

# Quality Management

## QUALITY CONCEPTS

### Quality –

- characteristic or attribute of something
- measurable characteristics — compare to known standards such as length, color, electrical properties, and malleability



Two kinds of quality:

- **quality of design** characteristics that designers specify for an item  
(e.g. grade of materials, tolerances, and performance specifications)
- **quality of conformance** - degree to which the design specifications are followed during manufacturing



Software development –

- quality of design - requirements, specifications, and the design of the system.
- quality of conformance - focused on implementation.

### Quality control –

- ⌘ Series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it.
- ⌘ Includes a feedback loop to the process that created the work product.
- ⌘ Combination of measurement and feedback tunes the process when the work products created fail to meet their specifications.
- ⌘ This approach views quality control as part of the manufacturing process.

### Quality assurance –



The auditing and reporting functions of management.



Goal of quality assurance - provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.

### Cost of quality-







includes all costs incurred in the pursuit of quality or in performing quality-related activities.






divided into costs associated with prevention, appraisal, and failure.


- *Prevention costs* include




-  quality planning
-  formal technical reviews
-  test equipment
-  training


- *Appraisal costs* include





-  in-process and interprocess inspection
-  equipment calibration and maintenance
-  testing

- *Failure costs* - those that would disappear if no defects appeared before shipping a product to customers.

-  *Internal failure costs* - incurred when we detect a defect in our product prior to shipment.

-  rework
-  repair
-  failure mode analysis




-  *External failure costs* - associated with defects found after the product has been shipped to the customer.

-  complaint resolution
-  product return and replacement
-  help line support
-  warranty work


## SOFTWARE QUALITY ASSURANCE


### Software quality -




Conformance to:

-  explicitly stated functional and performance requirements
-  explicitly documented development standards
-  implicit characteristics that are expected of all professionally developed software.




### SQA group -

-  SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting

 Asks questions like:

-  Does the software adequately meet the quality factors?
-  Has software development been conducted according to pre-established standards?
-  Have technical disciplines properly performed their roles as part of the SQA activity?

### Activities performed by an independent SQA group:

-  **Prepares an SQA plan for a project.**
  - Developed during project planning and is reviewed by all interested parties.
  - The plan identifies
    -  evaluations to be performed
    -  audits and reviews to be performed

- standards that are applicable to the project
- procedures for error reporting and tracking
- documents to be produced by the SQA group
- amount of feedback provided to the software project team

- **Participates in the development of the project's software process description.**

- **Reviews software engineering activities to verify compliance with the defined software process.**

- identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

- **Audits designated software work products to verify compliance with those defined as part of the software process.**

- reviews selected work products;
  - identifies, documents, and tracks deviations;
  - verifies that corrections have been made;
  - periodically reports the results of its work to the project manager.

- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure.**

- **Records any noncompliance and reports to senior management.**

- **Coordinates the control and management of change**

## SOFTWARE REVIEWS

**Software reviews** - applied at various points during software development and serve to uncover errors and defects that can then be removed.

- **Reasons:**

- People make mistakes.
  - Need technical reviews to catch a large classes of errors that escape the originator more easily than they escape anyone else.

- **A review is a way of using the diversity of a group of people to:**

- Point out needed improvements in the product of a single person or team;
  - Confirm those parts of a product in which improvement is either not desired or not needed;
  - Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

- ***Technical review - walkthrough or inspection***

- Most effective filter from a quality assurance standpoint.
  - Conducted by software engineers, effective means for improving software quality.

## Defect (IEEE Standard 610.12-1990)

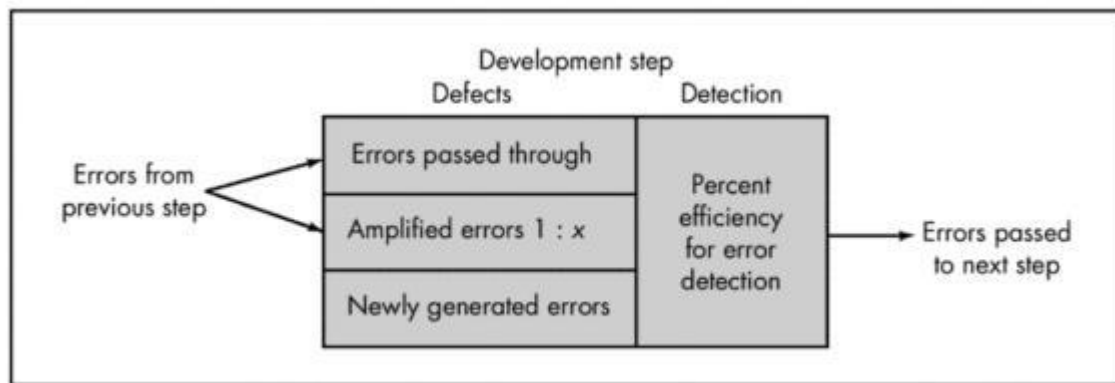
1. A defect in a hardware device or component; for example, a short circuit or broken wire.
2. An incorrect step, process, or data definition in a computer program.

