# Software Project Management
## Sixth Edition

# Chapter 13.1

## Software product quality

## The importance of software quality

- Increasing criticality of software
- The intangibility of software
- Project control concerns:
  - errors accumulate with each stage
  - errors become more expensive to remove the later they are found
  - it is difficult to control the error removal process (e.g. testing)

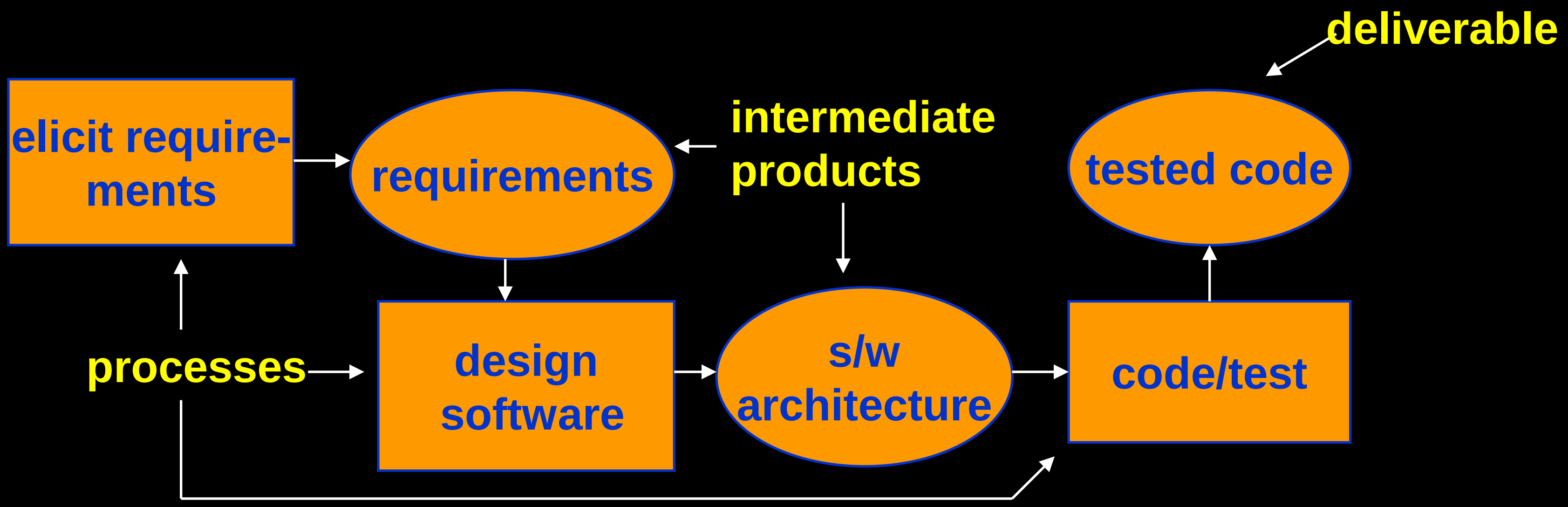
## Quality specifications

Where there is a specific need for a quality, produce a quality specification

- Definition/description of the quality
- Scale: the unit of measurement
- Test: practical test of extent of quality
- Minimally acceptable: lowest acceptable value, if compensated for by higher quality level elsewhere
- Target range: desirable value
- Now: value that currently applies

# ISO standards: development life cycles

A development life cycle (like ISO 12207) indicates the sequence of *processes* that will produce the software *deliverable* and the *intermediate products* that will pass between the processes.



