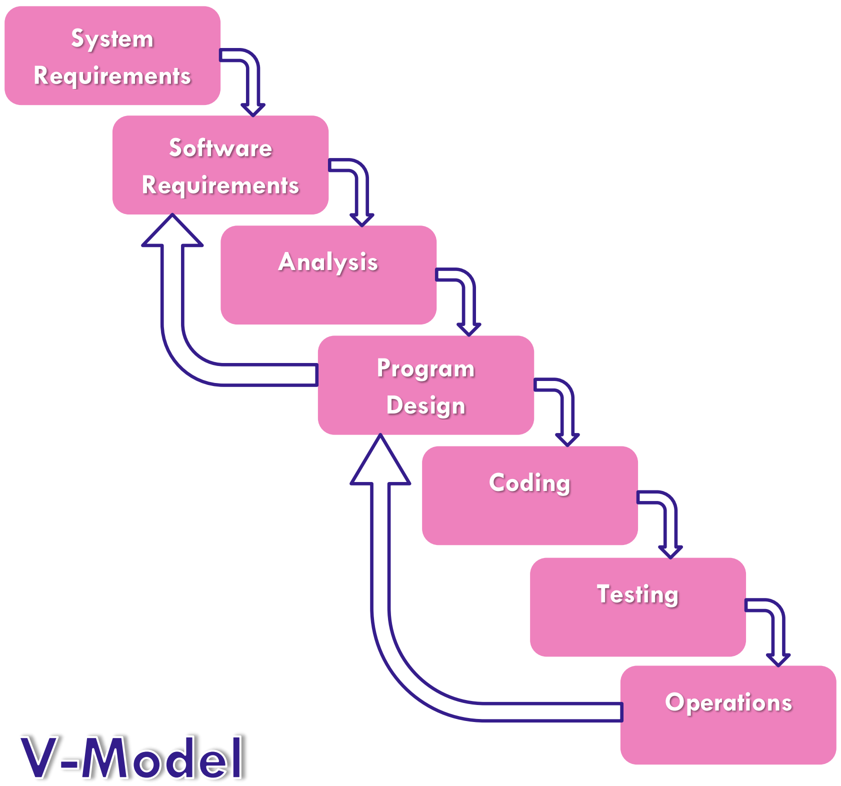
2. Life Cycles

A. CHARACTERISTICS OF GOOD TESTING IN SDLCS:

1. There is a corresponding test activity for every development activity
2. Each Test level has a specific objective
3. Test analysis and design for a given test level begin during corresponding development activity
4. Testers take part in discussions regarding defining and refining requirements and design and are involved in reviewing work product as soon as drafts are available (principle of early testing)

B1. SEQUENTIAL LIFE CYCLES:

Waterfall



System Requirements

Software Requirements

Analysis

Program Design

Coding

Testing

Operations

PROS:

Enforces discipline at each stage

Has a defined start and end

Progress can be easily identified

Emphasis on requirements before code is written means no wasted time and can improve quality

CONS:

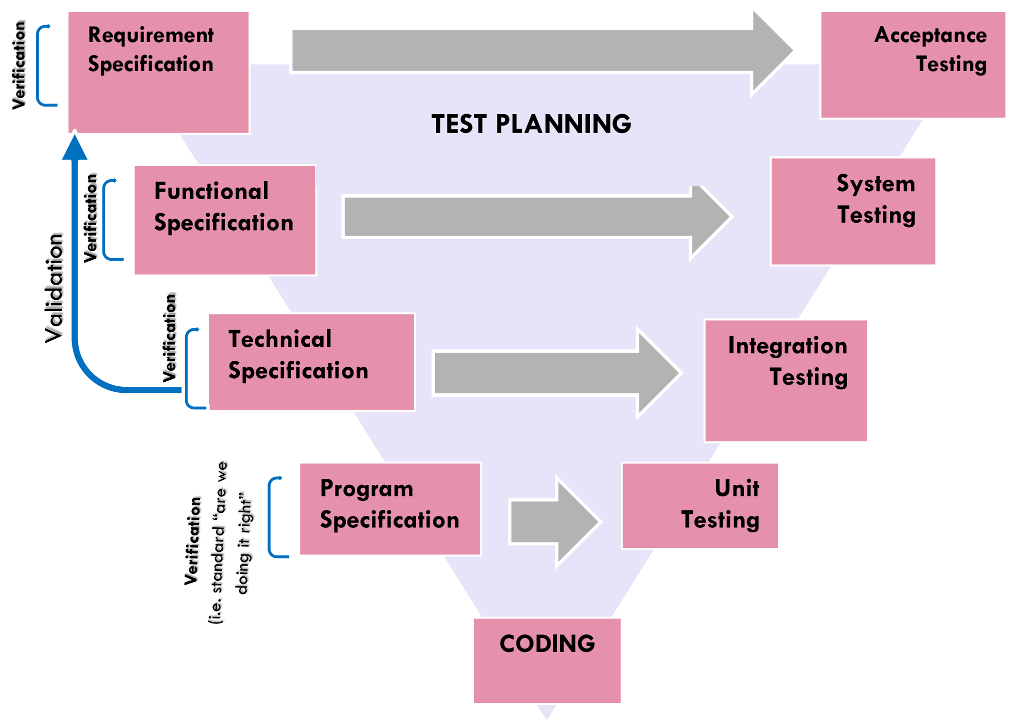
Estimations of time/cost will be difficult

Requirements (therefore tests) will change

Division of labour is unrealistic

What has asked to be created may not be feasible

V-Model



Verification

Requirement Specification

Acceptance Testing

TEST PLANNING

Validation

Verification

Functional Specification

System Testing

Verification

Technical Specification

Integration Testing

Verification (i.e. standard "are we doing it right"

Program Specification

Unit Testing

CODING

PROS:

Higher chance of success as test plans are developed earlier

Defects found earlier

Works very well on smaller projects.

CONS:

Quite rigid in execution

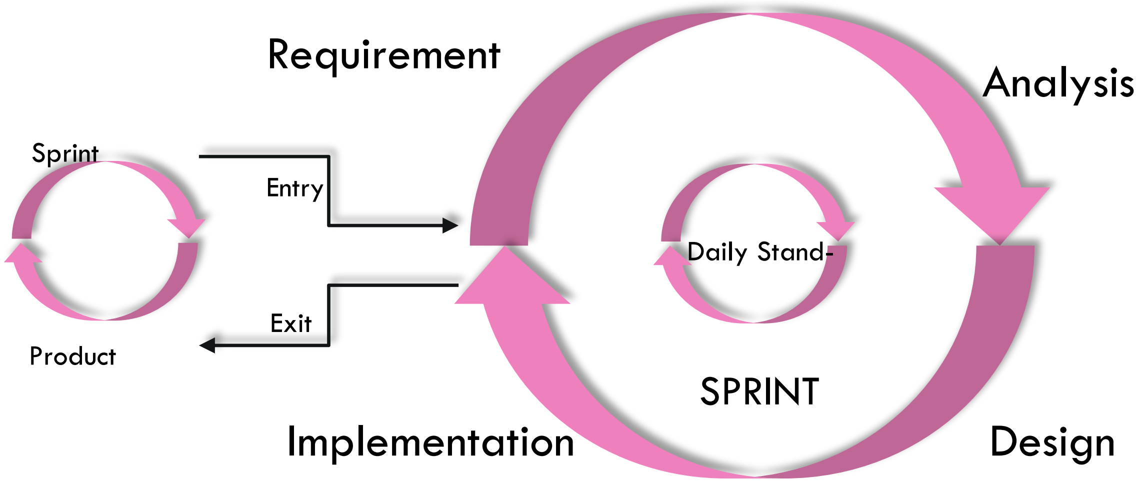
No early prototype

Test documents have to be updated along the way

***IMPORTANT**: There is also a 5th Testing Level in the V-model: "System Integration Testing" (**SIT**)*

B2. INCRIMENTAL LIFE CYCLES:

AGILE (e.g. Scrum, Kanban, RUP, Spiral)



Requirement

Analysis

Sprint

Entry

Daily Stand-

Exit

Product

SPRINT

Implementation

Design

PROS:

Small and frequent improvements

Fast deployment

Team skill improvement

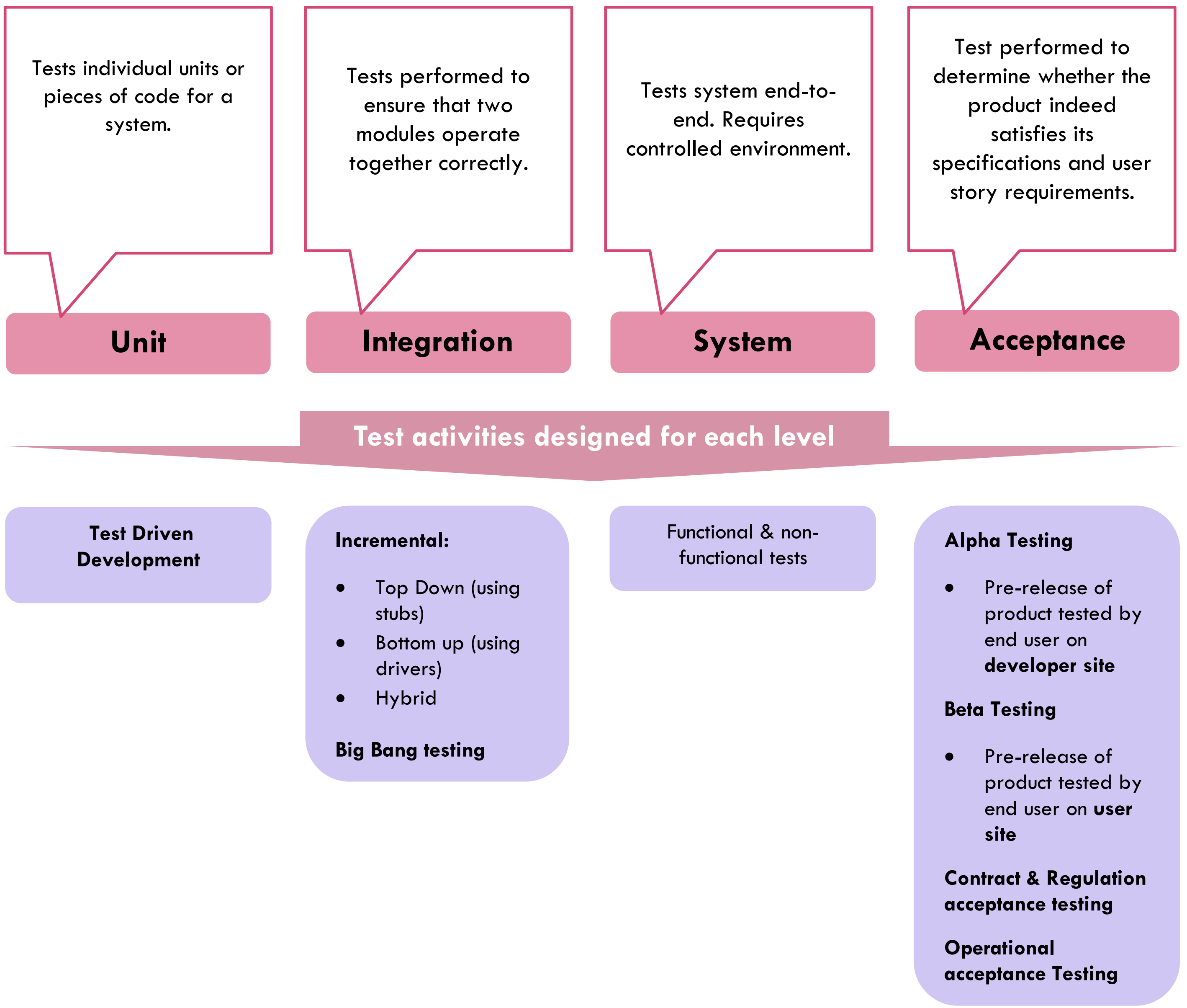
CONS:

Light documentation

Formal records of change may not be created

Regression testing may get out of control

C. TEST LEVELS:



Tests individual units or pieces of code for a system.

