Software Project Management
Fifth 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)

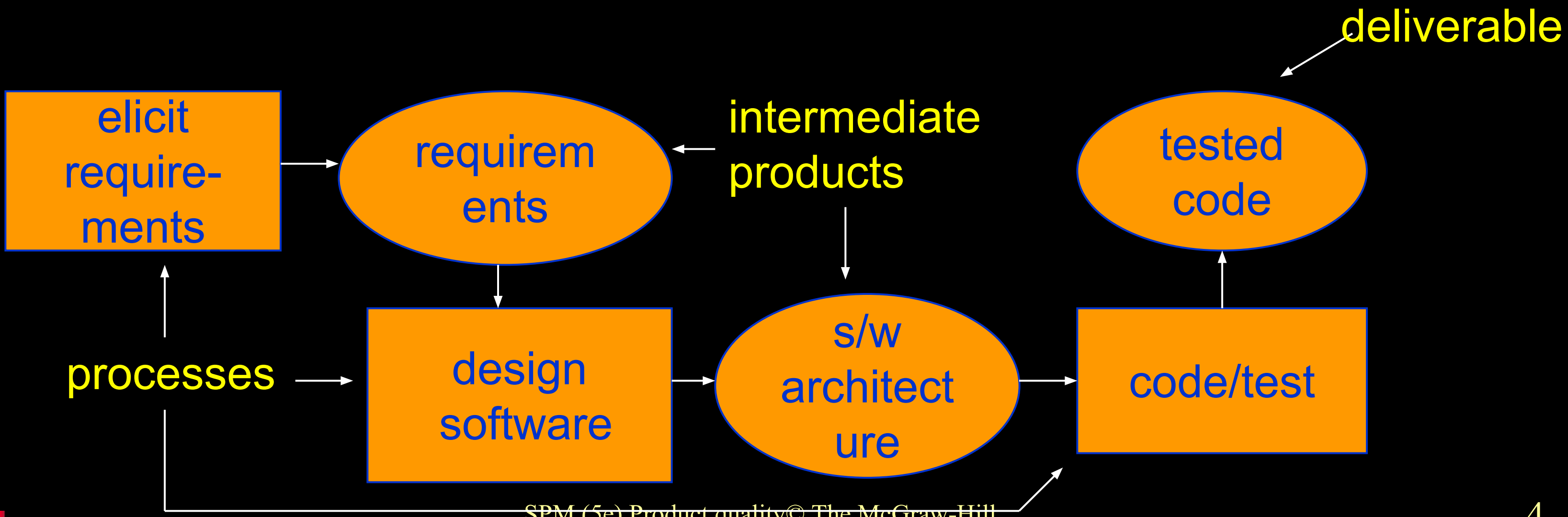
Quality specifications

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

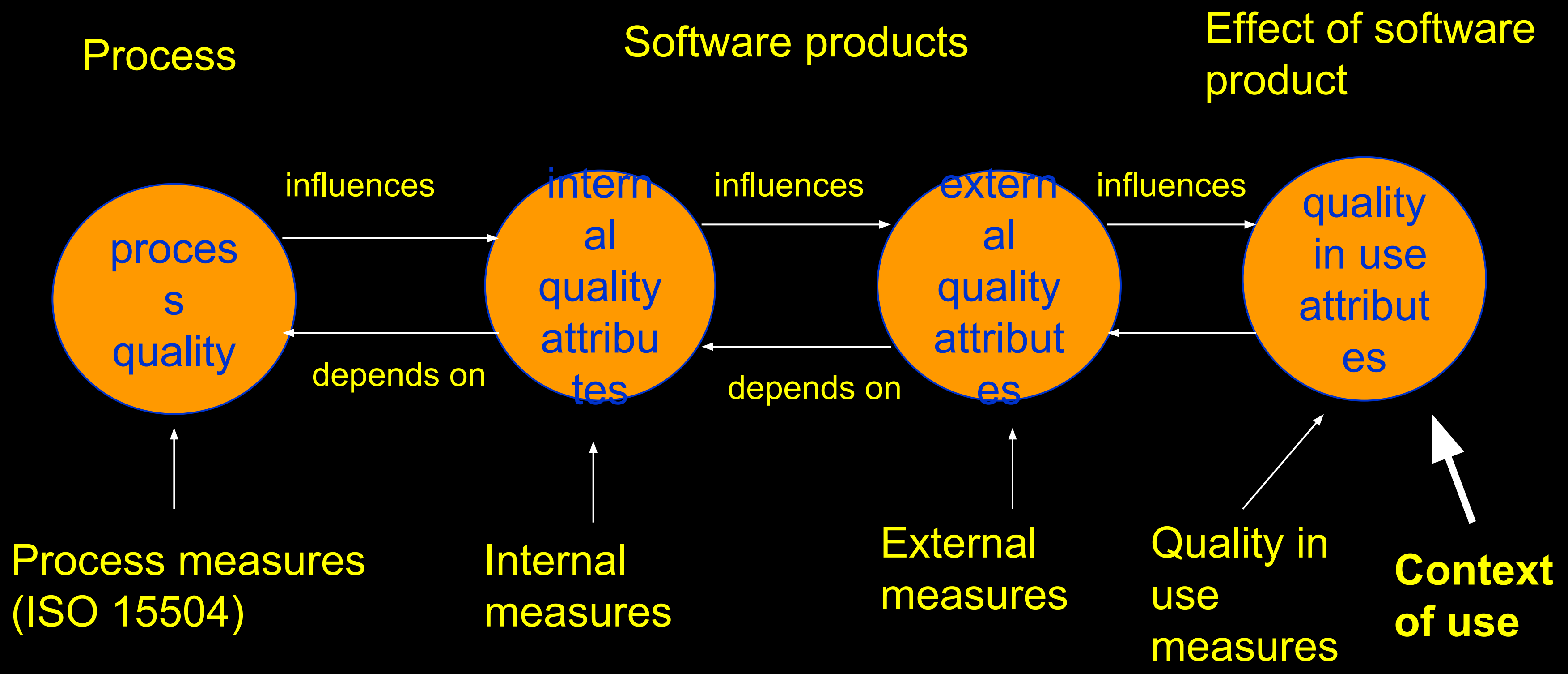
Types of quality assessment

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

Independent evaluation by assessors rating a software product for a particular community of users

ISO 9126 software product quality



Process

Software products

Effect of software product

influences

influences

influences

process quality

internal quality attributes

external quality attributes

quality in use attributes

depends on

depends on

Process measures (ISO 15504)

Internal measures

External measures

Quality in use measures

Context of use

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.

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

“Analysabillity”

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 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

e.g. 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

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’