Week 8

Verification and Validation

Verification and Validation

- Assuring that a software system meets a user's needs

Objectives

- To introduce software verification and validation and to discuss the distinction between them
- To describe the program inspection process and its role in V & V
- To explain static analysis as a verification technique
- To describe the Cleanroom software development process

Topics covered

- Verification and validation planning
- Software inspections
- Automated static analysis
- Cleanroom software development

Verification vs validation

- Verification:

  "Are we building the product right"

  - The software should conform to its specification
- Validation:

  "Are we building the right product"

  - The software should do what the user really requires

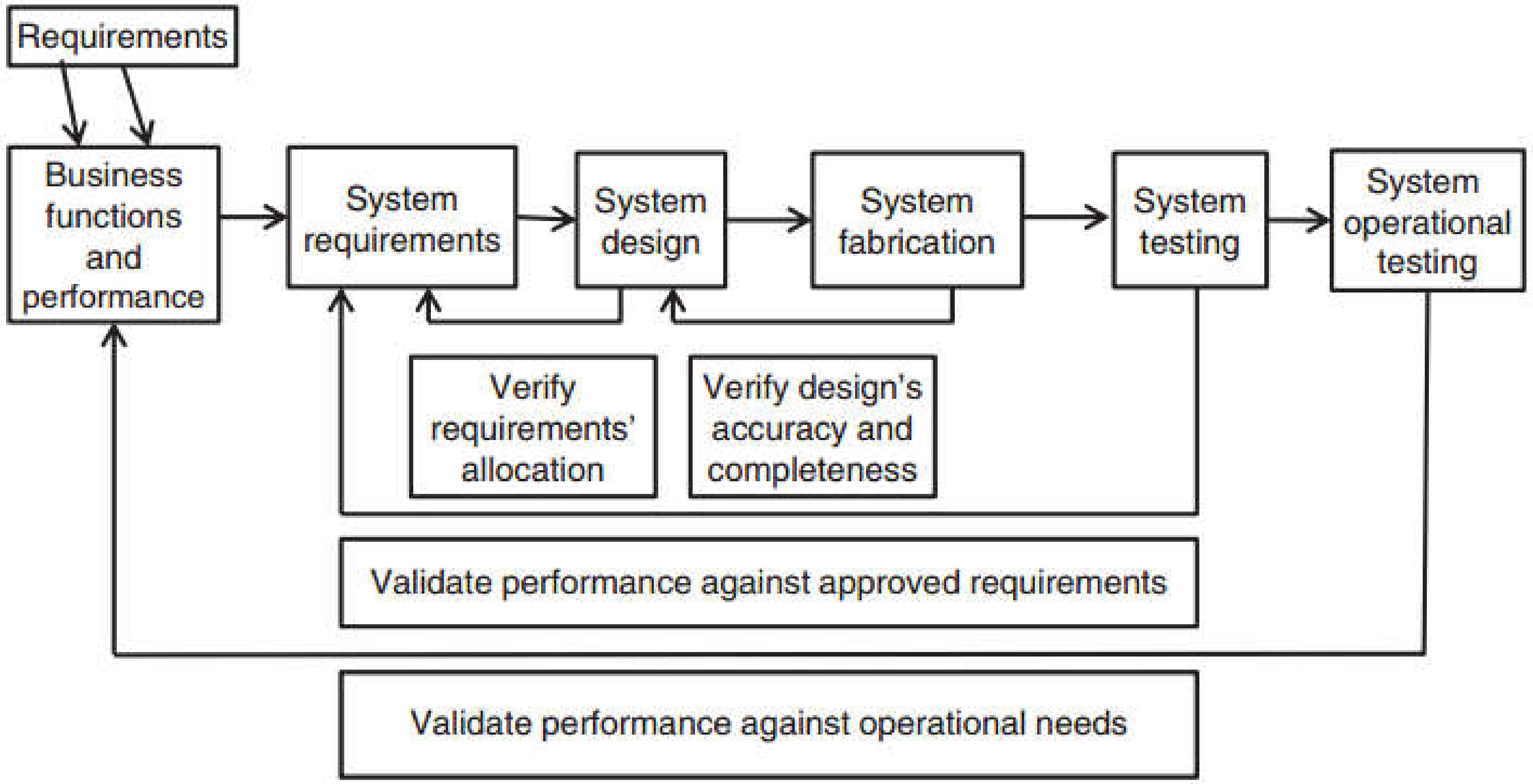
ISO 9000 Definition

- Verification
  - Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.
- Validation
  - Confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled.
  - Note 1 to entry: The objective evidence needed for a validation is the result of a test or other form of determination such as performing alternative calculations or reviewing documents.
  - Note 2 to entry: The word “validated” is used to designate the corresponding status.
  - Note 3 to entry: The use conditions for validation can be real or simulated

The V & V process

- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
  - The discovery of defects in a system
  - The assessment of whether or not the system is usable in an operational situation.

V&V activities in the software development life cycle


Requirements
Business functions and performance
System requirements
System design
System fabrication
System testing
System operational testing
Verify requirements' allocation
Verify design's accuracy and completeness
Validate performance against approved requirements
