ELSEWEDY
UNIVERSITY
OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

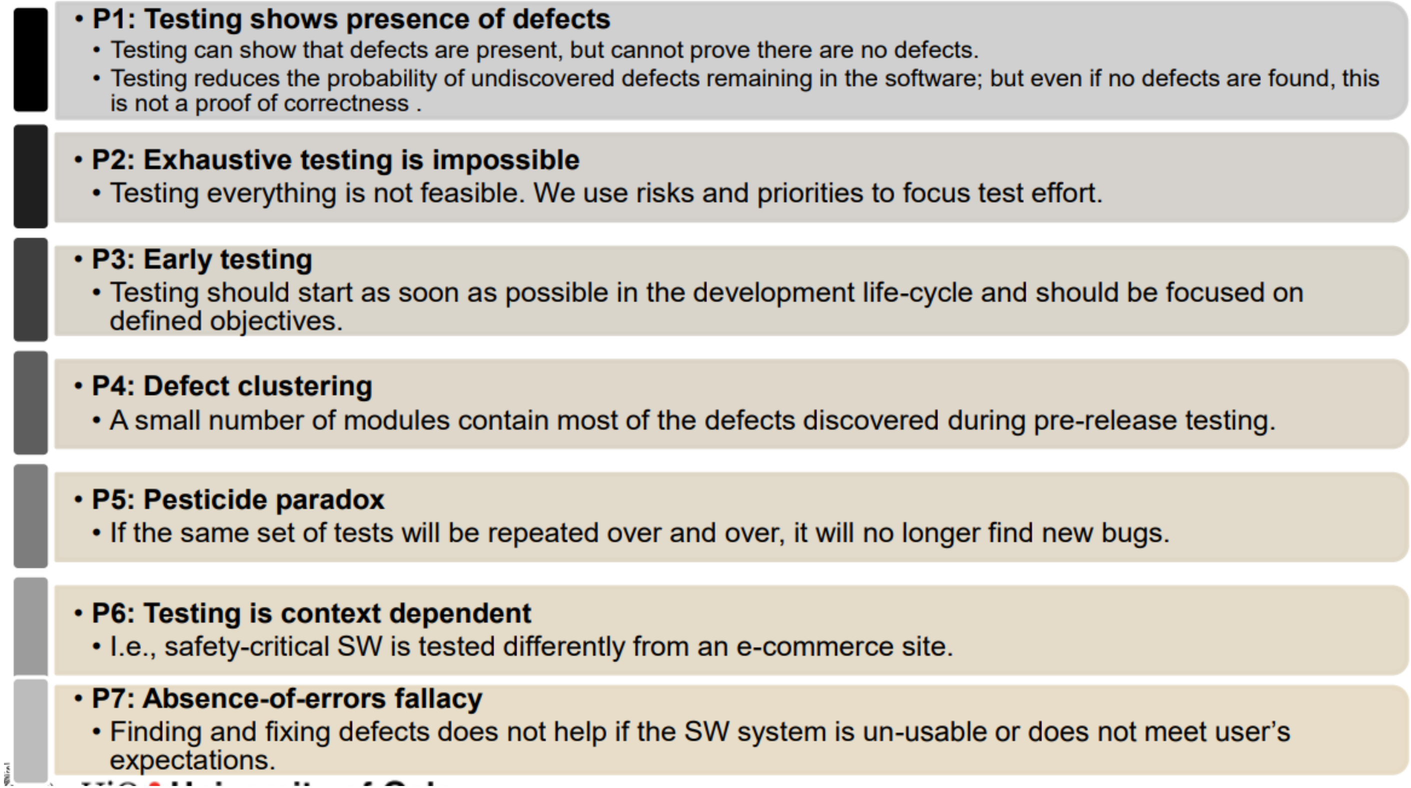
Software Testing and QA

Lecture 3-4

© Spring 2025 - **DR. Hesham Sakr**
Building, office No. 12
Hesham.sakr@sut.edu.eg

The seven test principles

The test principles



- **P1: Testing shows presence of defects**
  - Testing can show that defects are present, but cannot prove there are no defects.
  - Testing reduces the probability of undiscovered defects remaining in the software; but even if no defects are found, this is not a proof of correctness .
- **P2: Exhaustive testing is impossible**
  - Testing everything is not feasible. We use risks and priorities to focus test effort.
- **P3: Early testing**
  - Testing should start as soon as possible in the development life-cycle and should be focused on defined objectives.
- **P4: Defect clustering**
  - A small number of modules contain most of the defects discovered during pre-release testing.
- **P5: Pesticide paradox**
  - If the same set of tests will be repeated over and over, it will no longer find new bugs.
- **P6: Testing is context dependent**
  - I.e., safety-critical SW is tested differently from an e-commerce site.
- **P7: Absence-of-errors fallacy**
  - Finding and fixing defects does not help if the SW system is un-usable or does not meet user’s expectations.