## Primary objective of formal technical reviews

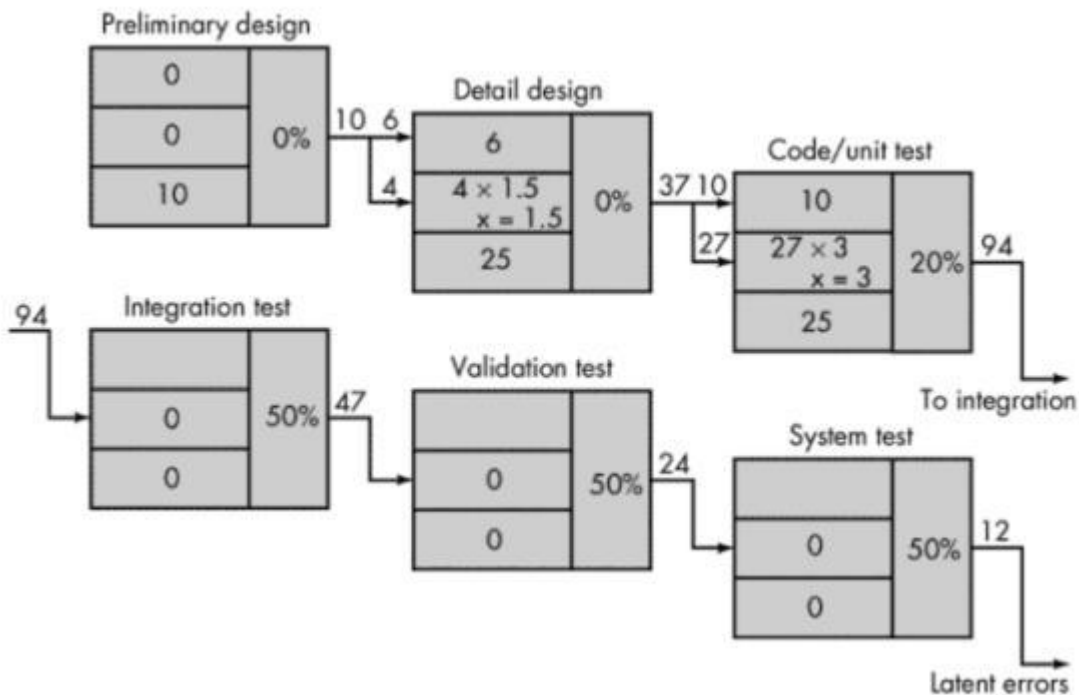
1. Find errors during the process so that they do not become defects after release of the software.
2. Benefit - the early discovery of errors so they do not propagate to the next step in the software process.

## Defect Amplification and Removal

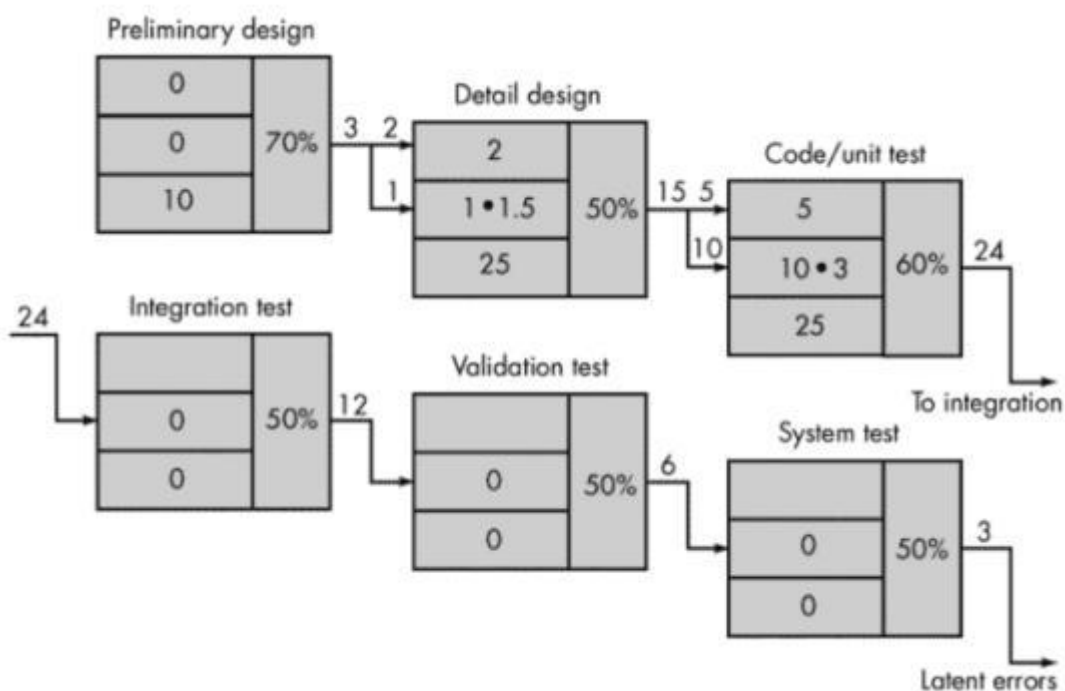
- Defect amplification model:
  - used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process.