Validate performance against operational needs

IEEE 1012 V&V Standard

- The IEEE 1012—Standard for System and Software Verification and Validation.
- Theassessment can include analysis, evaluation, reviews, inspections, and testing of the products and the development activity.

Purpose of IEEE 1012

- The intention of this standard is to perform the following:
  - establish a common framework for all the V&V processes, activities, and tasks in support of the system, software, and hardware life cycle processes;
  - define the V&V tasks, required inputs, and required outputs in each life cycle process;
  - identify the minimum V&V tasks corresponding to a four-level integrity scheme;
  - define the content of the V&V Plan.

V& V goals

- Verification and validation should establish confidence that the software is fit for purpose
- This does NOT mean completely free of defects
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed

V & V confidence

- Depends on system's purpose, user expectations and marketing environment
  - Software function
    - » The level of confidence depends on how critical the software is to an organisation
  - User expectations
    - » Users may have low expectations of certain kinds of software
  - Marketing environment
    - » Getting a product to market early may be more important than finding defects in the program

Verification in IEEE 1012

- The verification process provides objective evidence that the system, software, or hardware and its associated products:
  - conform to requirements (e.g., for correctness, completeness, consistency, and accuracy) for all life cycle activities during each life cycle process (acquisition, supply, development, operation, and maintenance)
  - satisfy standards, practices, and conventions during life cycle processes;
  - successfully complete each life cycle activity and satisfy all the criteria for initiating succeeding life cycle activities.