P1: Testing shows the presence of defects, not their absence

- Testing can show that defects are present, but cannot prove that there are no defects.

- Testing reduces the probability of undiscovered defects remaining in the software. However, even if no defects are found, this is not a proof of correctness.

P2: Exhaustive testing is impossible

- Testing everything(all combinations of input and preconditions) is not feasible except for specific cases.

- We use risks and priorities to focus the testing.

P3: Early testing saves time and money

- Testing activities should start as early as possible in the software or system development life cycle and should be focused on defined objectives.

P4: Defect clustering defects cluster together

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

P5: Pesticide paradox

- If the same tests are repeated over and over again, the same set of test cases will no longer find any new bugs.

- To overcome this 'pesticide paradox', the test cases need to be regularly reviewed and revised, and new and different tests need to be written to investigate different parts of the software.

P6: Testing is context dependent

- Testing is done differently in different contexts.

- For example, testing of safety-critical software is different from e-commerce site testing.

P7: Absence of error fallacy

- Finding and fixing defects does not help if the software system does not fulfill users' needs and expectations .

Fundamental Test Processes

Fundamental test processes

- Test planning
- Test monitoring and control
- Test analysis
- Test design
- Test implementation
- Test execution
- Test completion

Test planning , monitoring and control

- **Who, what , why, when and where**
- A plan encompasses: what, how, when, by whom?
  - Scope, objectives and risk analyses
  - Test levels and types that will be applied
  - Documentation that will be produced
  - Assign resources for the different test activities
  - Schedule test implementation, execution, evaluation
- Control and adjust the planning to reflect new information, new challenges of the project.

Analysis and design

- Review test basis:
  - Requirements
  - Product architecture
  - Product design
  - Interfaces
  - Risk analysis report
- Analysis: general test objectives are transformed into:
  - Test conditions
- Design:
  - Test cases
  - Test environments
  - Test data
  - Create traceability

Implementation and execution

- *Implement*:
    - Group tests into scripts
    - Prioritize the scripts
    - Prepare test oracles
    - Write automated test scenarios
- *Execute*:
    - Run the tests and compare results with oracles
    - Report incidents
    - Repeat test activities for each corrected discrepancy
    - Log the outcome of the test execution

Test completion

- *Evaluate*:
  - Assess test execution against the defined objectives
  - Check if:
    - More tests are needed
    - Exit criteria should be changed
- *Report*:
  - Write or extract a test summary report for the stakeholders.
- *Test closure activities*
  - The activities that make the test assets available for later use.

The psychology of testing

A good tester needs:

- Attention to details
- Good communication skills
- Experience at error guessing

