Software Engineering Economics Lecture1

Dr. Rasha Ismail
E-mail: Rashaismail@yahoo.com

Course Outline

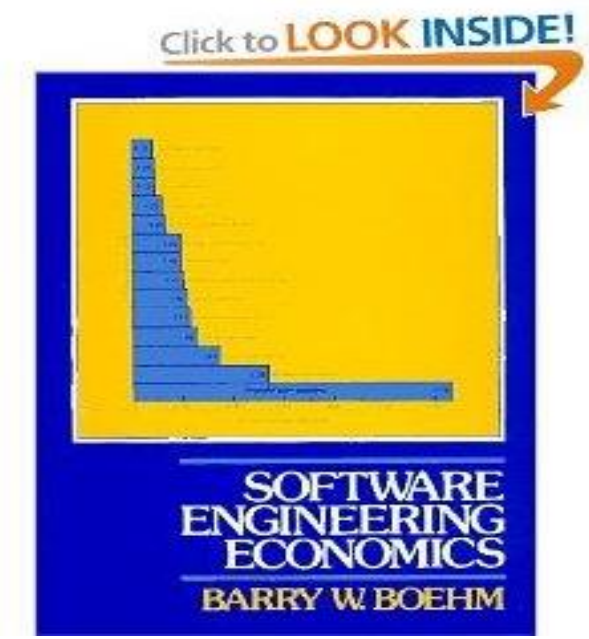
1. Introduction
2. Software Engineering Goals
3. Software Quality Assurance
4. Software Life Cycle – Quantitative model
5. The Basic COCOMO Model
6. The Basic COCOMO Model-Development Mode
7. The Basic COCOMO Model-Activity Distribution
8. The Intermediate COCOMO Model-Product Level Estimation
9. The Intermediate COCOMO Model-Components Level Estimation
10. COCOMO2 Model
11. Software Testing
12. Conclusion & Summary

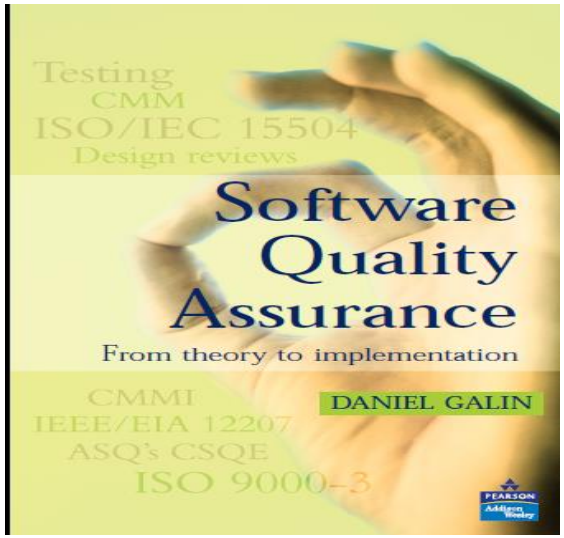
Text Book

1- Barry W. Boehm,"Software Engineering Economics"., Englewood Cliffs, NJ : Prentice-Hall,1981

2- Galin, D., "Software Quality Assurance: From Theory to Implementation", Addison Wesley (2003)

3- Jerry Zeyu Gao, Jacob Tsao, Ye Wu,"Testing and Quality Assurance for Component-Based Software",Pearson,2006





Assessment

- Coursework 20%
  - -project
  - -Assignments
- Final Exam 80%

I.1Motivation

The Software Crisis:

- IBM Consulting group estimates that 55% of large distributed systems projects cost more than expected,
- 68 % overrun their schedules, and 88% require redesign.
- The Standish group estimated the cost of ‘bad software’ for US businesses at $99 billion for 2000.

Some Famous Software Bugs

Therac-25 Problem

- When operating in soft X-ray mode, the machine was designed to rotate three components into the path of the electron beam, in order to shape and moderate the power of the beam. …

- The accidents occurred when the high-energy electron-beam was activated without the target having been rotated into place; the machine's software did not detect that this had occurred, and did not therefore determine that the patient was receiving a potentially lethal dose of radiation, or prevent this from occurring.

Therac-25 - the reasons

- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
- The engineer had reused software from older models. These models had hardware interlocks and were therefore not as vulnerable to the software defects.
- The hardware provided no way for the software to verify that sensors were working correctly.
- The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly. This was evidently missed during testing, since it took some practice before operators were able to work quickly enough for the problem to occur.
- The software set a flag variable by incrementing it. Occasionally an arithmetic overflow occurred, causing the software to bypass safety checks.