## ISO standards

ISO 9126 Software product quality

Attributes of software product quality

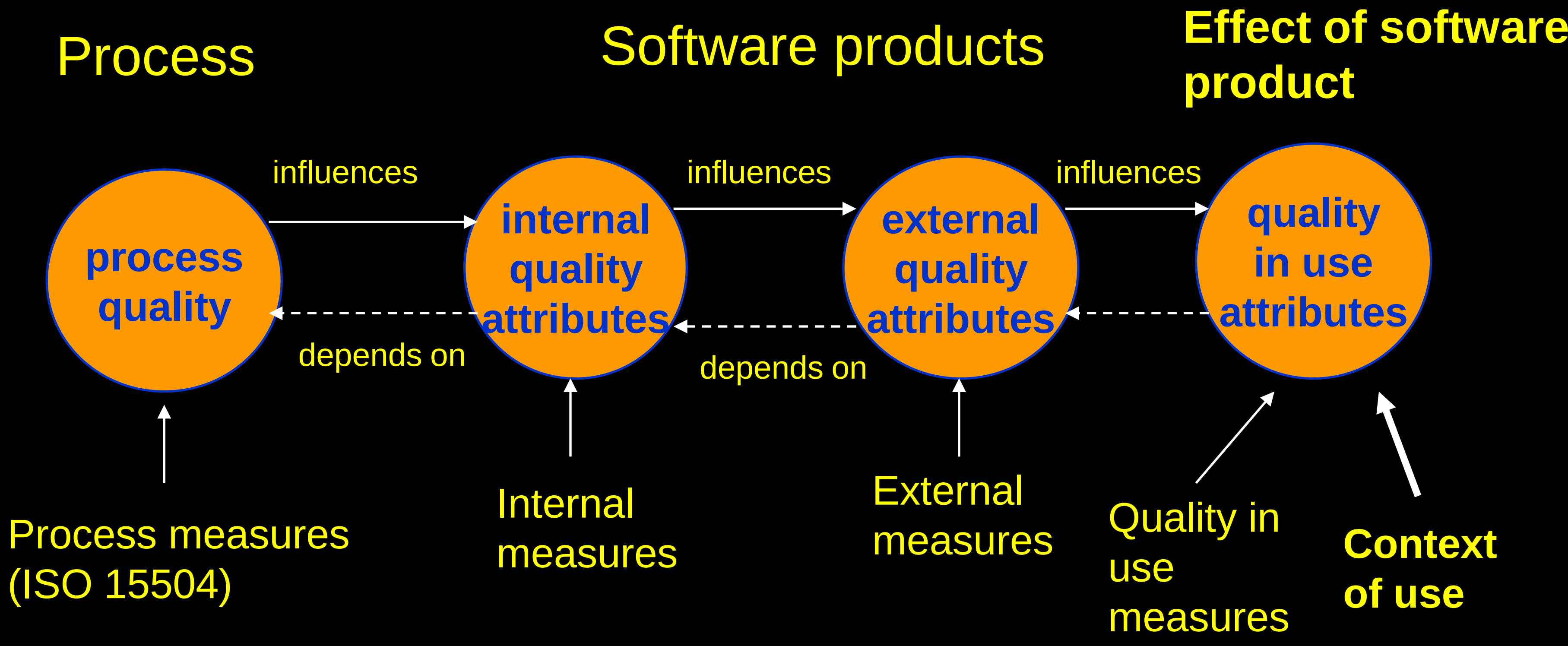
- External qualities i.e. apparent to the user of the deliverable
- Internal qualities i.e. apparent to the developers of the deliverables and the intermediate products

ISO 14598 Procedures to carry out the assessment of the product qualities defined in ISO 9126

## Types of quality assessment

- During software development:
  - To assist developers to build software with the required qualities
- During software acquisition:
  - To allow a customer to compare and select the best quality product
- For a particular community of users:
  - Independent evaluation by assessors rating a software product

# ISO 9126 software product quality



## Quality in use

- Effectiveness – ability to achieve user goals with accuracy and completeness
- Productivity – avoids excessive use of resources in achieving user goals
- Safety – within reasonable levels of risk of harm to people, business, software, property, environment etc,
- Satisfaction – happy users!

‘users’ include those maintain software as well as those who operate it.