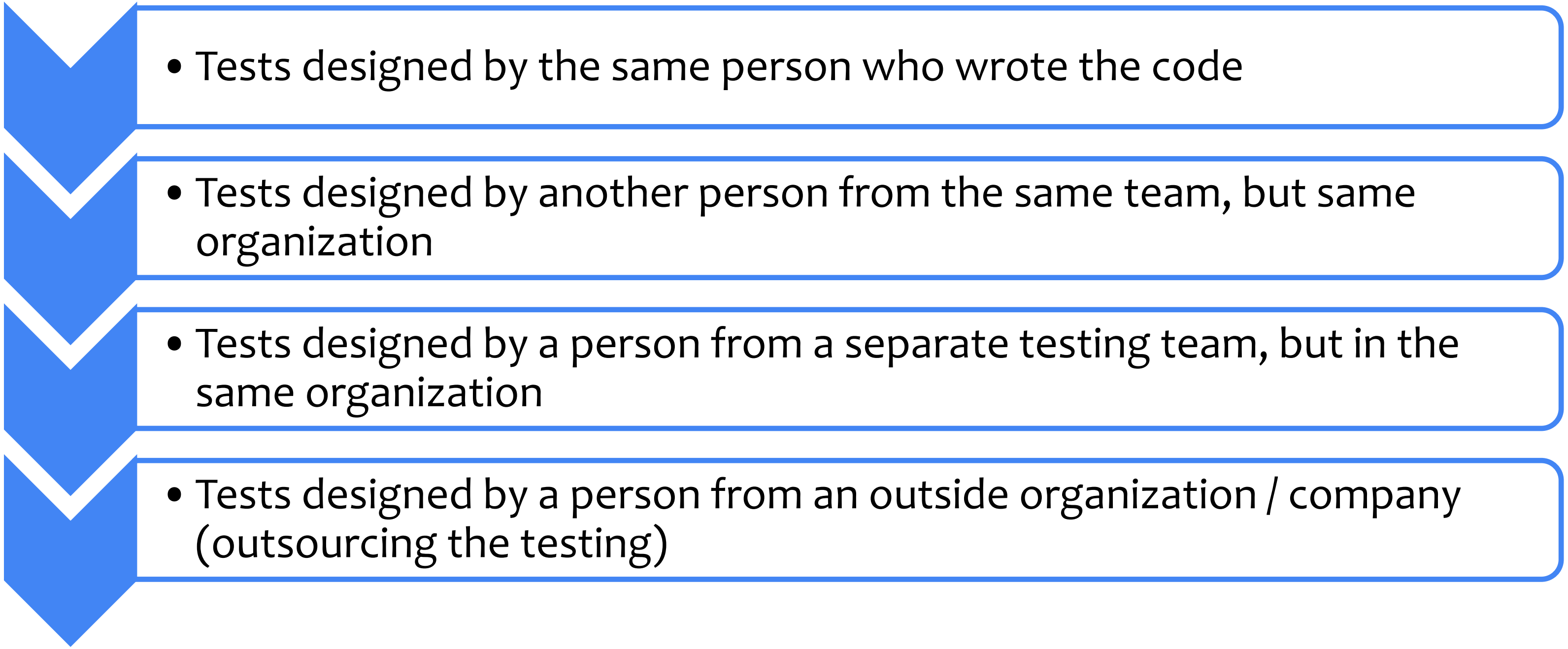
- To communicate defects and failures in a constructive way:
    - fact-focused reports and review of findings

Independence in testing

- A certain degree of independence is often more effective at finding defects and failures.
    - However, the developer can very efficiently find bugs in their own code.
- The level of independence in the testing depends on
    - the objective of testing.

Independence test levels

- Independence levels:



- Tests designed by the same person who wrote the code
- Tests designed by another person from the same team, but same organization
- Tests designed by a person from a separate testing team, but in the same organization
- Tests designed by a person from an outside organization / company (outsourcing the testing)

Tips and tricks

- Be clear and objective

- Confirm that:
  - You have understood the requirements
  - The person that has to fix the bug has understood the problem

ELSEWEDY
UNIVERSITY
OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Software Quality

Factors in Project Success & Failure

Software Reliability

Metrics of Software Quality – Performance & Scalability

- Performance
  - The ability to complete requested functions or services within the expected time span by the users.
  - e.g., average response time for a given task
- Scalability
  - The capacity of a system to handle increasing load or demand.
  - e.g., # of concurrent users, # of transactions per second, # of requests per second

Product Quality Metrics

- Two key metrics for intrinsic product quality are Mean Time To Failure (**MTTF**) and availability
- **MTTF** is most often used with safety critical systems such as air traffic control systems, avionics, and weapons
- **Availability** is the probability that a system will work as required when required during the period of a mission.
- Both are correlated, but different in the same way that failures and defects are different

Metrics of Software Quality – Mean Time Between Failures

- Mean time between failures (MTBF)
  - Average of intervals between consecutive failures.
- Mean time to failures (MTTF)
  - Average amount of time a system operates before it fails
- Mean time to repair (MTTR)
  - Average time to repair/restart the system and get it back to running
- MTBF is a simple measure of reliability

MTBF = MTTF + MTTR

Metrics of Software Quality – Availability & Reliability

- Availability
  - The probability of a system to be available.
  - The fraction of time the system is available.