Ariane 5 Rocket

- June 4, 1996 was the first test flight of the Ariane 5 launch system. The rocket tore itself apart 37 seconds after launch, making the fault one of the most expensive computer bugs in history.

- The Ariane 5 software reused the specifications from the Ariane 4, but the Ariane 5's flight path was considerably different and beyond the range for which the reused code had been designed.

- Because of the different flight path, a data conversion from a 64-bit floating point to 16-bit signed integer caused a hardware exception (more specifically, an arithmetic overflow, as the floating point number had a value too large to be represented by a 16-bit signed integer). Efficiency considerations had led to the disabling of the exception handler for this error. This led to a cascade of problems, culminating in destruction of the entire flight.

- **The Mars Climate Orbiter crashed in September 1999 due to a software defect**

  "The 'root cause' of the loss of the spacecraft was the failed translation of English units into metric units in a segment of ground-based, navigation-related mission software, as NASA has previously announced,“

  http://mars.jpl.nasa.gov/msp98/news/mco991110.html

- **Airbus A320 crashed at an air show in 1988**

Cause: "The pilot claims he was misled on the aircraft's true height by a bug in the software”

http://catless.ncl.ac.uk/Risks/8.77.html#subj6

How can we circumvent such disasters?

Observation:

- Quality problems prevent the system from functioning
- Bad SW cost estimation
- Management decisions results in wrong reactions
  - Missing design reviews
  - Fixing rather than analyzing
  - Erratic debugging rather than systematic testing

Systematic approach for the development of high quality software is required

What is software?

Software is:

Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

What is software Engineering?

Software Engineering is:

a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build

an disciplined and quantifiable approach that is concerned with all aspects of software product from the early stage of system specifications to maintaining after it has gone into use

the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation

software Trends

1. Cost
2. Social Impact
3. Quality /Testing

Software Trends: Cost

- We perform SW eng to determines the cost and the quality of produced SW.

- Software is a large and increasingly costly item, Software makes a large and increasing impact on human welfare.

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

- Software costs more to maintain than it does to develop. For systems with a long life cycle, maintenance costs may be several times development costs.[43.3% for development, 48.8% for the maintenance and 7.9% for other activities]

Software Trends: Cost

Software engineering is concerned with:

- Cost-effective software development
- Increase the SW development productivity(No of lines per man-day)

Productivity estimates are usually on measuring

$$= \frac{\text{attributes of software}}{\text{total effort required for development}}$$

- Increase the efficiency of SW maintenance

Software Trends: Social Impact

The increasing impact on human welfare presents several challenges for the software engineering profession. They are to develop and maintain SW which ensures that computer systems are:

- Reliable
- Easy to use
- Useful to man
- Hard to misuse
- Auditable

+ Economic productivity and Maintainability

(see the SW factors)

Software Trends: Quality

What is quality?

Various definitions:

- “The ability to satisfy stated or implied needs“ [ISO 8402]
- “A product or service free of deficiencies“ [American Society for Quality]
- “Fitness for use“ [Joseph M. Juran]
- “Degree to which a set of inherent characteristic fulfills requirements“ [ISO 9001]

This is problematical for software systems

- There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
- Some quality requirements are difficult to specify
- Software specifications are usually incomplete and often inconsistent.

Approaches to Tackle Quality

- **Transcendental view**: quality is universally identifiable, absolute, unique and perfect
- **Product view**: the quality of a product is measurable in an objective manner
- **User view**: quality is fitness for use
- **Manufacturing view**: quality is the result of the right development of the product
- **Value-based view (Economic)**: quality is a function of costs and benefits

Software Quality- IEEE Definition

Software quality is:

(1) The degree to which a system, component, or process meets specified requirements.