## Quality Models

- It is a model of the software:
  - Constructed with the objective to describe, assess and/or predict quality.
- Popular models:
  - Garvin's model
  - Boehm's model
  - ISO 9126
  - Dromey's Model

## ISO 9126 software qualities

| | |
|---|---|
| functionality | does it satisfy user needs? |
| reliability | can the software maintain its level of performance? |
| usability | how easy is it to use? |
| efficiency | relates to the physical resources used during execution |
| maintainability | relates to the effort needed to make changes to the software |
| portability | how easy can it be moved to a new environment? |

## Sub-characteristics of Functionality

- Suitability
- Accuracy
- Interoperability
  - ability of software to interact with other software components
- Functionality compliance
  - degree to which software adheres to application-related standards or legal requirements e.g audit
- Security
  - control of access to the system

## Sub-characteristics of Reliability

- Maturity
  - frequency of failure due to faults - the more the software has been used, the more faults will have been removed
- Fault-tolerance
- Recoverability
  - note that this is distinguished from 'security' - see above
- Reliability compliance

– complies with standards relating to reliability

## Sub-characteristics of Usability

- Understandability
  - easy to understand?
- Learnability
  - easy to learn?
- Operability
  - easy to use?
- Attractiveness – this is a recent addition
- Usability compliance
  - compliance with relevant standards

## Sub-characteristics of Efficiency

- Time behaviour
  - e.g. response time
- Resource utilization
  - e.g. memory usage
- Efficiency compliance
  - compliance with relevant standards

## Sub-characteristics of Maintainability

- "Analysability"
  - ease with which the cause of a failure can be found
- Changeability
  - how easy is software to change?
- Stability
  - low risk of modification having unexpected effects
- "Testability"
- Maintainability conformance

## Sub-characteristics of portability

- Adaptability
- “Installability”
- Co-existence
  - Capability of co-existing with other independent software products
- “Replaceability”
  - factors giving ‘upwards’ compatibility - ‘downwards’ compatibility is excluded
- Portability conformance
  - Adherence to standards that support portability

## Using ISO 9126 quality standards (development mode)

- Judge the importance of each quality factor for the application
  - for example, safety critical systems - *reliability* very important
  - real-time systems - *efficiency* important
- Select relevant external measurements within ISO 9126 framework for these qualities, for example
  - mean-time between failures for reliability
  - response-time for efficiency

## Using ISO 9126 quality standards

- Map measurement onto ratings scale to show degree of user satisfaction – for example response time

| **response (secs)** | **rating** |
|---|---|
| <2 | Exceeds requirement |
| 2-5 | Target range |
| 6-10 | Minimally acceptable |
| >10 | Unacceptable |

## Using ISO 9126 quality standards

- Identify the relevant internal measurements and the intermediate products in which they would appear
  - For example, at software design stage the estimated execution time for a transaction could be calculated

# Using ISO9126 approach for application software selection

- Rather than map engineering measurement to qualitative rating, map it to a score
- Rate the importance of each quality in the range 1-5
- Multiply quality and importance scores – see next slide

| Response (secs) | Quality score |
|---|---|
| <2 | 5 |
| 2-3 | 4 |
| 4-5 | 3 |
| 6-7 | 2 |
| 8-9 | 1 |
| >9 | 0 |

# Weighted quality scores

| | | Product A | | Product B | |
|---|---|---|---|---|---|
| Product quality | Importance rating (a) | Quality score (b) | Weighted score (a x b) | Quality score (c) | Weighted score (a x c) |
| usability | 3 | 1 | 3 | 3 | 9 |
| efficiency | 4 | 2 | 8 | 2 | 8 |
| maintain-ability | 2 | 3 | 6 | 1 | 2 |
| Overall totals | | | 17 | | 19 |

## How do we achieve product quality?

- The problem: quality attributes tend to *retrospectively* measurable
- Need to be able to examine processes by which product is created beforehand
- The production process is a network of sub-processes
  - Output from one process forms the input to the next
  - Errors can enter the process at any stage

## Correction of errors

- Errors are more expensive to correct at later stages
  - need to rework more stages
  - later stages are more detailed and less able to absorb change
- Barry Boehm
  - Error typically 10 times more expensive to correct at coding stage than at requirements stage
  - 100 times more expensive at maintenance stage

## For each activity, *define*:

- Entry requirements
  - these have to be in place before an activity can be started
  - example: ‘a comprehensive set of test data and expected results be prepared and independently reviewed against the system requirement before program testing can commence’

## For each activity, *define:*

- Implementation requirements
  - these define how the process is to be conducted
  - example ‘whenever an error is found and corrected, *all* test runs must be completed, including those previously successfully passed’