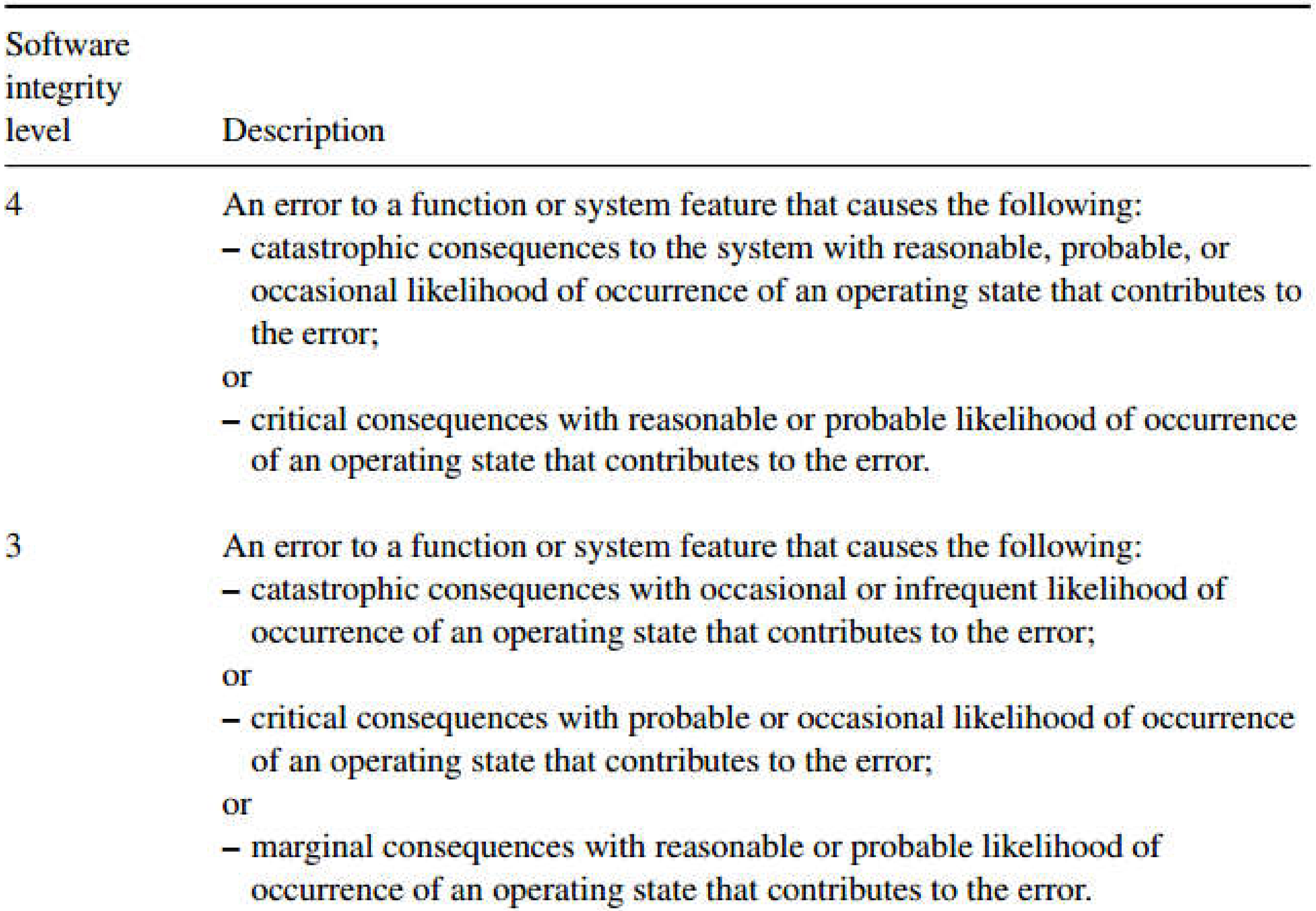
Validation in IEEE 1012

- The validation process provides evidence that the system, software, or hardware and its associated products:
  - satisfy requirements allocated to it at the end of each life cycle activity;
  - solve the right problem (e.g., correctly model physical laws, implement business rules, and use the proper system assumptions);
  - satisfy intended use and user needs.