- A box represents a software development step.
  - During the step, errors may be inadvertently generated.
  - Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through.
  - In some cases, errors passed through from previous steps are amplified (amplification factor,  $x$ ) by current work.
- box subdivisions represent each of these characteristics and the percent of efficiency for detecting errors, a function of the thoroughness of the review.



- ❖ **Hypothetical example** of defect amplification for a software development process in which no reviews are conducted.
- ❖ Each test step uncovers and corrects 50 percent of all incoming errors without introducing any new errors.
- ❖ Ten preliminary design defects are amplified to 94 errors before testing commences.
- ❖ Twelve latent errors are released to the field.



- ⌚ **Same conditions** except that design and code reviews are conducted as part of each development step.
- ⌚ Ten initial preliminary design errors are amplified to 24 errors before testing commences.
- ⌚ Only three latent errors exist.

## FORMAL TECHNICAL REVIEWS (FTR)

### Objectives of the FTR –

1. to uncover errors in function, logic, or implementation for any representation of the software
2. to verify that the software under review meets its requirements
3. to ensure that the software has been represented according to predefined standards
4. to achieve software that is developed in a uniform manner;
5. to make projects more manageable.

## FTR

- 🎬 serves as a training ground
- 🎬 promotes backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen.
- 🎬 class of reviews that includes:
  - walkthroughs
  - inspections
  - round-robin reviews
  - other small group technical assessments of software.
- 🎬 Each FTR is conducted as a meeting that is planned, controlled, and attended.

### The Review Meeting

#### Review meeting constraints:

- ⌚ Between three and five people should be involved in the review.
- ⌚ Advance preparation should require no more than two hours of work for each person.
- ⌚ The duration of the review meeting should be less than two hours.

NOTE: FTR focuses on a specific (and small) part of the overall software.

- 🎬 Walkthroughs are conducted for each component or small group of components.
- 🎬 FTR focuses on a work product (e.g., a portion of a requirements specification, a detailed component design, a source code listing for a component).
- 🎬 Individual who has developed the work product informs the project leader that the work product is complete and that a review is required.

■ The project leader contacts a *review leader*, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation.

■ Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.

■ Review meeting attended by

■ review leader

■ all reviewers

■ the producer.

■ One reviewer takes on the role of the *recorder*; that is, the individual who records (in writing) all important issues raised during the review.

1. The FTR begins with an introduction of the agenda and a brief introduction by the producer.
2. The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation.
3. When valid problems or errors are discovered, the recorder notes each.
4. At the end of the review, all attendees of the FTR must decide whether to
  1. accept the product without further modification
  2. reject the product due to severe errors (once corrected, another review must be performed)
  3. accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

■ The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings

### Review Reporting and Record Keeping

■ All issues that have been raised are summarized at the end of the review meeting and a **review issues list** is produced.

■ In addition, a formal technical **review summary report** is completed.

#### ■ **Review summary report**

■ answers three questions:

■ What was reviewed?

■ Who reviewed it?

■ What were the findings and conclusions?

■ single page form (with possible attachments).

- Becomes part of the project historical record and may be distributed to the project leader and other interested parties.

### ■ **Review issues list**

- serves two purposes:
  1. identify problem areas within the product
  2. serve as an action item checklist that guides the producer as corrections are made.
- attached to the summary report.