$$\frac{\text{available time (“up time”)}}{\text{total time}}$$

- Reliability
  - The probability of a system to operate without failures.
  - The fraction of all attempted operations that complete *successfully.*

$$\frac{\text{\# of successful operations}}{\text{\# of total operations attempted}}$$

Software Availability

- Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

  Availability = [MTTF/(MTTF + MTTR)] x 100%

- Consider 5 nines availability (99.999%); what does this mean?
  - 5 minutes of down time per year

[See Availability (system) – https://en.wikipedia.org/wiki/Availability_(system)]

Metrics of Software Quality – Error Rate & Completion Rate

- Reliability depends on the *unit* of operation
  - An operation may consists of multiple steps
  - Reliability ≠ Completion rate
- Error rate (per page)
  - The fraction of pages (unit of operation) that time out or fail
- Completion rate
  - The fraction of all attempted operations that *eventually* complete the operation
  - Completion ≠ Success

Integration & System Testing

- Integration testing
  - To expose defects in the interfaces and the interactions between integrated sub-systems.
- System ("end-to-end") testing
  - Test of an integrated system to determine whether it meets the specification.

Acceptance & Beta Testing

- Acceptance testing
  - To determine whether or not a system satisfies the user needs and requirements.
  - To enable the user, customers, or other authorized entity to determine whether or not to accept the system.
- Beta testing
  - One form of acceptance testing
  - Performed by *real* users in their own environment
  - Perform actual tasks without interference.

The Spectrum of Software Quality

نطاق

Software System Qualities

- Correctness
- Availability
- Reliability
- Performance
- Scalability
- Efficiency
- Safety
- Usability
- Security
- Robustness
- Maintainability
- Reusability
- Portability
- Interoperability

On Expected Behavior – Correctness vs. Reliability

- Correctness
  - Whether a system is consistent with its *specification*.
- Reliability
  - احتمال
  - The probability of a system to operate without failures.
  - Relative to its *specification* and a *usage profile*.
  - Statistical approximation to correctness

    100% reliable ≈ correct

On Exceptional Behavior – Safety vs. Robustness

- Safety
  - The ability of a software system to prevent certain undesirable behaviors, i.e., *hazards*.
- Robustness
  - The ability of a software system to *fail or degrade gracefully* outside its normal operating parameters.
  - Acceptable (degraded) behavior under extreme conditions.

The software's ability to withstand problems that may affect it.

Performance Related Qualities

- Performance
  - The ability to complete requested functions or services within the expected time span by the users.
- Scalability
  - The capacity of a system to handle increasing load or demand.
- Efficiency
  - The ability to make maximum and efficient use of system resources.

Usability & Security

- Usability
  - The ability for the users to use all the features of the system without special efforts.
- Security
  - The ability to maintain integrity of the system operation and the data.

Internal Qualities

- Maintainability
  - The ability to make changes, enhance, adapt, and evolve a software system over a long period of time.
- Reusability
  - The ability to use parts of the system in different project without special effort on the part of the developers
- Portability
  - The ability to port a software system to a different platform or operating environment

Software Quality

Conformance to customers’ requirements

Quality

- For software, two kinds of quality may be encountered:
    - **Quality of design** encompasses requirements, specifications, and the design of the system.
    - **Quality of conformance** is an issue focused primarily on implementation.
    - user satisfaction = compliant product + good quality + delivery within budget and schedule

Cost of Quality

- Prevention costs include
  - Quality planning
  - Formal technical reviews
  - Test equipment
  - Training
- Internal failure costs include
  - Rework
  - Repair
  - Failure mode analysis
- External failure costs are
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work

Customers' Expectations

- What's wrong with "performance to customers' expectations" rather than requirements?
- Often hear people say "We must exceed the customers' expectations!"
- What's the basic problem with this?
- The result is?

Application to Software

- Simplistically, software product quality is lack of “bugs” in the product
- Why is this problematical for software systems?
  - Correct operation is not sufficient – performance?
  - Usability by the end-user
  - Software specifications

Software Verification and Validation (V&V)

Verification and Validation

- **Verification**

  Does the software system meet the requirements specifications?