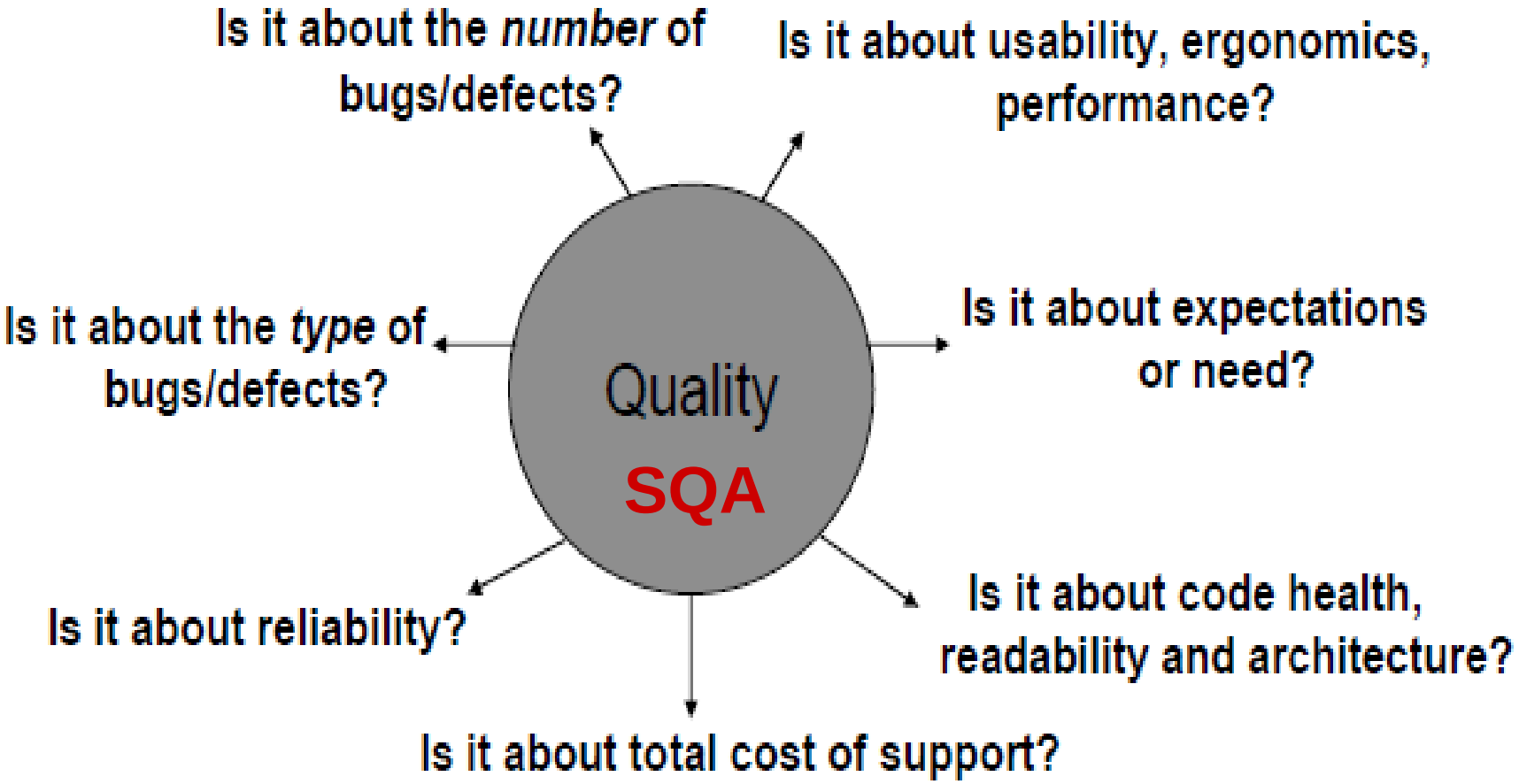
(2) The degree to which a system, component, or process meets customer or user needs or expectations.

[IEEE_Std_610.12-1990]

Software quality assurance

Software quality assurance is:

A systematic, planned set of actions necessary to provide adequate confidence that the software development process (Process oriented) or the maintenance process (Product oriented) of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines.


Is it about the *number* of bugs/defects?
Is it about usability, ergonomics, performance?
Is it about the *type* of bugs/defects?
Quality
SQA
Is it about expectations or need?
Is it about reliability?
Is it about code health, readability and architecture?
Is it about total cost of support?

Software quality assurance vs. software quality control

Quality Control [ISO 9000]: The operational techniques and activities that are used to fulfill requirements for quality

Quality Control is the series of inspections, reviews and tests used throughout the development cycle to ensure that each work product meets the requirements placed upon it.[Pressman2004]

while

Quality Assurance is to minimize the cost of guaranteeing quality by a variety of activities performed throughout the development and manufacturing processes/stages.

- *These Quality control activities are only a part of the total range of quality assurance activities.*

Do we need special Quality and cost models for software?

Why do the classical approaches to Quality and cost not apply to the software?

