QUALITY MANAGEMENT

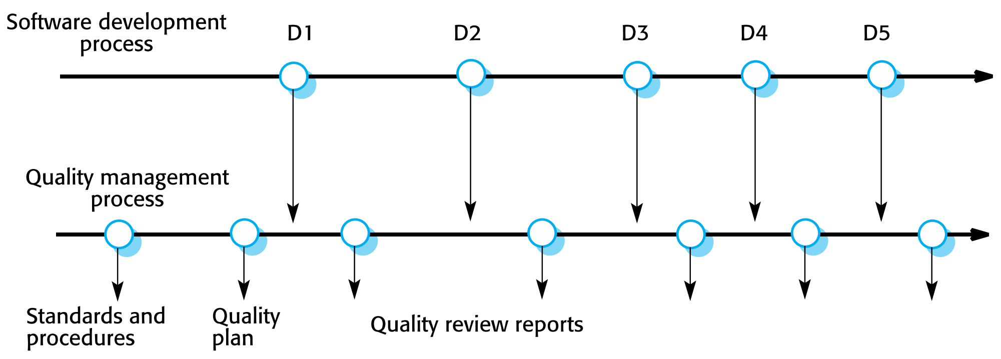
Software quality management:

- Concerned with ensuring that required level of quality is achieved in a software product
- Three concerns:
    - Organization level, standards for quality should be set by the organization (either developing side or client side), which if followed will ensure a high-quality product
    - Project level, planned quality processes should be defined by the team specifically assigned to that project in order to ensure quality. Organizations may give freedom to the team to themselves set standards for the quality of the product. A quality plan can also be set to define quality goals to be achieved and standards to be used

Quality management activities:

- Quality and development team should be independent when resizing, to avoid biasness by software developers. Hence, all software is run through all sorts of test-cases to ensure it delivers the required quality
- QM also ensures that project deliverables are inline and consistent with the standards set by the organization

Quality management and software development:



- The diagram shows that D1, D2 deliverables will be checked according to the QM process like checking the standard procedures, then analyzing the quality plan and then reviewing the reports etc.

Quality planning:

- Quality plan contains:
    - the desired product qualities (functional or mostly non-functional) set by the organization or client along with most significant quality attributes in which there is no compromise like safety in case of insulin pump case study etc.
    - Quality assessment such as test-cases to test defined quality attributes
    - Check which organizational standards are being applied and may also require to define new standards

Quality Plans:

- Plan structure:
    - Product introduction: A description of the product, its intended market, and the quality expectations for the product.
    - Product plans: The critical release dates and responsibilities for the product, along with plans for distribution and product servicing.
    - Process descriptions: The development and service processes and standards that should be used for product development and management.
    - Quality goals: The quality goals and plans for the product, including an identification and justification of critical product quality attributes.
    - Risks and risk management: The key risks that might affect product quality and the actions to be taken to address these risks.
- These plans should be short so easy to read

Scope of QM:

- Important for large complex systems
- Quality documentation: record of progress to support development continuity in-case of team changes
- For smaller systems this documentation is short, more focus on maintaing quality culture

- Differnet for agile as less documentation required there

Software quality:

- Quality: product should meets user specification
- Problems for software:
    - As customer has less domain knowledge hence differenece between customer and developer set quality requirements. E.g user wants a interactive UI for a database while developer want to make it more secure
    - Quality requirements maybe unclean or ambigous
    - Non-functional or functional requirements might be inconsistent such as requirements changes with increments for certain models
- Hence, focus should be on fitting it for purpose rather then focusing on minute specification details

Software fitness for purpose:

- Some points should be be focused as:
    - Proper testing
    - Performance testing
    - Well-structured and understandable
    - Standards should be set for documentation

Non-functional charactersitics:

- Software have large dependance on these non-functional attributes
- Such as one function not working causes less concern rather than unbearable downtimes, less secure etc.
- Quality and non-functional attributes are almost similar

