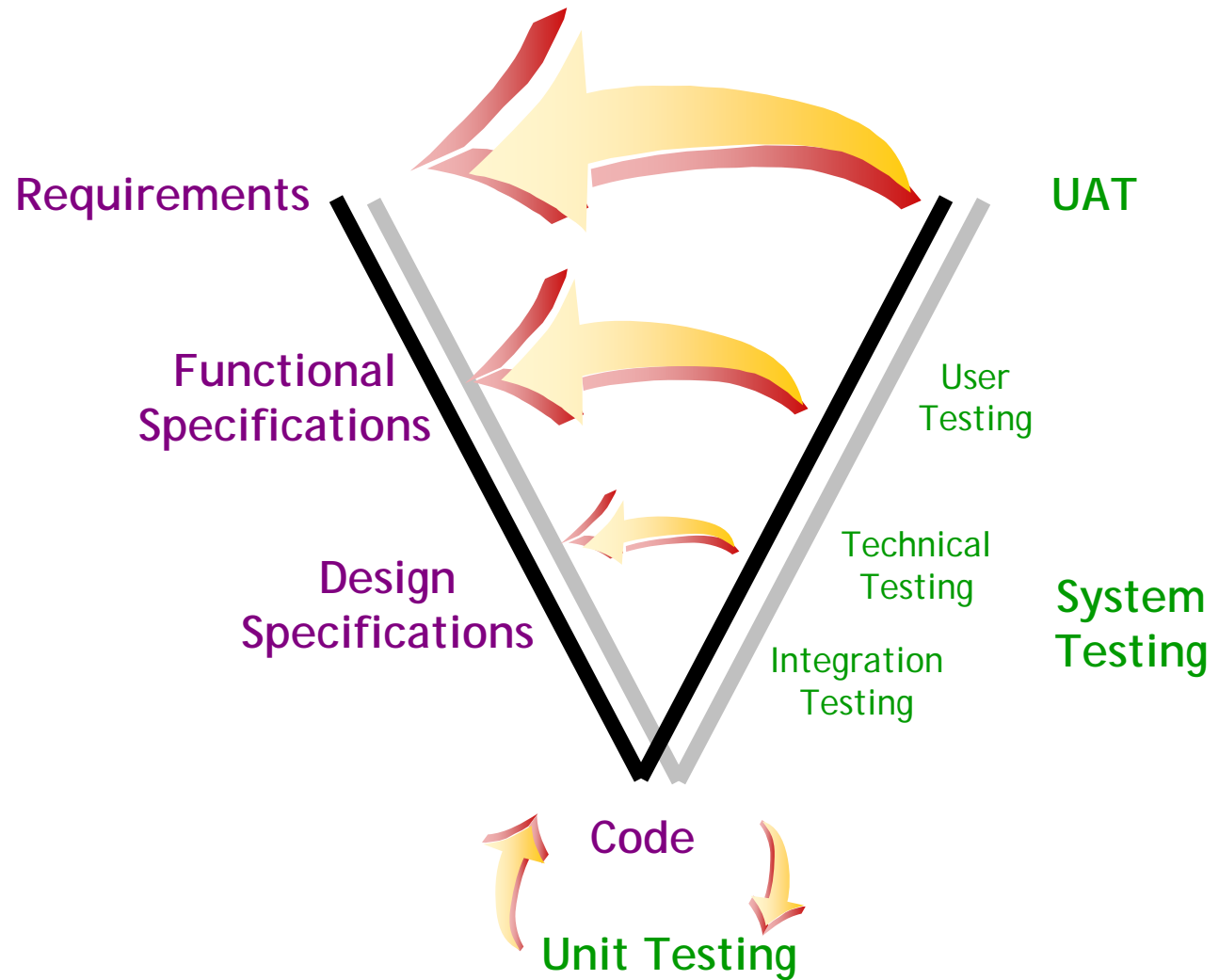
- A build consists of the deliverables needed...
 - To realise a set of requirements.
 - To validate that they work to the business's satisfaction.
- Multiple builds enable the business...
 - To gain planned benefits as quickly as possible.
 - New requirements, changing competition, new technologies, etc.
 - To validate its strategy as early as possible.
 - Through build-related V+V programmes.
- Builds enable the development/test process...
 - To be split into manageable 'chunks'.
 - To enable the development process to respond to changing business conditions.