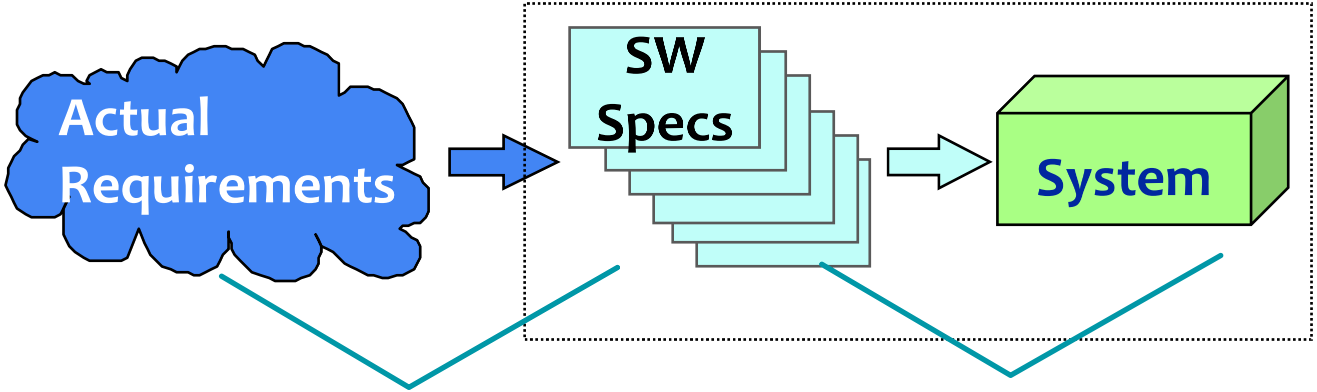
  Are we building the software right?

- **Validation**

  Does the software system meet the user's real needs?

  Are we building the right software?