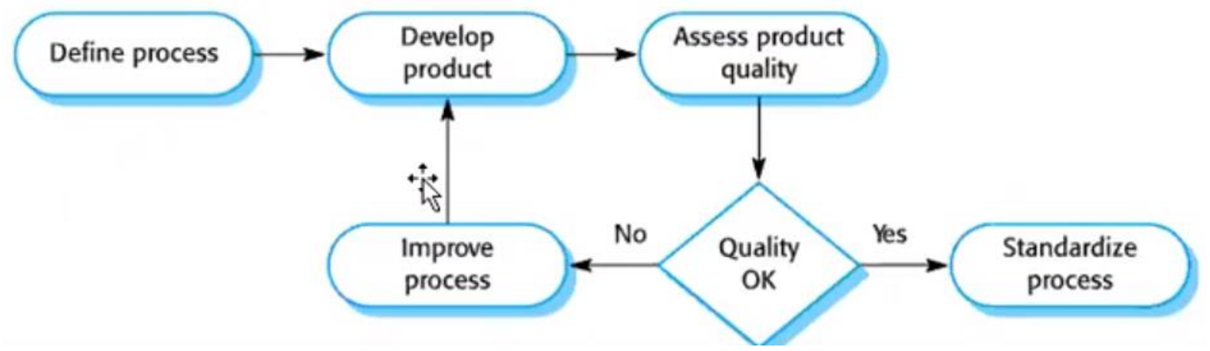
Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Quality conflicts:

- Trade off on quality attributes as optimizing all is impossible. As making system better performing may lead to more maintainence down times (availibilty).

- Hence, plan must focus or define the most important of these attributes, along with their assessment such as testing for robustness that where the system fails and recovers

Process-based quality cycle:



- Standarize means to make it a common practice in organization

Quality culture:

- Managers should enforce this culture where everyone should be committed to achieve high software quality and may develop new ways to improve quality. Moreover, they should also support such who encourage to do so

Software Standards:

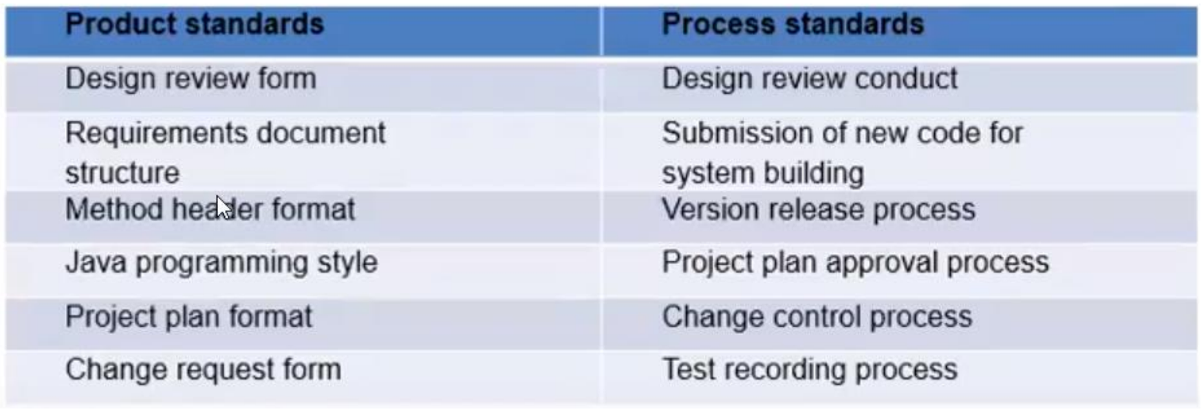
- Standard: provide framework that define the required attributes of a quality product / process.
- Vary from organization, national(HEC), international(ISO, IEEE), project related etc. Companies following international standards or standards of renowned companies are preferable

Importance standards:

- Best practices should be encapsulated so that they are referred to for software development hence helping to avoid past mistakes
- Help to provide continuity for new staff so that they follow the standards set by the organization

Product and Process Standards:

- Product standards: apply to the product being developed such as, programming standards to follow as comment headers, coding standards etc. Moreover, documentation structure etc.
- Process standards: this applies to the process of forming the standards for the product development



Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

- E.g company deciding to form SRS is the process standard. While deciding the format of the SRS is product standard

Problem with standards:

- Standards quickly get outdated due to ever evolving industry.
- Moreover, documenting standard is tedious hence, reluctance to maintain this
- Without version control, maintaining standards documentation is also hefty hence avoided.