The V-Model Of Testing

Testing folds the development process back on itself.

Typically, this sequence will be followed for each build.



Unit Testing

Objectives

- To check that all components meet their specification.
- To check that all individual components work in isolation.



- Approach
 - Verify code logic & data structures.
- Typical tests
 - Screen interactions, such as hot spots, dialog boxes, widgets and radio buttons.
 - Mouse behaviour.
 - Hyperlinks.
 - File creation.
 - Data field editing & validation.
 - Calculations.
 - Message handling.
 - Document & report formats.
 - Data updating.

Exactly how much can be tested at this level depends on how the system is designed.

Integration Testing

Objectives

- To check the application against its design.
- To check that the application works.

- Approach
 - Check programs, inter-program communications, batch flow, on-line dialog.
- Typical tests
 - Application integration.
 - Job streams.
 - Internal component-to- component interfaces.
 - Resource availability & contention.
 - Networks, internet, communications, etc.
 - Support functions.
 - Interfaces with other applications.
 - Batch processes.
 - System-level design features.

Constructing integration tests is as hard as designing and developing the application.



Technical Testing

Objectives

- To check that the system's performance meets its requirements.

- Approach
 - Check all applications + platforms + communications, etc.
- Typical tests
 - Run times.
 - Performance, security, reliability.
 - Load management & balancing.
 - Portability & multi-platform implementations.
 - Crash proofing.
 - Disaster recovery.

Technical testing does not check whether the system is acceptable to the business.



User Testing

Objectives

- To check that the business models have been fully implemented.
- To check that users are satisfied with the systems' usability.



- Approach
 - Check that business cycles, transactions & work flows work correctly.
- Typical tests
 - Usability.
 - Documentation, help.
 - Security.
 - End-to-end business processes.
 - Subjective application/system performance.
 - Audit trails.
 - User access.
 - Verification of documentation
 - Help
 - Manuals
 - Procedures.

User testing does not have to be done by users - but it is much better if it is.

User Acceptance Testing

Objectives

- To check the business requirements are all met.
- To check the business is satisfied with the system.
- To authorise the system.



- Approach
 - Testing executed by real users.
 - Testing in a simulated production environment.
 - Target: formal acceptance of the system for production.
- Typical tests
 - Core operational tests.
 - Performed on behalf of business management & Project Sponsor.
 - Performed by real users.
 - Similar to a user demonstration.
 - Usually brief.



Other Industry Test Types

There are other types of testing you will hear about within the industry.

Names and arrangement of phases, processes, etc, can also vary between projects & programmes



- Thread/string testing.
 - Testing a group of related modules/units.
 - Designed to verify a specific function or sub-process.
- Front-end/End-to-end testing.
 - User interfaces only vs full system.
- Production Verification Testing.
 - Shake-down on the production platform.
 - Verifies environment, procedures, key functions.
 - Alternative platforms, communications, etc.
- BAU.
 - 'Business as usual' testing.
 - Maintenance/production support mode.
 - Defects, minor enhancements, infrastructure changes, new roll outs, etc.

Relationship Between Test Levels

Each level of testing should be able to take its predecessors for granted.

So subsequent levels should be able to take your testing for granted.

- No errors should be inherited from previous levels.
- You should be able to define entry criteria at each level.
 - Including parallel (eg, local or specialised) test programmes.
- These criteria should include:
 - Explicit status report.
 - Individual test results.
 - Coverage analysis.
 - Residual issues & risks.
 - Outstanding change requests.
- You should also be able to commit your own testing to rigorous exit criteria.



Independent Testing

Independent Testing is a test strategy in which pre-specified levels of testing are conducted independently of the developers.