Validation vs. Verification



Validation

Includes

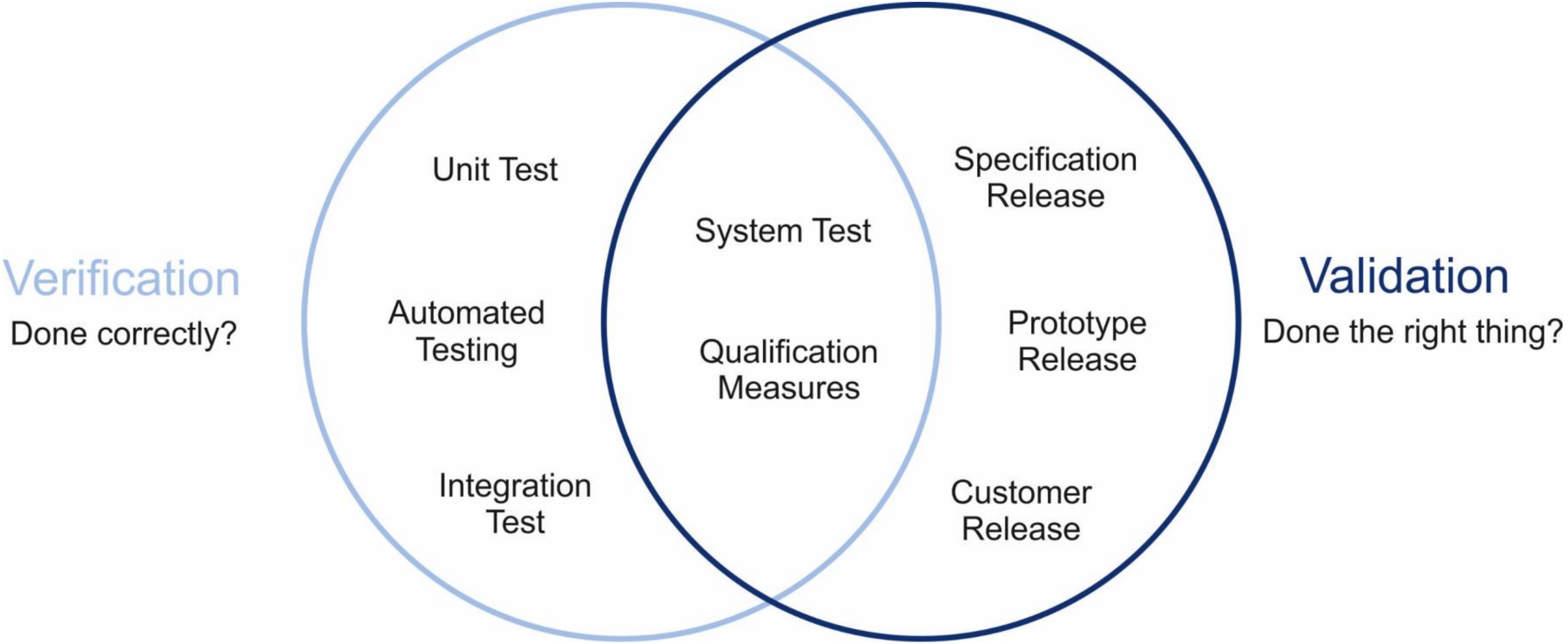
- usability testing
- user feedback

Verification

Includes

- testing (mostly)
- inspections
- static analysis

Validation vs. Verification



Software Testing in V&V

- Testing can be done for verification and validation
- Verification:

  To find defects by executing a program in a test or simulated environment

  - e.g., functional test, integration test
- Validation:

  To find defects by executing a program in a real environment or with real users

  - e.g., usability test, beta test

Software Testing in Development Life Cycle

Software Qualities and Process

- Qualities cannot be added after development
  - Quality results from a set of inter-dependent activities
  - Analysis and testing are crucial but far from sufficient.
- Testing is not a phase, but a lifestyle
  - Testing and analysis activities occur from early in requirements engineering through delivery and subsequent evolution.
  - Quality depends on every part of the software process
- An essential feature of software processes is that software test and analysis is thoroughly integrated and not an afterthought

The Quality Process

- Quality process: set of activities and responsibilities
  - focused primarily on ensuring adequate dependability
  - concerned with project schedule or with product usability
- The quality process provides a framework for
  - selecting and arranging activities
  - considering interactions and trade-offs with other important goals.

Testing Activities in Life Cycle

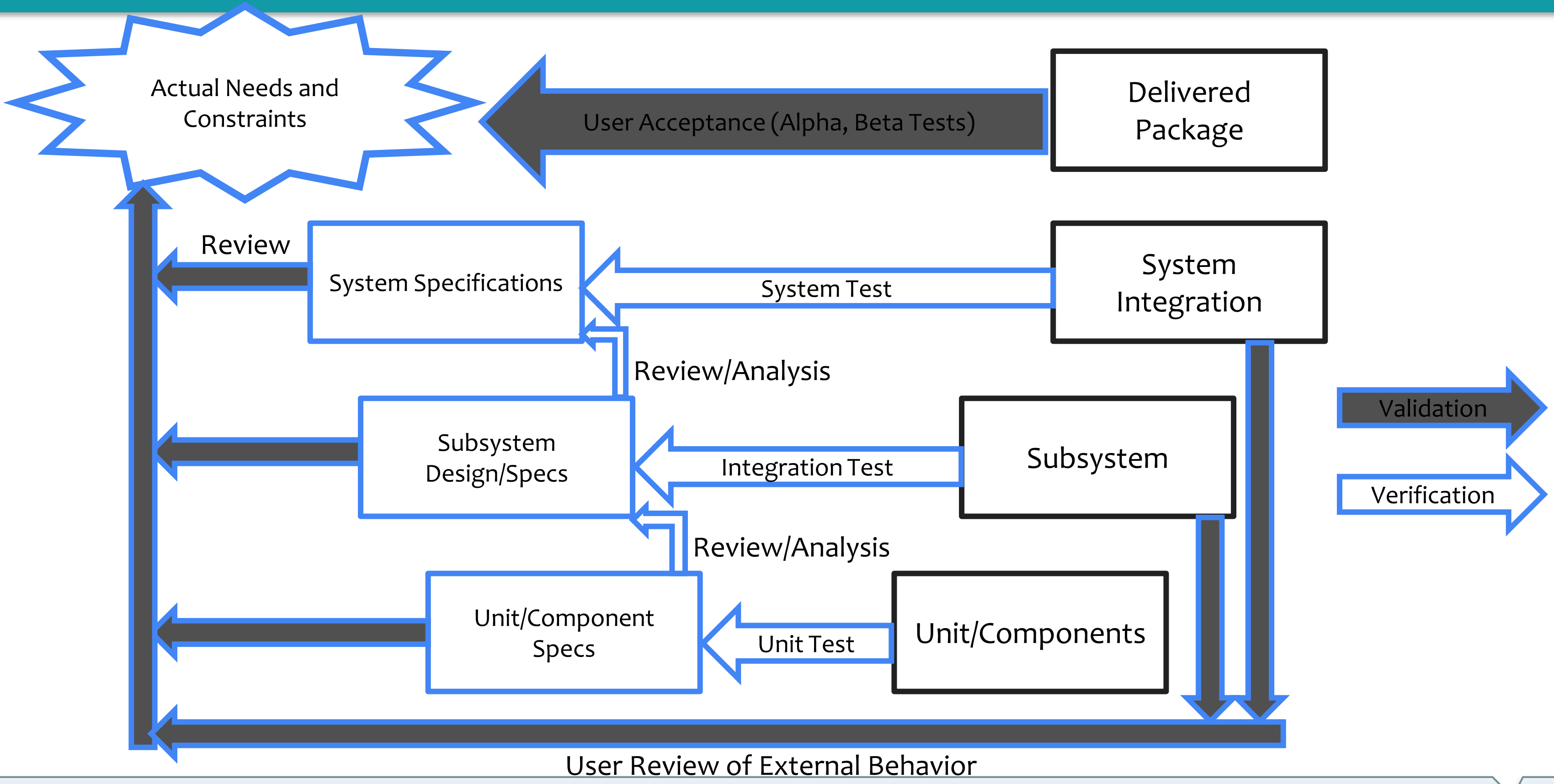
- For every development activity there is a corresponding testing activity
  - Development phases, development levels
- Each test level has objectives specific to that level
- Test design should start as early as possible
  - as soon as relevant documents are available