The uniqueness of the software development process

Characteristic	Software products	Other industrial products
Complexity	Usually, very complex product allowing for very large number of operational options	Degree of complexity much lower, allowing at most a few thousand operational options
Visibility of product	Invisible product, impossible to detect defects or omissions by sight (e.g. of CD storing the software)	Visible product, allowing effective detection of defects by sight
Nature of development and production process	Opportunities to detect defects arise in only one phase, namely product development Different SW objectives do indeed conflict	Opportunities to detect defects arise in all phases of development and production: ■ Product development ■ Product production planning ■ Manufacturing

What is special about Software?

- Invisibility of the product
- Limited opportunities to detect defects(“bugs”)
- Often new demanding functionality has to be realized
- Often software has to realize extraordinary high complexity
- Different SW objectives do indeed conflict

Human Resource Package

- “We’ve used the Simplex HR software in our Human Resources Department for about three years and we have never had a software failure.”

- “I started to use Simplex HR two months ago; we had so many failures that we are considering replacing the software package.”

- “We have been using the same software package for almost four years. We were very satisfied throughout the period until the last few months,when we suddenly faced several severe failures. The Support Center of the software house from which we bought the package claims that they have never encountered failures of the type we experienced even though they serve about 700 customers who utilize Simplex HR.”

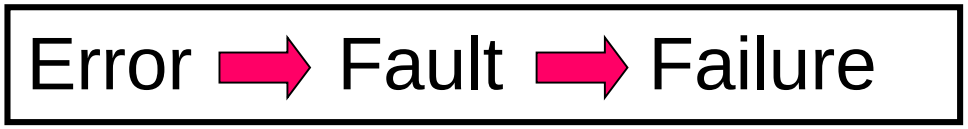
Is it possible for such a variation in users' experience with failure to appear with the same software package?

Can a software package that successfully served an organization for a long period "suddenly" change its nature (quality) and become "bugged"?

Errors, Faults and Failures

- An **error** is a derivation of the required operation of the system or subsystem.
  (Manifestation of a fault in a system)
- **Software errors** are a human action which results in software containing a fault
- A **fault** is a defect within the system.
  (Error cause)
- A **system failure** occurs when the system fails to perform its required function.
  (Transition to incorrect service delivery)
- If the distinction between fault and failure is not critical, **defect** can be used as a generic term

Fault/Failure Chain



Error ⇨ Fault

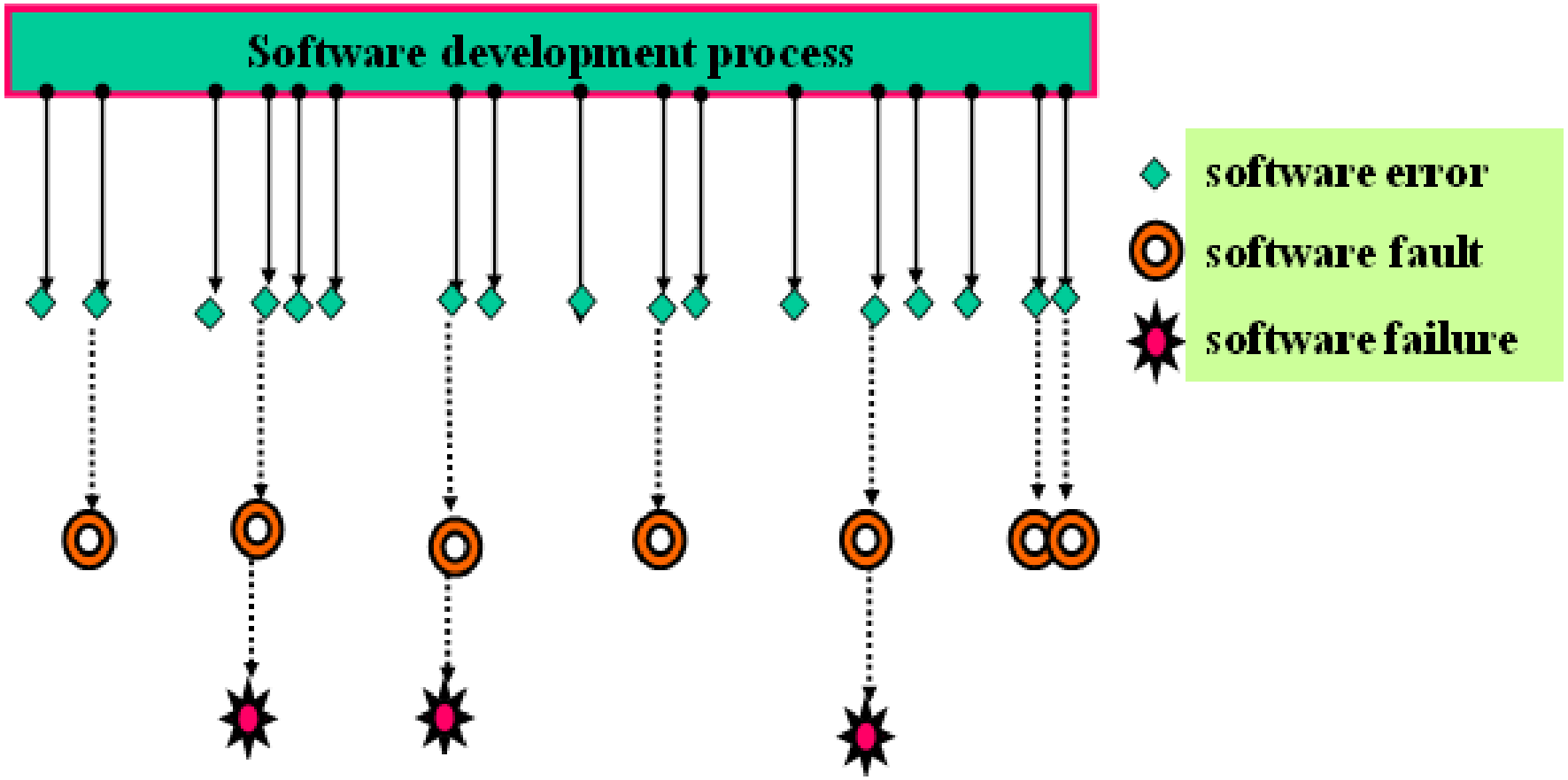
- an error is *latent* when it has not been recognized
- an error which has not been activated by the computation process is *dormant*
- an error is *active* when it produces a fault