Unit

Tests performed to ensure that two modules operate together correctly.

Integration

Tests system end-to-end. Requires controlled environment.

System

Test performed to determine whether the product indeed satisfies its specifications and user story requirements.

Acceptance

Test activities designed for each level

Test Driven Development

Incremental:

- Top Down (using stubs)
- Bottom up (using drivers)
- Hybrid

Big Bang testing

Functional & non-functional tests

Alpha Testing

- Pre-release of product tested by end user on **developer site**

Beta Testing

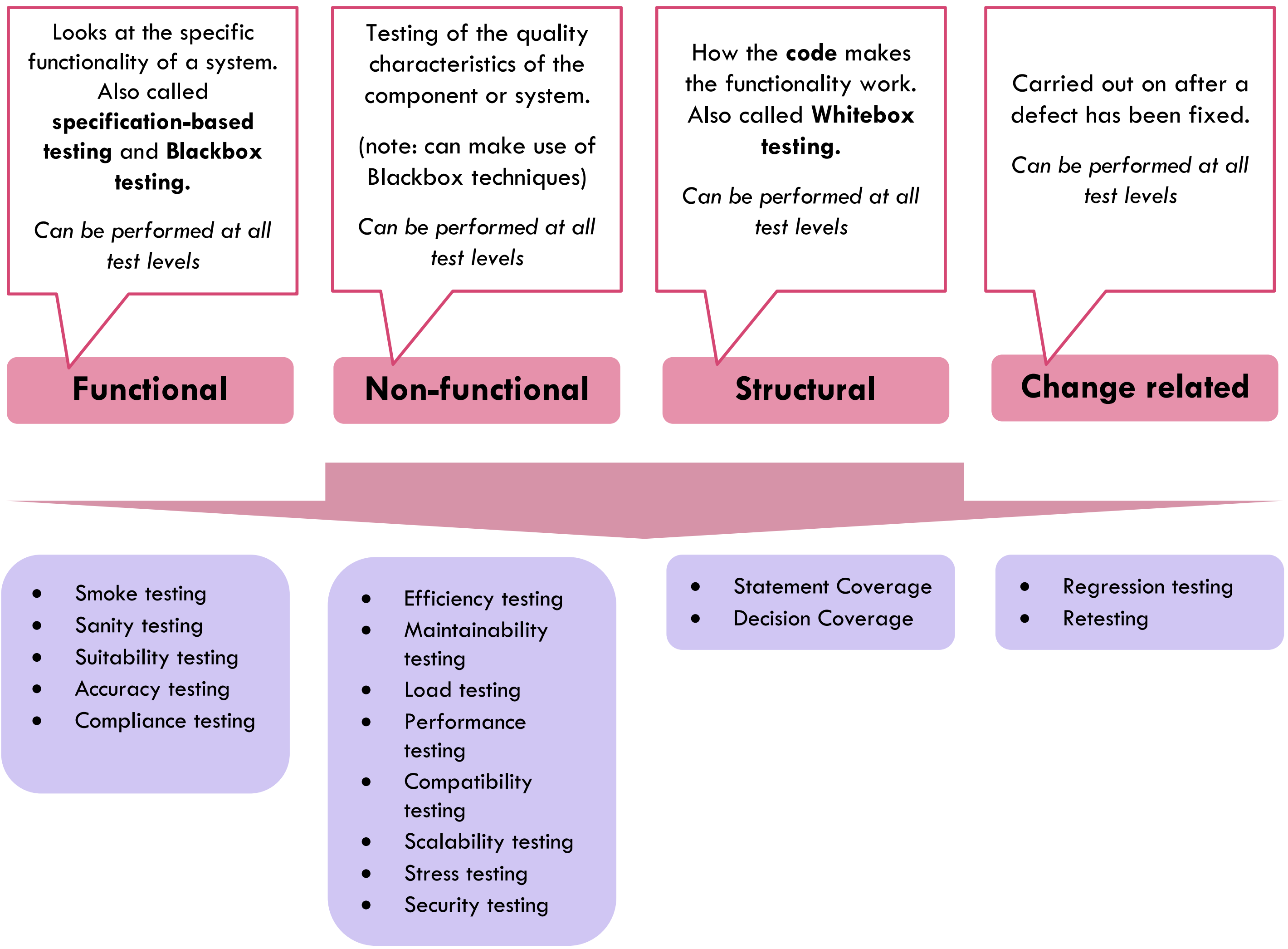
- Pre-release of product tested by end user on **user site**