- Applicable to waterfall and agile development model

Levels of Granularity of Testing

- Unit (component, module) testing
- Integration testing
- System testing
- Acceptance testing

The V-Model of – Validation & Verification



Unit Testing

- Testing of individual software unit/module/components
  - Synonymous to *module testing*, *component testing*
- Focus on the functions of the unit
  - functionality, correctness, accuracy
- Usually carried out by the developers of the unit
- Basis for unit testing
  - component specifications
  - detailed design and code

Integration Testing

- Testing performed to expose defects in the *interfaces* and in the *interactions between* integrated components or sub-systems.
- Focus on the interactions between modules
- Usually carried out by the developers of the sub-systems involved

- Basis for integration testing
  - system design and architecture
  - subsystem and interface specification

Regression Testing

- Used when a large amount of testing is needed and the changes, while small, can affect many parts of the system.
- Best example is in compiler development:
    - Collect selected examples of code that exercise each part of the compiler
    - Add new examples when a bug is detected
    - Run the compiler over the entire collection and capture the output
    - After any change of the code within the compiler, repeat the run
    - Compare with the baseline results

System Testing

- Testing of an integrated system to verify that it meets the specification.
  - A.k.a. the *end-to-end test*
- Verify functional and *non-functional* requirements
- Carried out by the developers and *independent testers*
- Basis for system testing
  - software requirement *specification*
  - functional *specification*

Acceptance Testing

- Test the whole system to ensure that it meets the *requirements*
- Focus on customer acceptance
- Carried out by *independent testers* and the *customers*
- Basis for acceptance testing
  - system and user requirements
  - use cases, business processes, risk analysis

Acceptance Testing & Criteria

- Acceptance testing

  Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the *acceptance criteria* and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

- Acceptance criteria

  The *exit criteria* that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.

Acceptance Testing Techniques

- Random (statistical) testing
- Alpha testing
- Beta testing

Acceptance Testing – Random Test

- Random test (statistical test)
  - Test cases are selected randomly, possibly using a pseudo-random number generation algorithm, to match an *operation profile,* or *usage profile.*
- Not the same as *ad hoc* testing

Acceptance Testing – Alpha Test

- Simulated operational testing.
- Performed by personnel *acting as* potential users/customers.
- Carried out in a *controlled* environment.
- Observed by the development organization.

Acceptance Testing – Beta Test

- Operational testing to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes.
- Performed by *real* users in their own environment.
- Perform actual tasks without interference or close monitoring

Summary: Key Concepts

- Spectrum of software qualities
- Metrics of quality attributes
- Cost of software defects
- V-model of validation and verification
- Levels of granularity of testing
    - Unit, integration, system, acceptance test
    - Regression test