## For each activity, *define:*

- Exit requirements
  - an activity will not be completed until these requirements have been met
  - example: ‘the testing phase is finished only when all tests have been run in succession with no outstanding errors’

# Techniques to Enhance Product Quality

- Increasing visibility
  - ‘egoless programming’. Weinberg encouraged the simple practice of programmers looking at each other’s code.
- Procedural structure
  - Over the years there has been the growth of
  - Methodologies
- Checking intermediate stages
  - checking the correctness of work at various stages of development

## Testing

- Objective of testing is to expose as many bugs as possible.
- How is testing done?
  - Input test data to the program.
  - Observe the output: Check if the program behaves as expected

# 3 Testing Levels

- Software tested at 3 levels:
  - Unit testing
  - Integration testing
  - System testing

$\left.\begin{array}{l}\text{Unit testing}\\\text{Integration testing}\\\text{System testing}\end{array}\right\}$ **Regression testing**

## Test Levels

- Unit testing
  - Test each module (unit, or component) independently
  - Mostly done by developers of the modules
- Integration and system testing
  - Test the system as a whole
  - Often done by separate testing or QA team
- Acceptance testing
  - Validation of system functions by the customer

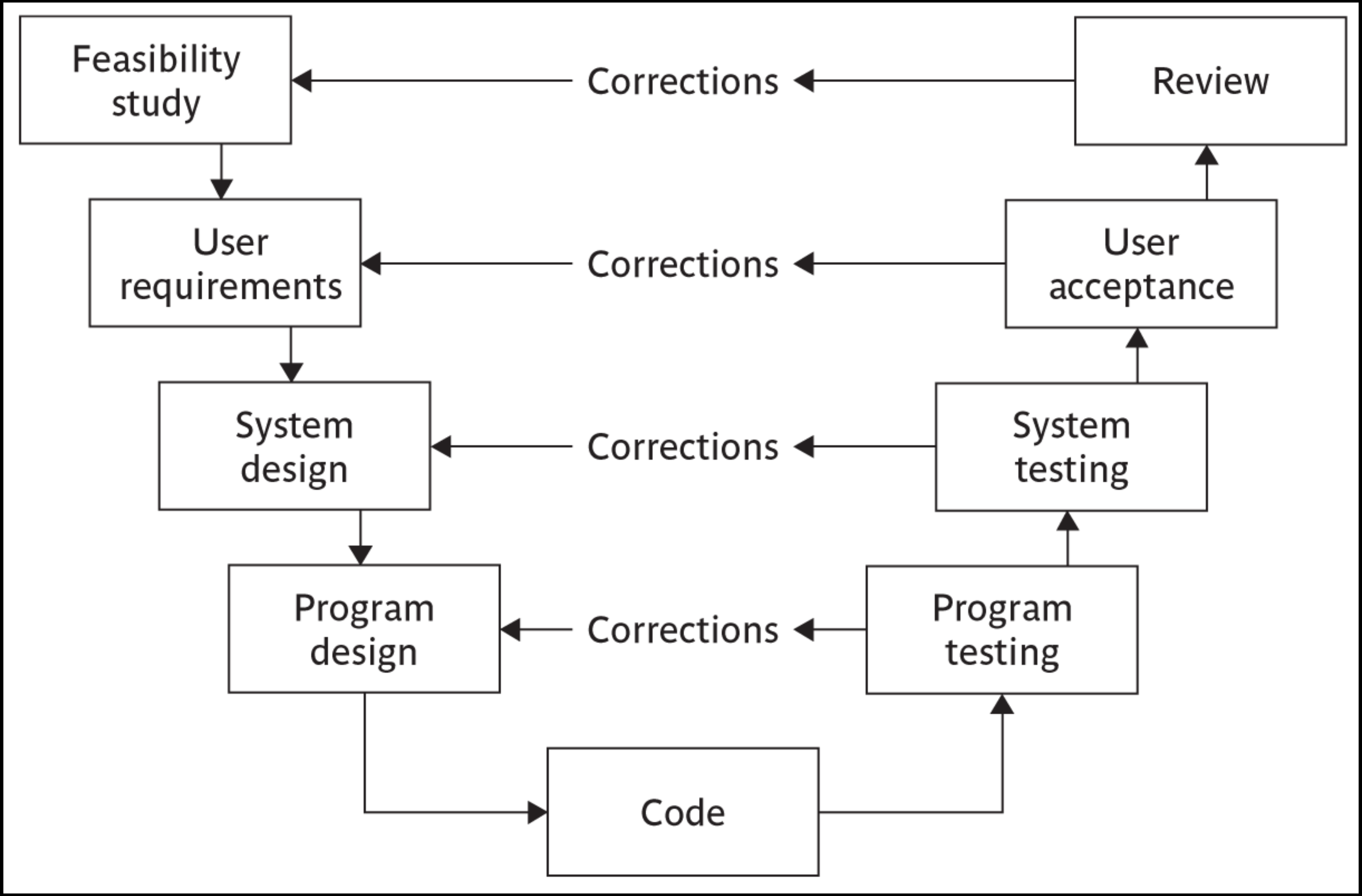
## Verification versus Validation

- Verification is the process of determining:
  - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
  - Whether a fully developed system conforms to its SRS document.
- Verification is concerned with phase containment of errors:
  - Whereas the aim of validation is that the final product be error free.

## Testing: the V-process model

- This shown diagrammatically in the next slide
- It is an extension of the waterfall approach
- For each development stage there is a testing stage
- The testing associated with different stages serves different purposes e.g. system testing tests that components work together correctly, user acceptance testing that users can use system to carry out their work

# Testing: the V-process model



## Black box versus glass box test

- **Glass box testing**
  - The tester is aware of the internal structure of the code; can test each path; can assess percentage test coverage of the tests e.g. proportion of code that has been executed
- **Black box testing**
  - The tester is not aware of internal structure; concerned with degree to which it meets user requirements

## Levels of testing

- Unit testing
- Integration testing
- System testing