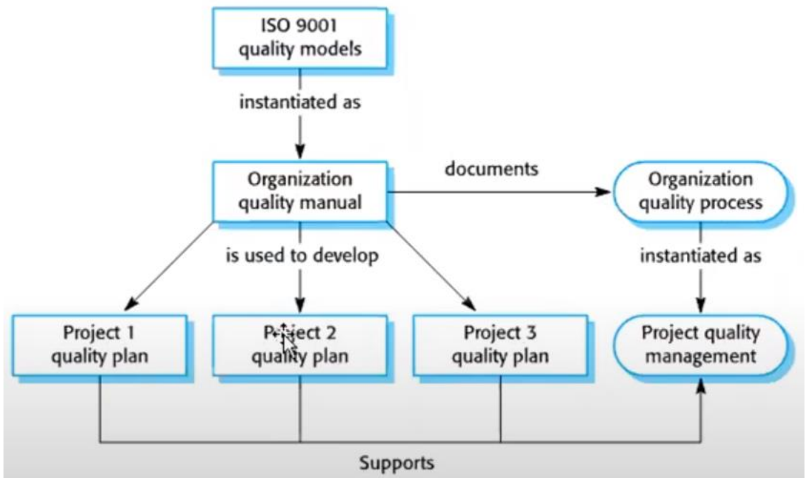
Standards development:

- Practitioners (those that have domain knowledge) form standards mostly and they should understand rationale underlying a standard.
- IEEE and ISO were general standards hence practitioners make standards
- Standards need to be reviewed and used regularly as they get outdated fast.
- Standards involving much documentation should be made using specialized tools

ISO 9001 standards frameworks:

- ISO is governing body to set and enforce standards
- The 9001 framework is used to ensure very generalized QM standards for organizations that involve software pertaining to any category

- Basically, highlights general quality principle and defines standards and procedures to be followed.
- Organization who follow this should document this in their quality manuals. Quality attributes are also added from post implementation experiences



Software quality and ISO framework:

- Organizations are reluctant to follow as these standards takes no account of quality experienced by users. Such as following standards may lead to incomplete testing parameters and hence, poor quality for end-user as these standards are generalized and not user-specific.

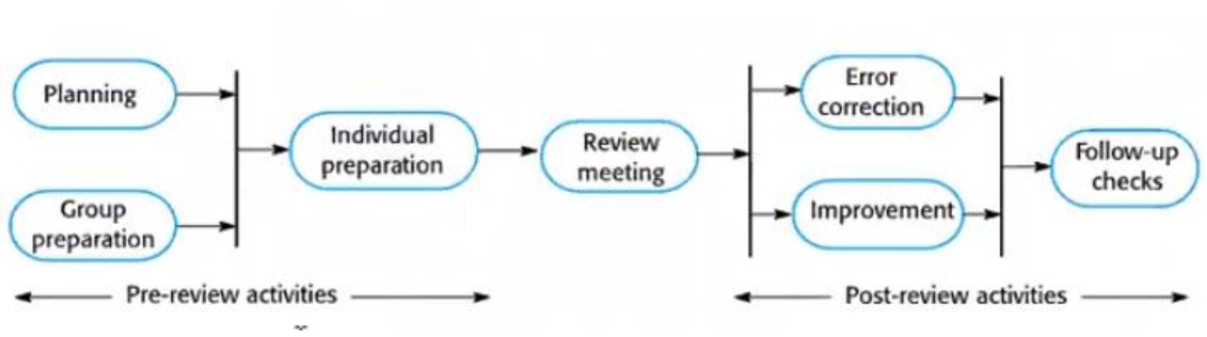
Review and Inspections:

- This is to ensure and maintain product quality by finding potential problems by a team. Project only progresses if 'signed-off' by these teams.
- Review is basically high-level analysis of the product and relevant documentation
- Inspection is a more-detailed form of review (lower-level) such as analyzing coding conventions etc.
- Types:
  - Inspect to remove defect (product)
  - Review for progress (product + process): ensuring deadlines are met etc.
  - Quality reviews (product): ensuring software in accordance with the QM manual set

Quality reviews:

- Code, design, specifications, test plans are all reviewed. Code is reviewed using dry runs, brief overview of logic etc.

Phases in review process:



- Planning: review data, resources, documents to be used when review Individual preparation: members individually analyze the agenda and record their reviews
- Review meeting: author of the part being reviewed, explains the members about how the job was carried out which is then analyzed by the members
- Follow-up checks: this ensures that after the improvement, was quality ensured or not

Distributed reviews:

- Organizations are more distributed and less co-located hence, reviews are done using video conferencing and cloud platforms and other means are used for resource sharing and correction

Program inspections:

- Peer review the program in order to detect anomalies and defects.
- Can be used before implementation and considered to be effective
- Any source could be inspected e.g code, design, test data, etc.

Inspection checklists:

- These are detailed hence a checklist of common errors should be made for inspections.
- Dependent on the programming platform used
- Some faults:
  - Data faults: inspecting incoming data for data structures

    - Control faults: control statement inspections as loops, conditions
    - Input/output faults: variable initialization inspections
    - Interface faults: functions, method, components inspection
    - Storage faults: types of data structures inspection such as allocation, deallocation, a specific structure used
    - Exception faults: exception handling inspection

Quality management and agile development:

- QM is made easy, freedom based and informal as it is between inter-organization members
- However, quality culture should be maintained instead of formal documentation, hence this is different from ISO based standard approach

Shared good practice:

- Check before check-in: programmers should themselves get their code reviewed for bugs before deploying to build
- Never break build: don't check in code that may have bugs to the whole system hence unit testing is necessary.
- Fix problems when you see them: fix bugs directly when reviewing code rather than referring to original developer

Review and agile methods:

- Different for each software approach:
    - Agile: informal review process
    - Scrum: review meeting after each iteration (sprint review)
    - Extreme programming: peer ensures that code is constantly reviewed

Pair Programming weaknesses:

- Both peers made same mistake of misunderstanding system. Discussions required to overcome this
- Pairs may try to ignore bugs for the sake of avoiding hinderance to project progress
- Working relationships may hinder pairs to criticize partner regarding bugs

Agile QM and large systems:

- Agile approaches in this case is impractical.
- This is because:
    - Large companies want developer companies to maintain their standard hence, agile QM not valid
    - Informal communications impossible in case of distributed teams
    - Documentation is necessary for such large systems else continuity can be compromised

Software Measurement:

- Quality attributes are qualitative and not quantitative
- Hence, we need method to quantify attributes as robustness, reliability, dependibility etc.
- Large companies make use of measurement tool inorder to perform QM. However, smaller companies limit themselves to test cases.

Software Metrics:

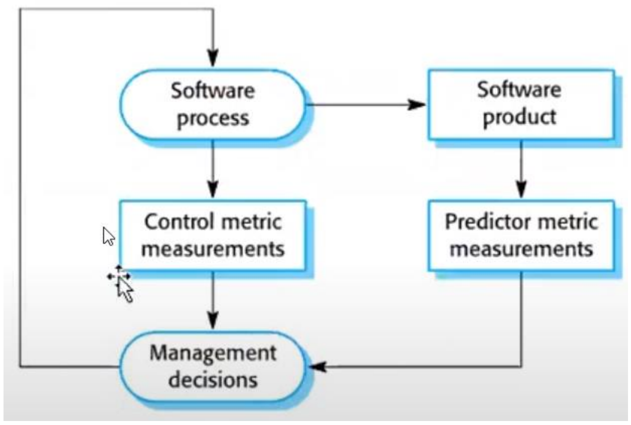
- Metric: any measurement that relates to software system, process or related documentation and allow to quantify them. E.g lines of code, duration, fog index (time to understand program)
- Can be used for predictions, identification of buggy components, control software processes etc.