IEEE 1012 Integrity Levels

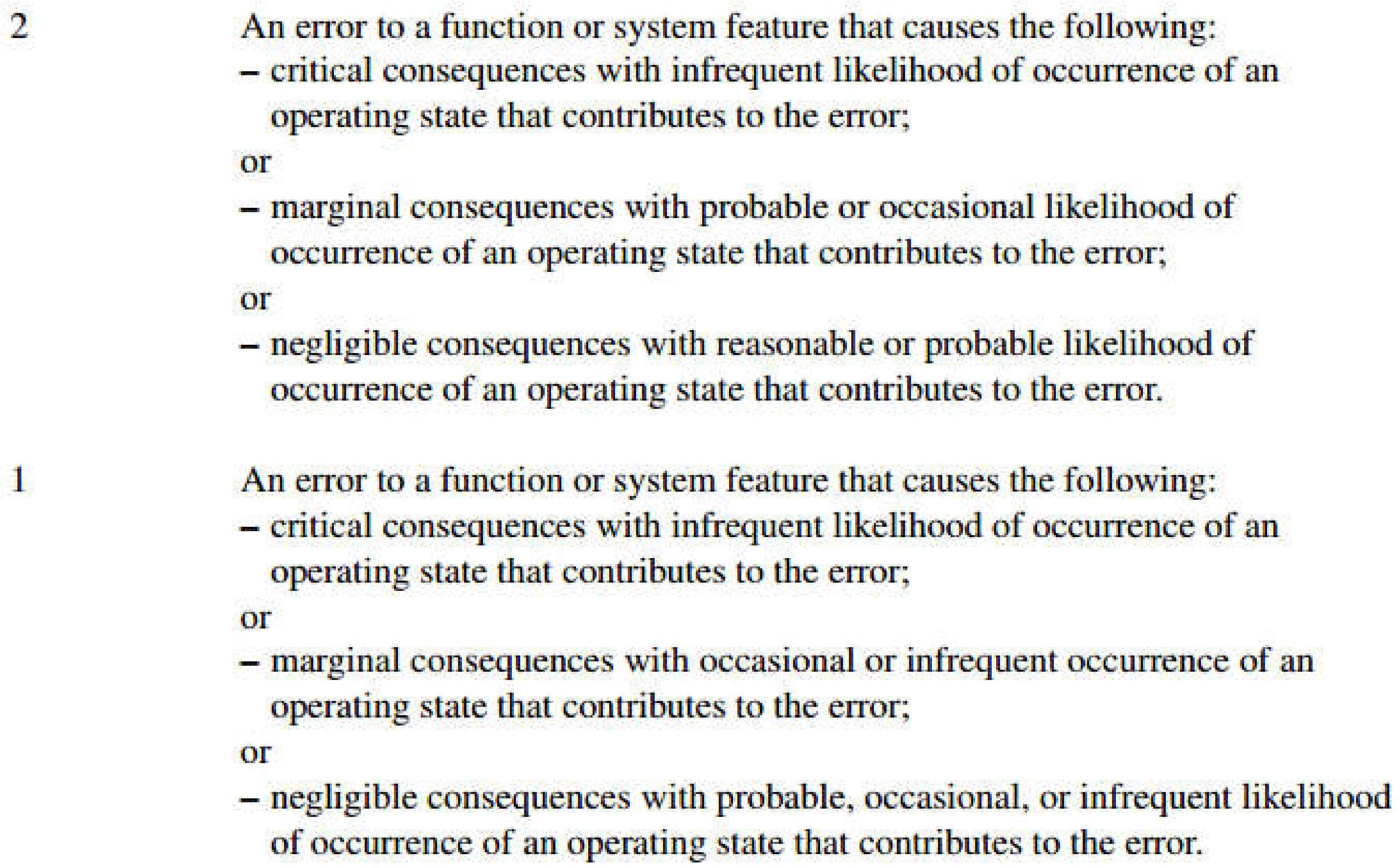
- Integrity Levels: A value representing project-unique characteristics (e.g., complexity, criticality, risk, safety level, security level, desired performance, and reliability) that defines the importance of the system, software, or hardware to the user.
- IEEE 1012 uses integrity levels to identify V&V tasks that should be executed depending on the risk.
- High integrity level system and software require more emphasis on V&V processes as well as a more rigorous execution of the V&V tasks in the project.

Integrity Levels and the Description of Consequences based on IEEE 1012



Software integrity level	Description
4	An error to a function or system feature that causes the following: – catastrophic consequences to the system with reasonable, probable, or occasional likelihood of occurrence of an operating state that contributes to the error; or – critical consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.
3	An error to a function or system feature that causes the following: – catastrophic consequences with occasional or infrequent likelihood of occurrence of an operating state that contributes to the error; or – critical consequences with probable or occasional likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.

Integrity Levels and the Description of Consequences based on IEEE 1012



2	An error to a function or system feature that causes the following: – critical consequences with infrequent likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with probable or occasional likelihood of occurrence of an operating state that contributes to the error; or – negligible consequences with reasonable or probable likelihood of occurrence of an operating state that contributes to the error.
1	An error to a function or system feature that causes the following: – critical consequences with infrequent likelihood of occurrence of an operating state that contributes to the error; or – marginal consequences with occasional or infrequent occurrence of an operating state that contributes to the error; or – negligible consequences with probable, occasional, or infrequent likelihood of occurrence of an operating state that contributes to the error.