Fault ⇨ Failure

- a failure occurs when an error “passes through” and affects the service delivered

Example

- *The "Meteoro-X" meteorological equipment firmware* The software requirements for "Meteoro-X" meteorological equipment firmware (software embedded in the product) were meant to block the equipment's operation when its internal temperature rose above 60°C.
- A programmer error resulted in a software fault when the temperature limit was coded as 160°.
- This fault could cause damage when the equipment was subjected to temperatures higher than 60°.
- Because the equipment was used only in those coastal areas where temperatures never exceeded 60°, the software fault never turned into a software failure.



This figure illustrates the relationships between software errors, faults and failures. In this figure, the development process yields 17 software errors, only eight of which become software faults. Of these faults, only three turnout to be software failures.

Causes of software errors

1) Faulty requirements definition

- One of the main causes of software errors
- Erroneous definition of requirements
- Missing vital requirements
- Incomplete requirements
- Unnecessary requirements

2) Client-developer communication failures

- Misunderstanding of the client's requirement document
- Miscommunications during client-developer meetings
- Misinterpretation of the client's requirement changes

Causes of software errors (*cont*.)

3) Deliberate deviations from software requirements

- The developer reuses software modules from earlier projects without sufficient analysis of the changes needed to fulfill the requirements
- Due to time/money pressures, the developer leaves parts of the required functions in an attempt to manage the pressure
- The developer makes unapproved improvements to the software without the client's knowledge

4) Logical design errors

- Defining the software requirements by means of erroneous algorithms
- Process definitions contain sequencing errors
- Erroneous definition of boundary conditions
- Omission of definitions concerning special cases (lack of error handling and special state handling)

Causes of software errors (*cont*.)

5) Coding errors

- Misunderstanding of the design documents, linguistic errors, development tool errors, and so forth

6) Non-compliance with documentation and coding instructions

- Non-compliance with coding/documentation standards makes it harder to understand, review, test and maintain the software

7) Shortcomings of the testing process

- Greater number of errors are left undetected and uncorrected
- Incomplete test plans will leave many of the functions and states untested
- Failures to report and promptly correct found errors and faults
- Incomplete test due to time pressures