Types of process metrics:

- Time duration
- Computational cost (resources required)
- Count number of occurences of an event such as requirements modification, by stakeholder, number of bugs, avg. loc for changing requirements
- E.g machine learning (developing time is less but computational cost is higher)

Predictor and control measurements:

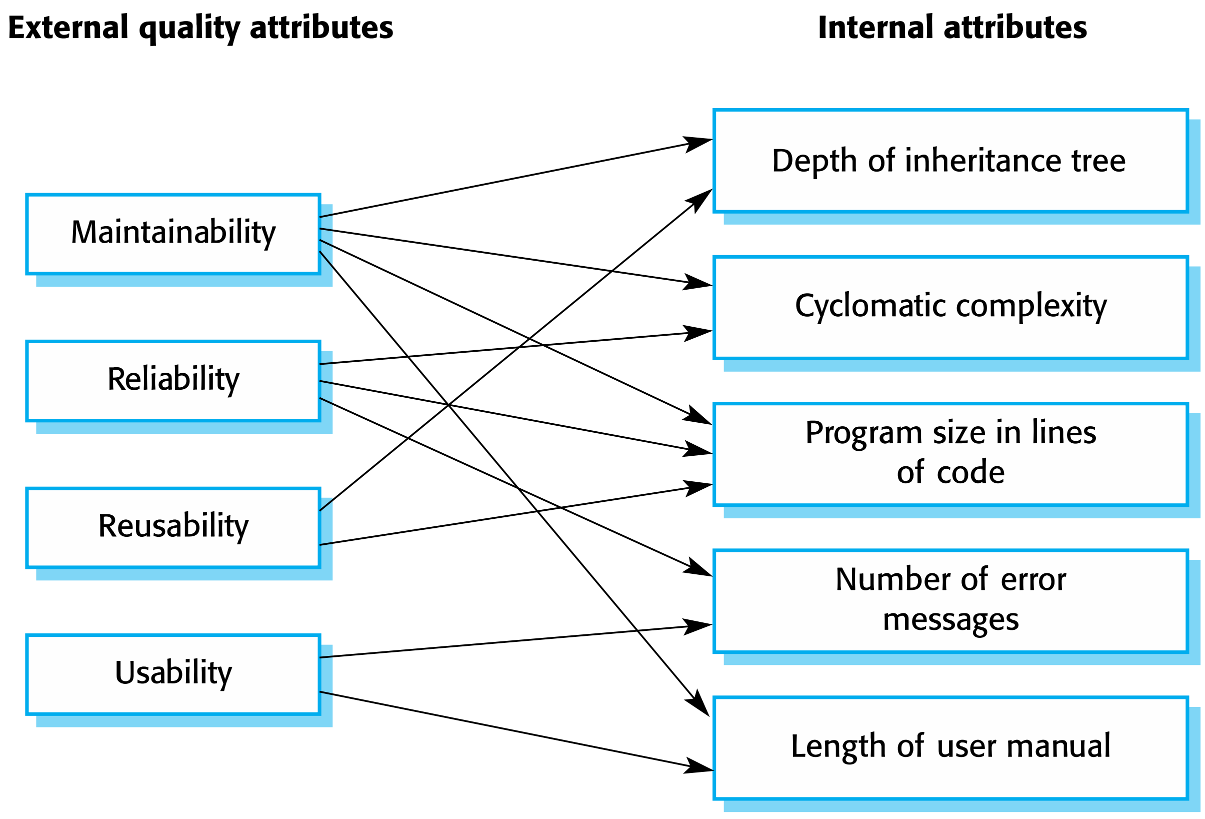
- Software process: agile, scrum etc.
- Then control metrics are determined. After completion, the metrics and the predicted metrics are compared by the management team to decide whether quality was ensured or not