### **Review Guidelines**

Minimum set of guidelines for formal technical reviews:

1. ***Review the product, not the producer.***
2. ***Set an agenda and maintain it.***
3. ***Limit debate and rebuttal.***
4. ***Enunciate problem areas, but don't attempt to solve every problem noted.***
5. ***Take written notes.***
6. ***Limit the number of participants and insist upon advance preparation.***
7. ***Develop a checklist for each product that is likely to be reviewed.***
8. ***Allocate resources and schedule time for FTRs.***
9. ***Conduct meaningful training for all reviewers.***
10. ***Review your early reviews.***

## **STATISTICAL SOFTWARE QUALITY ASSURANCE**

Statistical quality assurance implies the following steps:

1. Information about software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause (e.g., non-conformance to specifications, design error, violation of standards, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").
4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.

### **Example:**

- ⌘ Assume that a software engineering organization collects information on defects for a period of one year.
- ⌘ Some of the defects are uncovered as software is being developed. \
- ⌘ Others are encountered after the software has been released to its end-users.
- ⌘ Although hundreds of different errors are uncovered, all can be tracked to one (or more) of the following causes:
  - ⌘ incomplete or erroneous specifications (IES)
  - ⌘ misinterpretation of customer communication (MCC)
  - ⌘ intentional deviation from specifications (IDS)
  - ⌘ violation of programming standards (VPS)



- ❖ error in data representation (EDR)
- ❖ inconsistent component interface (ICI)
- ❖ error in design logic (EDL)
- ❖ incomplete or erroneous testing (IET)
- ❖ inaccurate or incomplete documentation (IID)
- ❖ error in programming language translation of design (PLT)
- ❖ ambiguous or inconsistent human/computer interface (HCI)
- ❖ miscellaneous (MIS)

Error	Total		Serious		Moderate		Minor	
	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

To apply statistical SQA, a table is built.

- ❖ Table indicates that IES, MCC, and EDR are the *vital few* causes that account for 53 percent of all errors.
- ❖ Once the vital few causes are determined, the software engineering organization can begin corrective action.
- ❖ Statistical quality assurance techniques for software have been shown to provide substantial quality improvement up to 50 percent reduction per year in defects after applying these techniques.

## SOFTWARE RELIABILITY

- Software reliability can be measured, directed and estimated using historical and developmental data.
- *Software reliability* is defined in statistical terms as – the probability of failure-free operation of a computer program in a specified environment for a specified time

## Measures of Reliability and Availability

- Simple measure of reliability - *meantime-between-failure* (MTBF)

$$MTBF = MTTF + MTTR$$

- MTTF - mean-time-to-failure
- MTTR - mean-time-to-repair

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

$$\text{Availability} = [\text{MTTF}/(\text{MTTF} + \text{MTTR})] \times 100\%$$

- MTBF reliability measure is equally sensitive to MTTF and MTTR.
- Availability measure is more sensitive to MTTR, an indirect measure of the maintainability of software.

### Software Safety

- *Software safety* - software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- Modeling and analysis process is conducted as part of software safety.
  - Initially, hazards are identified and categorized by criticality and risk.
  - E.g., some of the hazards associated with a computer-based cruise control for an automobile :
    - 📖 causes uncontrolled acceleration that cannot be stopped
    - 📖 does not respond to depression of brake pedal (by turning off)
    - 📖 does not engage when switch is activated
    - 📖 slowly loses or gains speed
  - Once identified, analysis techniques are used to assign severity and probability of occurrence
  - Once hazards are identified and analyzed, safety-related requirements can be specified for the software.
    - 📖 Specification can contain a list of undesirable events and the desired system responses to these events.
  - The role of software in managing undesirable events is then indicated

### THE ISO 9000 QUALITY STANDARDS

- 📖 *Quality assurance system* - defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.

- 📖 Created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.
- 📖 Cover wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process.
- 📖 ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.
- 📖 The ISO 9000 standards adopted by many countries including all members of the European Community, Canada, Mexico, the United States, Australia, New Zealand, and the Pacific Rim.
- 📖 After adopting the standards, a country typically permits only ISO registered companies to supply goods and services to government agencies and public utilities.
- 📖 To become registered to one of the quality assurance system models contained in ISO 9000, a company's quality system and operations are scrutinized by third party auditors for compliance to the standard and for effective operation.
- 📖 Upon successful registration, a company is issued a certificate from a registration body represented by the auditors. Semi-annual surveillance audits ensure continued compliance to the standard.

### **The ISO Approach to Quality Assurance Systems**

- 📖 ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes.
  - 📖 ISO 9000 describes the elements of a quality assurance system in general terms.
  - 📖 Elements needed to implement:
    - quality planning
    - quality control
    - quality assurance
    - quality improvement
- are
- organizational structure
  - procedures
  - processes
  - resources

- 📖 ISO 9000 does not describe how an organization should implement quality system elements.