8) Procedure errors

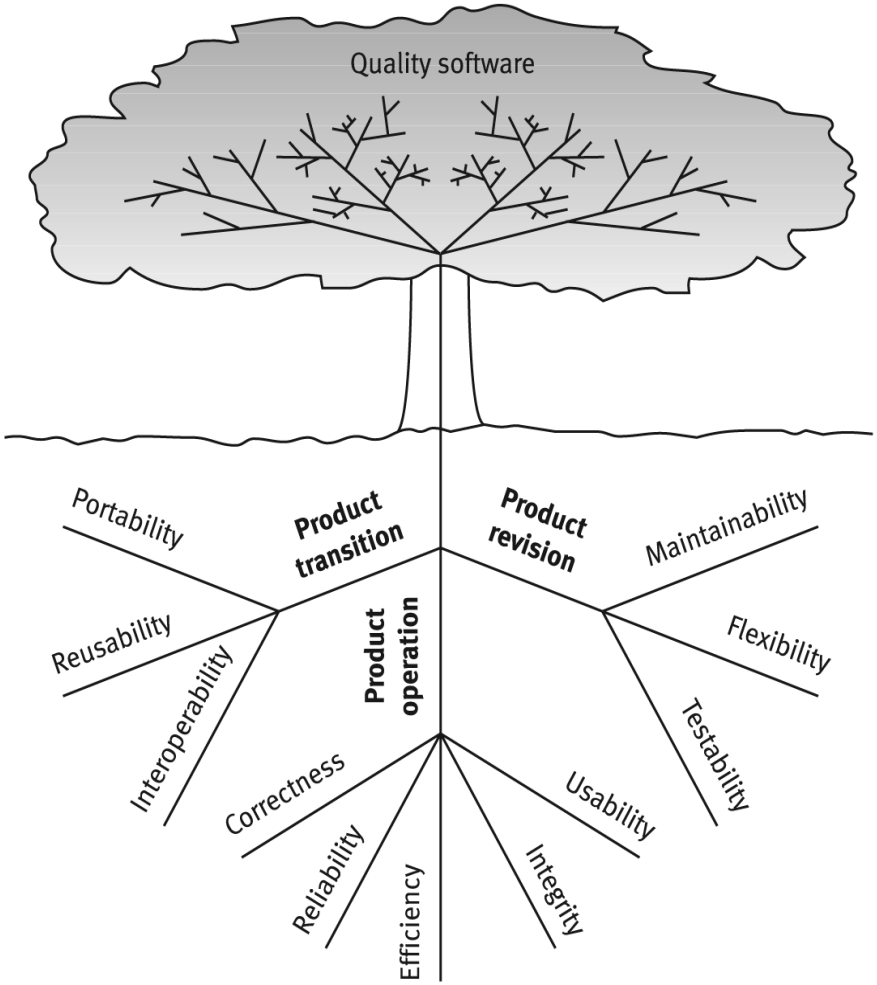
9) Documentation errors

Question

- To know that quality has improved, it would be helpful to be able to measure quality(useful to man).
- **How can we measure quality?**

  by determining the quality factors
- Software Quality Factors
  - how do you know quality when you see it?
  - how do you measure quality?

McCall's Factor Model Tree



- McCall's quality factor model
- 11 quality factors Grouped into 3 categories:

1) Product operation factors

2) Product revision factors

3) Product transition factors

Product Operation Factors

Deal with requirements that directly affect the daily operation of the s/w.

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability

Correctness

Defined by outputs

- The output mission (invoice printout)
- Required accuracy (inaccurate data/calculations)
- Completeness (incomplete data)
- Up-to-date (defined as the time between the event and its consideration by the software system)
- event and its consideration by the software system
- Availability (reaction time or response time to get information)
- Standards (for coding & documenting the software system)

Correctness Example (club membership info system)

1. Output mission: list of x reports, y standard letters, z interactive queries.
2. Required accuracy: probability of a non-accurate output <1%.
3. Completeness: probability of missing data about a member, attendance, payments etc < 1%.
4. Up-to-dateness: no more than 2 days for info about event participation to be valid
5. Availability: reaction time to queries < 2 seconds, to reports < 4 hours
6. Standards: software & its documents to comply with client's guidelines

Reliability

- Rate of failure occurrence.
- Deals with failure to provide service.
- Determine max allowed failure rate.
- Can refer to entire system or separate functions
- Deals with recovery
- Deals with fault tolerance.