Use of measurements:

- To assign numeric values to system quality attributes
    - Such as Cyclomatic complexity: quantitative measurement (a numeric value) of the complexity of the system
- To identify components with ‘sub-standard’ quality, hence helping to rectify those

Relationship between internal and external attributes:



- This signifies that we can monitor internal attributes in order to ensure that our system fulfils the external quality attributes

Metric assumptions:

Some assumptions underlying metrics include:

- Software property can be accurately measured
- There must be a internal attributes that we can measure for all the external quality attributes that we require. For e.g measuring usability by ensuring less error messages

Product metrics:

- The metrics should be a good predictor of quality
- Classes of product metrics:
    - Dynamic metrics:
        - measurements taken through program execution.
        - help to assess efficiency and reliability
        - closely related to software quality attributes, as can be practically determined
    - Static metrics:
        - measurements taken through documentations, system representations, test-cases
        - help to assess understandibility, complexity, maintainability
        - indirectly related to software quality attributes as we need to derive a relationship in order to measure the attributes

Static software product metrics:

Metrics	Description
Fan-in/fan-out	Deals with coupling and cohesion. Fan in: number of function calls for a function. Greater means more coupling Fan out: number of function calls made inside a function. Greater means more complex due to more calls
Length of Code	More size of code, more complex and error prone
Cyclomatic complexity	Numeric measure of complexity of system
Length of identifiers	Concerned with variable names. More long names means greater understandibility
Depth of conditional nesting	Deep nesting of if-else means less understandability and more error prone
Fog Index	Concerned with manuals. More wordiness in

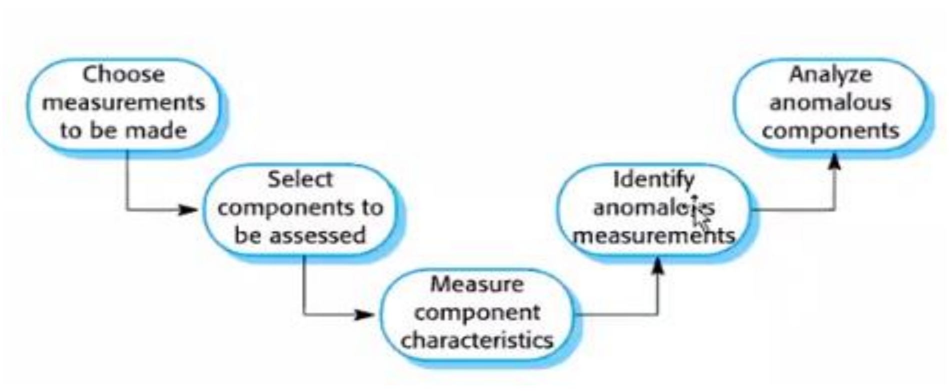
documentation means less understandable

CK object-oriented metrics suite:

OO metric	Description
Weighted methods per class (WMC)	Number of methods + complexity of each. Greater the value more complex, less understandable, less cohesive hence less resuable
Depth of inheritance tree (DIT)	Measure inheritance depth of classes. More inheritance, means more complex to understand superclasses and subclasses
Number of children (NOC)	Measure of immediate or direct subclasses. Measure breadth while DIT measures depth. Great reuse but need more validation as more classes depend on them
Coupling between object classes (CBO)	Greater coupling means changing one class can affect more classes hence less maintainable
Response for a class (RFC)	Number of methods calls made by class on receiving input hence more means more complex
Lack of cohesion in methods (LCOM)	Difference between shared and non-shared attributes methods. However, less affect on any attribute

Software component analysis:

- components can be measured using range of metrics and the values can be compared for different components along with past project experiences to identify and analyze anomalous components


Choose measurements to be made
Select components to be assessed
Measure component characteristics
Identify anomalous measurements
Analyze anomalous components