### **ISO 9001 Standard**

- 📖 ISO 9001 is the quality assurance standard that applies to software engineering.
- 📖 Contains 20 requirements that must be present for an effective quality assurance system.
- 📖 Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guidelines (ISO 9000-3) have been developed to help interpret the standard for use in the software process.
- 📖 Requirements delineated by ISO 9001 address topics such as:
  - management responsibility
  - quality system
  - contract review
  - design control
  - document and data control
  - product identification and traceability,
  - process control
  - inspection and testing
  - corrective and preventive action
  - control of quality records
  - internal quality audits, training
  - servicing
  - statistical techniques.
- 📖 For a software organization to become registered to ISO 9001, it must establish policies and procedures to address each of the requirements just noted (and others) and then be able to demonstrate that these policies and procedures are being followed.

## THE SQA PLAN

- 📖 **SQA Plan** provides a road map for instituting software quality assurance.
- 📖 Serves as a template for SQA activities that are instituted for each software project.
- 📖 Standard for SQA plans has been recommended by the IEEE
  - 📖 Standards, practices, and conventions section
    - lists all applicable standards and practices that are applied during the software process (e.g., document standards, coding standards, and review guidelines).
    - all project, process, and (in some instances) product metrics that are to be collected as part of software engineering work are listed.
  - 📖 Reviews and audits section

- identifies the reviews and audits to be conducted by the software engineering team, the SQA group, and the customer.
- provides an overview of the approach for each review and audit.



Test section references the *Software Test Plan and Procedure*

- defines test record-keeping requirements.
- problem reporting and corrective action defines procedures for reporting, tracking, and resolving errors and defects, and identifies the organizational responsibilities for these activities.



Tools and methods section

- Tools that support SQA activities and tasks
- software configuration management procedures for controlling change
- contract management approach
- establishes methods for assembling, safeguarding, and maintaining all records; identifies training required to meet the needs of the plan;
- defines methods for identifying, assessing, monitoring, and controlling risk

## Chapter 27: Change Management

### SOFTWARE CONFIGURATION MANAGEMENT



Output of the software process:

1. computer programs (both source level and executable forms)
2. documents that describe the computer programs (targeted at both technical practitioners and users)
3. data (contained within the program or external to it). The items that comprise all information produced as part of the software process are collectively called a *software configuration*.



Number of *software configuration items* (SCIs) grows rapidly as software process progresses.



Change may occur at any time, for any reason.



First Law of System Engineering:

"No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle."



Four sources of change:


1. New business or market conditions dictate changes in product requirements or business rules.


2. New customer needs demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
3. Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.
4. Budgetary or scheduling constraints cause a redefinition of the system or product.

#### **Software configuration management:**


- Set of activities that have been developed to manage change throughout the life cycle of computer software.
- Viewed as a software quality assurance activity that is applied throughout the software process.

#### **Baselines**

 Software configuration management concept that helps control change without seriously impeding justifiable change.


 IEEE Std. No. 610.12-1990 defines a baseline as:

A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

 Software engineering context:

A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review.

 After SCIs are reviewed and approved, they are placed in a *project database* (also called a *project library* or *software repository*).

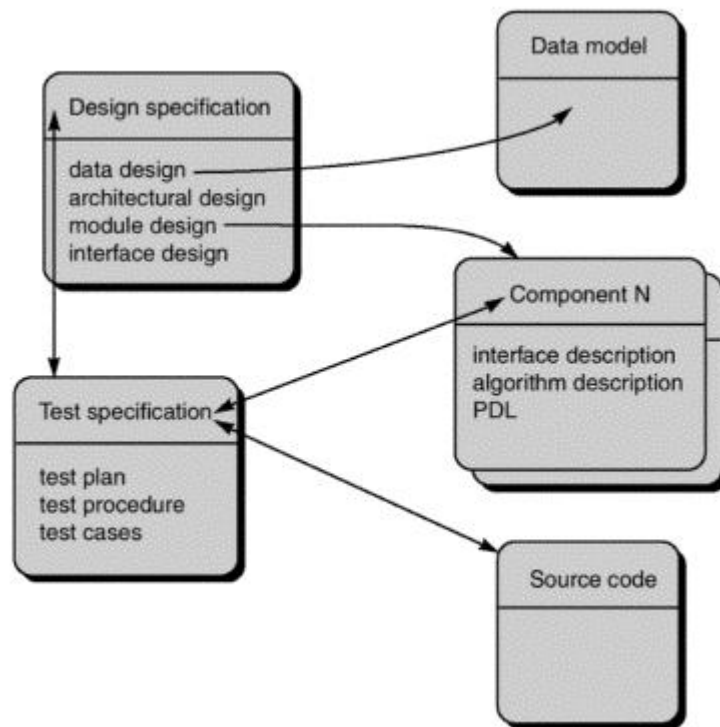
 When a member of a software engineering team wants to make a modification to a baselined SCI:

- it is copied from the project database into the engineer's private work space.
- this extracted SCI can be modified only if SCM controls are followed

#### **Software Configuration Items**

- A SCI is a document, an entire suite of test cases, or a named program component (e.g., a C++ function or Java class).
- SCIs are organized to form *configuration objects*
  - cataloged in the project database with a single name.


- Has a name, attributes, and is "connected" to other objects by relationships.




e.g.

- Configuration objects, **Design Specification**, **data model**, **component N**, **source code** and **Test Specification** are each defined separately.
- Each of object is related to others by the arrows.

 Curved arrow indicates a *compositional relation*.

 **data model** and **component N** are part of the object **Design Specification**.

 Double-headed straight arrow indicates an interrelationship.

## THE SCM PROCESS

SCM introduces a set of complex questions:

- ❖ How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- ❖ How does an organization control changes before and after software is released to a customer?
- ❖ Who has responsibility for approving and ranking changes?
- ❖ How can we ensure that changes have been made properly?
- ❖ What mechanism is used to appraise others of changes that are made?

Must define five SCM tasks:

- ✧ *Identification*
- ✧ *version control*
- ✧ *change control*
- ✧ *configuration auditing*
- ✧ *reporting*

## IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

Two types of objects can be identified:

- ✧ *basic objects*  
A "unit of text" that has been created by a software engineer during analysis, design, code, or test.
- ✧ *aggregate objects*  
A collection of basic objects and other aggregate objects.

Each object has a set of distinct features that identify it uniquely:

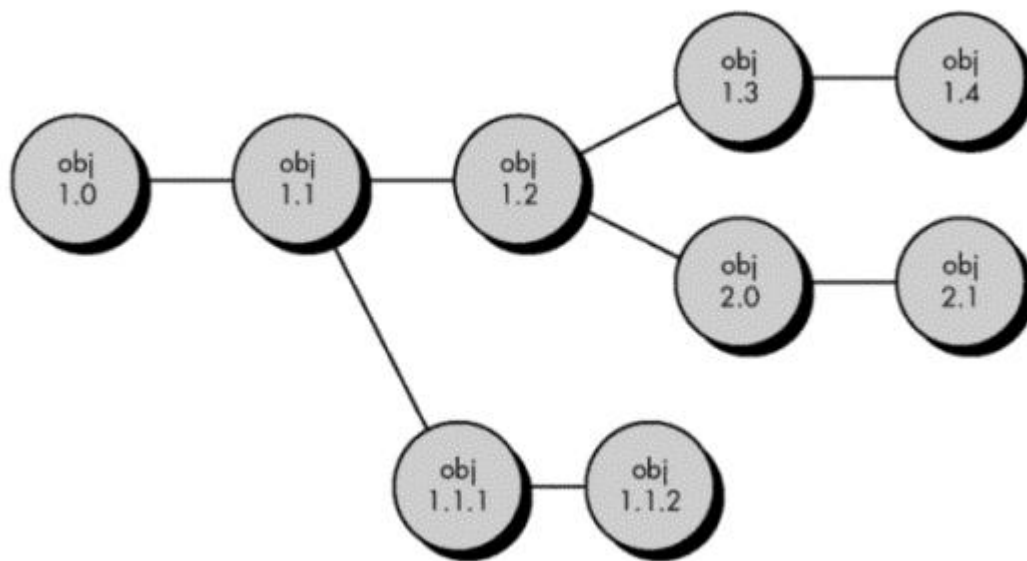
- ✧ **Name**  
a character string that identifies the object unambiguously
- ✧ **Description**  
list of data items that identify
  - the SCI type (e.g., document, program, data) represented by the object
  - project identifier
  - change and/or version information
- ✧ **list of resources**  
entities that are provided, processed, referenced or otherwise required by the object
- ✧ **"realization."**  
a pointer to the "unit of text" for a basic object and null for an aggregate object



Identification scheme must recognize that objects evolve throughout the software process.