- Ex:
  – Failure of heart monitor required to be < 1 per 20 years. Heart attack detection < 1 per million cases.
  – Bank branch SW fails < 10 mins per month in office hours; recovery time < 0.5%

Efficiency

- Deal with HW resources.
- Processing power (MIPS, MHz)
- Storage capacity (GBytes, TBytes)
- Data Communications (MBPS, GBPS)

Integrity

- Security requirements
- Access to unauthorized personnel
- Access levels (read v write, user v sys)
- Deliberate attacks; Accidental
- losses/damage; Backups.
- Ex:

  Local authority GIS system. Public to view maps etc, but not edit.

Usability

- Scope of staff resources needed

– to train new employees

– to operate software system

- Ex: Help Desk in service company specs

– One staff to handle 60 calls per day

– Two days to train new staff who will then be able to handle 45 calls per day

Product revision factors

Deal with requirements that affect the complete range of s/w maintenance activities (corrective, adaptive, perfective)

- Maintainability
- Flexibility
- Testability

Maintainability

- Determine efforts needed by users and maintenance personnel to
  - identify reasons for s/w failures
  - to verify success of corrections
- Refer to
  - modular structure of s/w
  - Internal program documentation
  - Programmers manual
- Typical requirements:
  - Typical s/w module <30 statements
  - Code to company standards and guidelines

Flexibility

- Capabilities & efforts to support adaptive maintenance activities
- Example: adapt a software package to
  - different customers
  - different activities
  - different product ranges etc
- Requirements to support perfective s/w
  - improve service
  - adapt to changes in technical/commercial environment
- Ex: Teacher Support s/w:
  - should adapt to schools of all levels
  - non-professionals to be able to create new reports according to the school teacher's requirements and/or the city's education department demands.

Testability

- Requirements deal with testing system itself & its operation
- Special program features for intermediate results and log files
- Automated diagnosis prior to start/at start/on demand for fault reporting
- Ex: Testability requirement to develop standard test data against which to validate production line performance at each stage (run each morning?)

Product Transition Factors

Pertains to the adaptation of s/w to other environments and its interaction with other s/w systems

- Portability
- Reusability
- Interoperability

Portability

- Portability requirements tend to the adaptation of a s/w system to other environments
  - hardware
  - operating systems etc
- Use same s/w in diverse situations
- Ex: windows NT program required to work in Vista and Linux

Reusability

- Deal with the use of software modules originally designed for one project in a new software project currently being developed.

Benefits:

– Save development resources
– Save test resources
– Shorten development period
– Provide higher quality (already quality assured, tested and used)

Interoperability

- Focus on creating interfaces with other s/w systems
- Requirements can specify eg the names of s/w or firmware for which interface is required.

Alternative Models to McCall

- Basically extend McCall's factors/requirements by 5:

12. Verifiability: design & programming features for efficient verification (modularity, simplicity, adherence to doc & prog standards)
13. Expandability: future efforts for improved service, bigger system, increased user numbers. "flexibility"

Alternative Models to McCall 2

14. Safety: eliminate hazardous conditions as a result of errors in process control software (eg failure to provide alarm signals, inappropriate reactions to dangerous conditions)
15. Manageability: refer to admin tools for s./w modification during development and maintenance eg configuration management, change procedures

”

McCall's Factor Model and Alternative Models

No.	Software quality factor	McCall's classic model	Alternative factor models	
			Evans and Marciniak model	Deutsch and Willis model
1	Correctness	+	+	+
2	Reliability	+	+	+
3	Efficiency	+	+	+
4	Integrity	+	+	+
5	Usability	+	+	+
6	Maintainability	+	+	+
7	Flexibility	+	+	+
8	Testability	+		
9	Portability	+	+	+
10	Reusability	+	+	+
11	Interoperability	+	+	+
12	Verifiability		+	+
13	Expandability		+	+
14	Safety			+
15	Manageability			+
16	Survivability			+

Assignment

1) Determine your project.
2) Specify the teamwork

Then submit a report for the following tasks:

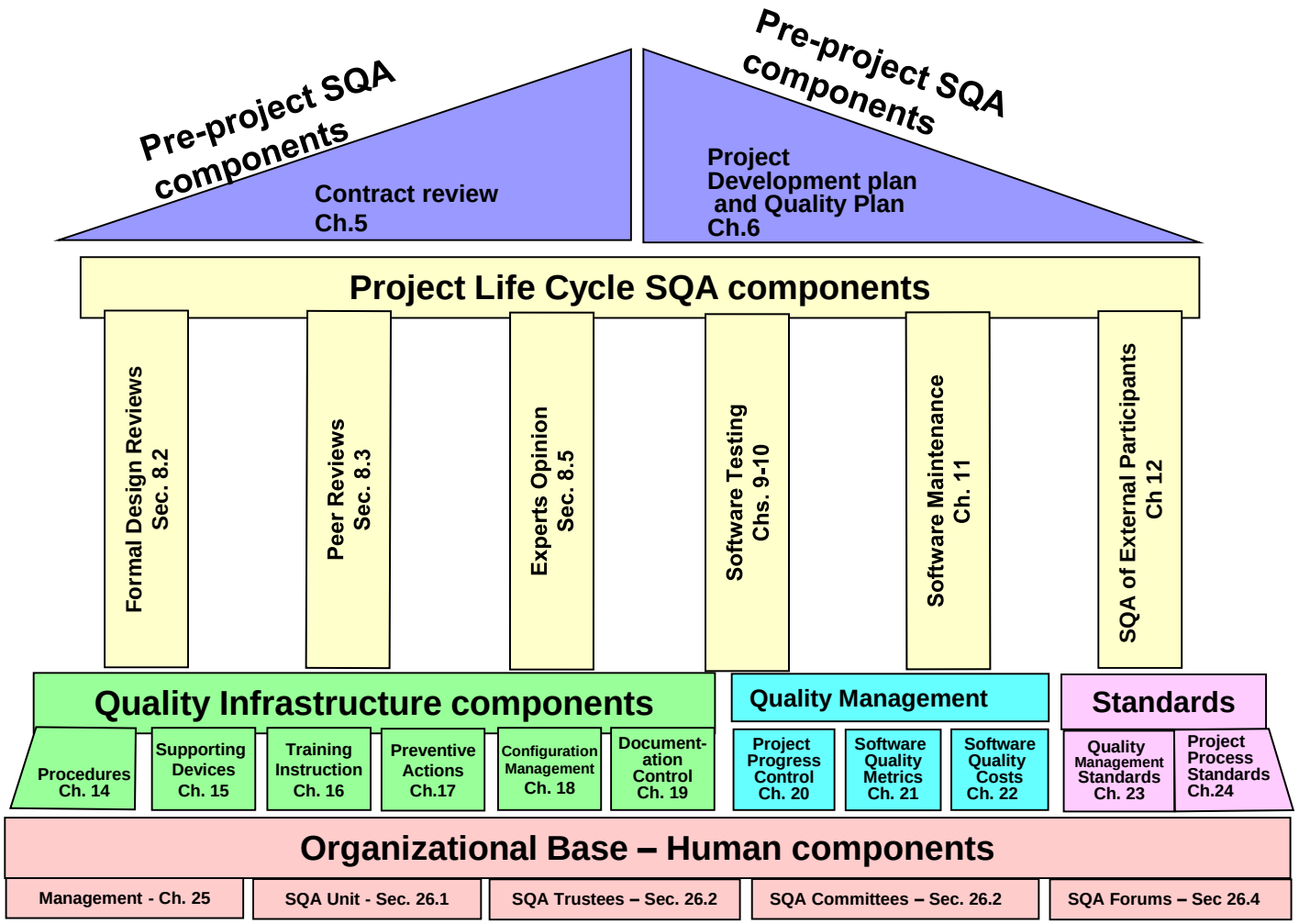
- Describe your system
- Apply the software McCall's Factor to your project?

Components of software quality assurance system overview

- Pre-project components
- Software project life cycle components
- Infrastructure components for error prevention and improvements
- Management SQA components
- SQA standards, system certification and assessment components
- Organizing for SQA – the human components
- Considerations guiding construction of organization's SQA system

OHT 4.59

The Software Quality Architecture



Pre-project SQA components

- Contract reviews
- Development and quality plans

Software project life cycle components

- Reviews
- Expert opinions
- Software testing
- Software maintenance components
- Assurance of the quality of external participants' work

Infrastructure components for prevention and improvement

- Procedures and work instruction
- Templates and checklists
- Staff training, retraining and certification
- Preventive and corrective actions
- Configuration management
- Documentation control

Management SQA components

- Project progress control
- Software quality metrics
- Software quality costs

SQA standards, system certification and assessment componenets

- Project process standards
- Quality management standards

Objectives:

- Utilization of international professional knowledge
- Improvement of coordination with other organizations' quality systems
- Objective professional evaluation and measurement of the organization's SQA achievement

Organizing for SQA - the human components

- Management's role in SQA
- The SQA unit
- SQA trusties
- SQA committees
- SQA forums

Thank you for your attention!

Questions are welcomed!