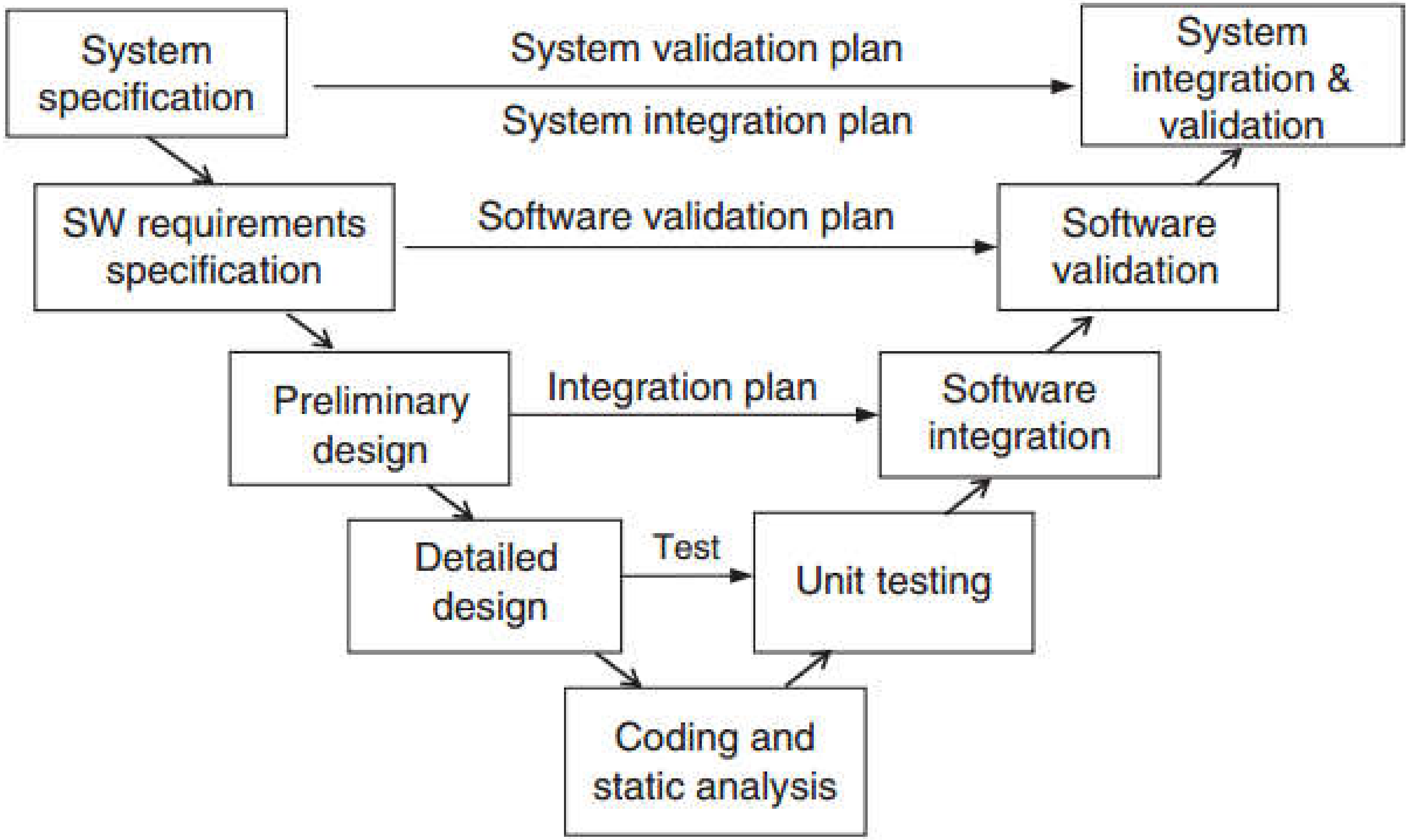
Another Model for V&V

- ISO/IEC/IEEE 12207
  - Presents the requirements for V&V processes.
- THE CMMI MODEL
- ISO/IEC 29110
  - Standard V&V for very small entities.