Provides...	Because...
An egoless approach	Developers tend to confirm their software works. Independent testers focus on proving it works correctly - or has specific defects.
Detection of more errors of more types	Independent testers will be more skilled, structured & systematic.
Controlled & disciplined test management	Independent test management treats testing as an activity in its own right, separate from development goals.
A fresh perspective	A 'second pair of eyes'. Developers & testers are unlikely to share the same (potentially mistaken) preconceptions.
Objective results	Independent testers have no vested interest in a positive outcome.
A greater sense of responsibility among developers	It encourages developers to test their work more thoroughly because they know that it will also be tested independently.
User re-validation of requirements	Developing a test plan provides a fresh opportunity for uncovering changing expectations.



Engineering For Testability

Sometimes, it is extremely difficult - if not impossible - to test a system.

- When is software testable?
- Untestable requirements:
 - The symptoms
 - A practical - and common - example
 - The cure
- Getting involved in the development process.



When Is Software Testable?

In very general terms, an item is testable when...

- It can be traced from end to end of the development process:
 - So you tell whether it is actually the item intended.
 - From requirement to design to code to test case to test data, and so on.
- It can be given a practical example:
 - A step by step execution sequence.
 - Known input data transformed into known (or calculable) output results (data or event).
- It is consistent with all the other items at its level:
 - And is under formal change control, so it will stay that way!
- There is an SME who can field queries.



And remember, this must known for the **deliverable** version, not just the **current** version

Untestable Requirements: The Symptoms

An item is
untestable
when...

- It is inadequately defined:
 - There is no known practical example.
 - Where, when, why, what, who or how are unknown.
 - No SME exists for the actual or planned function.
- Requirements are:
 - Incoherent or inconsistent.
 - Incomplete or incorrect.
 - Ambiguous or incompletely 'decomposed'.
- Requirements imply a 'combinatorial explosion'.
 - Of test cases.
 - Of test conditions.
 - Of test data.
- Requirements include untestable steps:
 - Eg, requirements for informal users to behave like experts.
 - Eg, manual steps included in application requirements.
 - The developed item will not be accessible to the testers!



Untestable Requirements: A Classic Case

A classic untestable requirement is that 'the new system must perform as well as the existing system'.

This is reasonable as a business or system goal, but...



- Many existing systems have no measures of how well they perform now.
 - And no plans to discover how well they perform.
- The two systems are fundamentally different:
 - Why else are you replacing the one with the other?
 - Eg: the new system works differently (eg, on-line vs off-line data preparation)
 - Eg: The new system will have different users (eg, it decentralises currently centralised control)
 - Eg: the platforms are fundamentally incomparable (eg, internet vs mainframe)
- Etc., etc.



Untestable Requirements: The Cure

Untestable requirements can be exposed and remedied - but only if you take early action.

- Verify requirements for testability:
 - Make sure this is done when requirements are first written.
 - Make sure this is done by a real tester.
 - Preferably one who will have to test these requirements personally!
 - Make sure they ask 'test for what?'
 - Ie, to demonstrate a specific business benefit.
- Define practical test cases that would verify the requirement:
 - Does everyone agree it would be a valid test?
 - ... and what would be a pass or a fail?
- Use the business models to capture at least one practical example:
 - This will provide real test conditions and usable test data.
 - The fact that it exists will be very reassuring!



Getting Involved In The Development Process

However, the most effective remedy is to get actively involved in the development process:

- Understand the development process.
 - And align your own work with theirs.
- Establish pro-active relationships with your stakeholders.
 - Get to know their SME's.
 - They are the ultimate arbiters of the requirements.
- Participate in the development process:
 - From project initiation to roll out.
 - Insist on reviewing phase-end development deliverables.
 - Integrate your testing with theirs.
 - Provide expertise and consultancy wherever you can.



If you would like to know more...

Capability Architects provide the full range of testing support, including process design, recruitment and training.

- You can contact us at:
 - www.CapabilityArchitects.com
- You can speak to an experienced ISO 9000 consultant on:
 - +44 7962 227886
- Or email us at:
 - mail@CapabilityArchitects.com