## Testing activities

- *Test planning*
- *Test suite design*
- *Test case execution and result checking*
- *Test reporting:*
- *Debugging:*
- *Error correction:*
- *Defect retesting*
- *Regression testing*
- *Test closure:*

## Test plans

- Specify test environment
  - In many cases, especially with software that controls equipment, a special test system will need to be set up
- Usage profile
  - failures in operational system more likely in the more heavily used components
  - Faults in less used parts can lie hidden for a long time
  - Testing heavily used components more thoroughly tends to reduce number of operational failures

## Management of testing

The tester executes test cases and may as a result find discrepancies between actual results and expected results – **issues**

**Issue resolution –** could be:

- a mistake by tester
- a fault – needs correction
- a fault – may decide not to correct: **off-specification**
- a change – software works as specified, but specification wrong: submit to change control

## Decision to stop testing

The problem: impossible to know there are no more errors in code

Need to estimate how many errors are likely to be left

**Bug seeding –** insert (or leave) known bugs in code

Estimate of bugs left =

(total errors found)/(seeded errors found) x (total seeded errors)

## Alternative method of error estimation

- Have two independent testers, A and B
- $N_1$ = valid errors found by A
- $N_2$ = valid errors found by B
- $N_{12}$ = number of cases where same error found by A and B
- Estimate = $(N_1 \times N_2)/ N_{12}$
- Example: A finds 30 errors, B finds 20 errors. 15 are common to A and B. How many errors are there likely to be?

## Test automation

- Other than reducing human effort and time in this otherwise time and effort-intensive work,
  - Test automation also significantly improves the thoroughness of testing.
- A large number of tools are at present available both in the public domain as well as from commercial sources.

## Types of Testing Tools

- Capture and playback
- Automated test script
- Random input test
- Model-based test

## Software reliability

- The reliability of a software product essentially denotes its *trustworthiness* or *dependability*.
  - Usually keeps on improving with time during the testing and operational phases as defects are identified and repaired.
- A reliability growth model (RGM) models how the reliability of a software product improves as failures are reported and bugs are corrected.
  - An RGM can be used to determine when during the testing phase a given reliability level will be attained, so that testing can be stopped