Contract & Regulation acceptance testing

Operational acceptance Testing

Test Level	**Test Basis**	**Typical Test Objects**	**Typical Defect's and Failures**
Unit	• Code • Data Models • Detailed Design • Component Specification	• Components, units or modules • Code and data structures • Classes • Database modules	• Incorrect functionality (e.g., not as described in design specifications) • Data flow problems • Incorrect code and logic
Integration	• Software and system design • Sequence diagrams • Interface and communication protocol specifications • Use cases • Architecture at component or system level • Workflows • External interface definitions	• Subsystems • Databases • Infrastructure • Interfaces • APIs • Microservices	• Inconsistent message structures between systems • Incorrect data, missing data, or incorrect data encoding • Interface mismatch • Failures in communication between systems • Unhandled or improperly handled communication failures between systems
System	• System and software requirement specifications (functional and non-functional) • Risk analysis reports • Use cases • Epics and user stories • Models of system behaviour • State diagrams • System and user manuals	• Applications • Hardware/software systems • Operating systems • System under test (SUT) • System configuration and configuration data	• Incorrect calculations • Incorrect or unexpected system functional or non-functional behaviour • Incorrect control and/or data flows within the system • Failure to properly and completely carry out end-to-end functional tasks • Failure of the system to work properly in the production environment(s) • Failure of the system to work as described in system and user manuals
Acceptance	• Business processes • User or business requirements • Regulations, legal contracts and standards • Use cases • System requirements • System or user documentation • Installation procedures • Risk analysis reports	• System under test • System configuration and configuration data • Business processes for a fully integrated system • Recovery systems and hot sites (for business continuity and disaster recovery testing) • Operational and maintenance processes • Forms • Reports • Existing and converted production data	• System workflows do not meet business or user requirements • Business rules are not implemented correctly • System does not satisfy contractual or regulatory requirements • Non-functional failures such as security vulnerabilities, inadequate performance efficiency under • high loads, or improper operation on a supported platform

D. TEST TYPES:



Looks at the specific functionality of a system. Also called **specification-based testing** and **Blackbox testing.**

Can be performed at all test levels

Functional

- Smoke testing
- Sanity testing
- Suitability testing
- Accuracy testing
- Compliance testing

Testing of the quality characteristics of the component or system.

(note: can make use of Blackbox techniques)

Can be performed at all test levels

Non-functional

- Efficiency testing
- Maintainability testing
- Load testing
- Performance testing
- Compatibility testing
- Scalability testing
- Stress testing
- Security testing

How the **code** makes the functionality work. Also called **Whitebox testing.**

Can be performed at all test levels

Structural

- Statement Coverage
- Decision Coverage

Carried out on after a defect has been fixed.

Can be performed at all test levels

Change related

- Regression testing
- Retesting

E. CHANGE RELATED TESTING

i. Confirmation Testing (AKA Retesting)

After a defect has been fixed, the software should be retested to confirm original defect has been removed.

ii. Regression testing

- Carried out on every other part of the system to check that a fixed defect hasn't changed other parts of the system
- Repeated testing of already tested program
- Performed when software or environment is changed
- Based on risk
- Regression testing is used in agile and is automated (see below).
- Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation. Automation of these tests should start early in the project

iii. What is Change Related Maintenance testing?

- It is testing that is done on a system in a **live environment** when software undergone:
  - *Modification*
  - *Migration*
  - *Retirement*
  - *Hot fixes*
- It is very high risk
- Impact analysis (**Risk**) and Metrics from previous projects are very important in this area
  - They help estimate the amount of re-testing and regression testing
  - What are the possible consequences?
  - What areas will remain unchanged