- Before an object is baselined, it may change many times
- After a baseline has been established, changes may be quite frequent.
- Possible to create an *evolution graph*.
- Describes the change history of an object





- 📖 Configuration object 1.0 undergoes revision and becomes object 1.1.
- 📖 Minor corrections and changes result in versions 1.1.1 and 1.1.2
- 📖 major update that is object 1.2.

## VERSION CONTROL

🎬 *Version control* combines procedures and tools to manage different versions of configuration objects that are created during the software process.

## CHANGE CONTROL

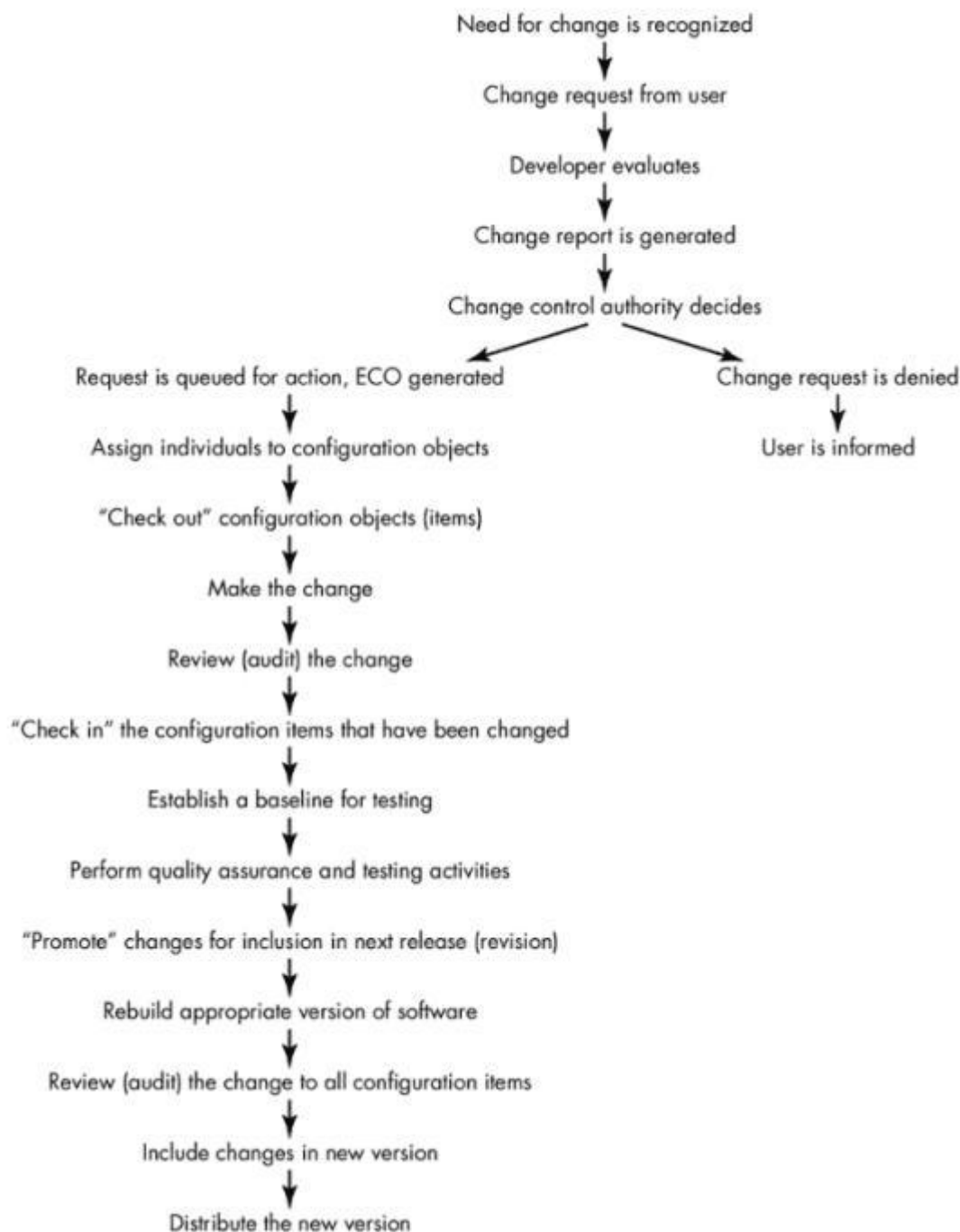
🎬 *Change control* combines human procedures and automated tools to provide a mechanism for the control of change.

🎬 *Change Control Process:*

- A *change request* is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.
- The results of the evaluation are presented as a *change report*, which is used by a *change control authority* (CCA) — a person or group who makes a final decision on the status and priority of the change.
- An *engineering change order* (ECO) is generated for each approved change.

