Testing Principles & Quality Process

Principles
↳ Methods Techniques
↳ Methodologies
↳ Tools.

Principles of Testing & Analysis:

1. What: The quality process.
2. Where: Focusing on problematic constructs and modules most likely to contain bugs.
3. When: Performing testing as early & often as possible.
4. Who: Structuring organization for effective testing.
5. How: Strategies for effective testing.

- WHAT : The Quality Process :

The quality process provides a framework for —

1) Selecting & arranging activities.
2) Considering interactions & trade-offs

with other important goals.

Quality Goals:

→ Process qualities:
- ↳ repeatability, timeliness, cost, ...
- ↳ continuous improvement

→ Product qualities
- ↳ internal qualities
  - ↳ reusability, manageability, maintainibility, modifiability.
- ↳ external qualities
  - ↳ Dependability qualities
    availability, correctness, reliability, safety
  - ↳ Useful qualities
    usability, performance, security, portability

Programmers MAKE MISTAKES!!

Reasons:

> Time pressure
> Complex existing code

> Complex infrastructure
> Misunderstanding of requirements
> Incorrect requirements

Average of 15–50 bugs per 1000 SLOC

○ WHERE : Error prone Aspects:

> Floating-point numbers :

↳ Inherently imprecise. The imprecision may lead to invalid comparisons

↳ Sequences of computations may lose precision.

> Pointers (C/C++) :

↳ Pointers referring to the wrong memory areas can corrupt data.

↳ Aliasing can make programs difficult to understand & change.

eg. using of malloc() and then freeing the memory once utilized

> Parallelism :

↳ Can result in subtle timing errors because of unforeseen interaction between parallel processes.

↳ Can result in deadlock if synchronization is used incorrectly.

> Numeric Limits / Boundaries :

↳ Very large values for integers / floats

↳ Boundary values for relational expressions.

> Interrupts : → for embedded softwares

↳ Interrupts can cause a critical operation to be terminated and make a program difficult to understand.

↳ Compatable to go-to statements.

> Complex boolean expressions (eg. nested boolean operators)

> Casts & conversions b/w types
(eg. numeric types, can lose precision or overflow)

80-20 rule for module testing

A small number of modules usually contain most of the defects discovered during pre-release testing.

i.e → 20% of the modules contain 80% of the bugs.

- **HOW! : Strategies for effective testing.**

1. Divisibility → Divide & Conquer
   - Scope of test. → different strategies to be used.
   - Purpose of test. → strategising based on purpose
   - Testing techniques. → overall techniques used.

2. Visibility
   - Observability.
   - State exposure.
   - Logging.

3. Repeatability

- Tests that sometimes fail are called "flakey tests".

(could be due to software tests)
(could be bad test, program/environment)

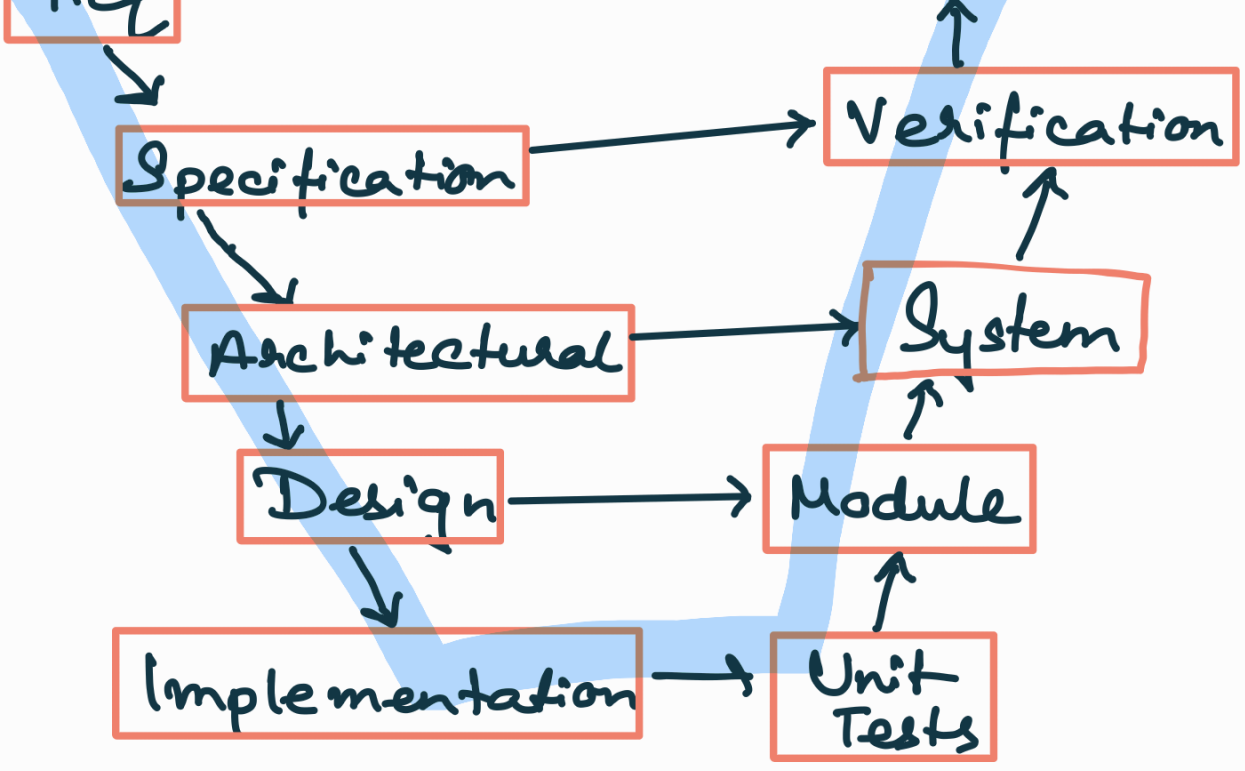
4. Redundancy

- Most verification methods are unsound: they miss errors

5. Feedback

- Different applications have different "pain points".
- Update tests to more thoroughly test areas known to be problematic.
- Learn which classes of bugs are most likely.
- Work with developers to reduce systematic errors.

- **WHEN! : The V-Model**



Recap of the V model:

> Model of software development which pairs phases.

> Each phase on the left side of the V generates test planning for an accompanying phase on the right.

> While criticism is justified, the V model has a place in any modern software dev. process.

(eg. Agile uses multiple mini V models in its system)

∴ The V-model is a software development model that pairs different stages of software development with the appropriate

testing procedures.
These tests are later used when checking the verification of each phase of the software.

Validation & Verification in V-Model

Validation:

- Confirm that the software performs to the user's satisfaction.
- Assuring that the software system meets the user's needs
- Answering — 'Are we building the right product ?'

Verification:

- Confirm that the software performs and conforms to its specification.
- Answering — 'Are we building the product right ?'