The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit.

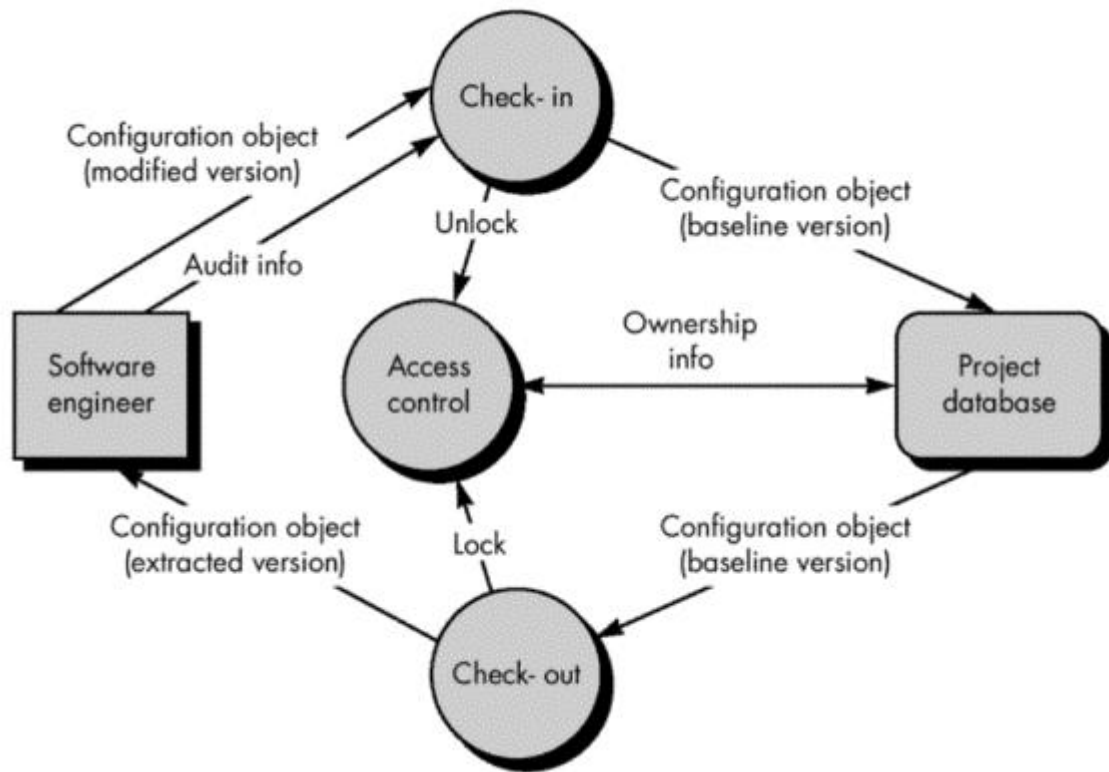
- The object to be changed is "checked out" of the project database, the change is made, and appropriate SQA activities are applied.
- The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.



NOTE:

■ "check-in" and "check-out" process implements two elements of change control

- *Access control* - governs which software engineers have the authority to access and modify a particular configuration object.
- *Synchronization control* - helps to ensure that parallel changes, performed by two different people, don't overwrite one another



## CONFIGURATION AUDIT

How to ensure that the change has been properly implemented:

- ✎ formal technical reviews
- ✎ the software configuration audit.

The *formal technical review* focuses on the technical correctness of the configuration object that has been modified.

A *software configuration audit* complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review.



The audit asks and answers the following questions:






1. Has the change specified in the ECO been made? Have any additional modifications been incorporated?
2. Has a formal technical review been conducted to assess technical correctness?
3. Has the software process been followed and have software engineering standards been properly applied?

4. Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
5. Have SCM procedures for noting the change, recording it, and reporting it been followed?
6. Have all related SCIs been properly updated?

## STATUS REPORTING

*Configuration status reporting* is an SCM task that answers the following questions:

- (1) What happened?
- (2) Who did it?
- (3) When did it happen?
- (4) What else will be affected?

-  Each time an SCI is assigned new or updated identification, a CSR entry is made.
-  Each time a change is approved by the CCA (i.e., an ECO is issued), a CSR entry is made.
-  Each time a configuration audit is conducted, the results are reported as part of the CSR task.
-  Output from CSR may be placed in an on-line database so that software developers or maintainers can access change information by keyword category.
-  CSR report is generated on a regular basis and is intended to keep management and practitioners apprised of important changes.