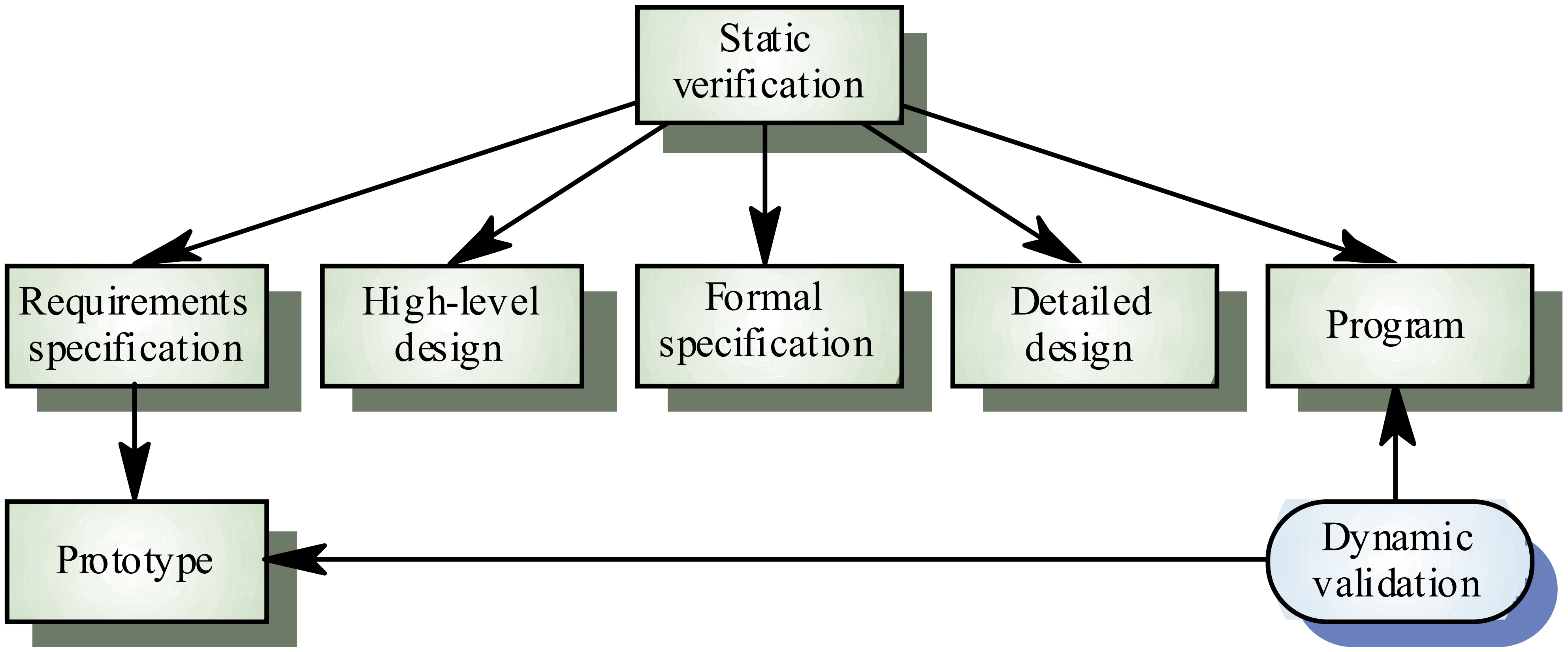
Static and dynamic verification

- *Software inspections* Concerned with analysis of the static system representation to discover problems *(*static verification)
  - May be supplement by tool-based document and code analysis
- *Software testing* Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed

V Development Life Cycle Process



Static and dynamic V&V



Software Traceability

- A simple V&V technique that ensures that all the user requirements have been:
  - documented in specifications;
  - developed and documented in the design document;
  - implemented in the source code;
  - tested;
  - delivered.

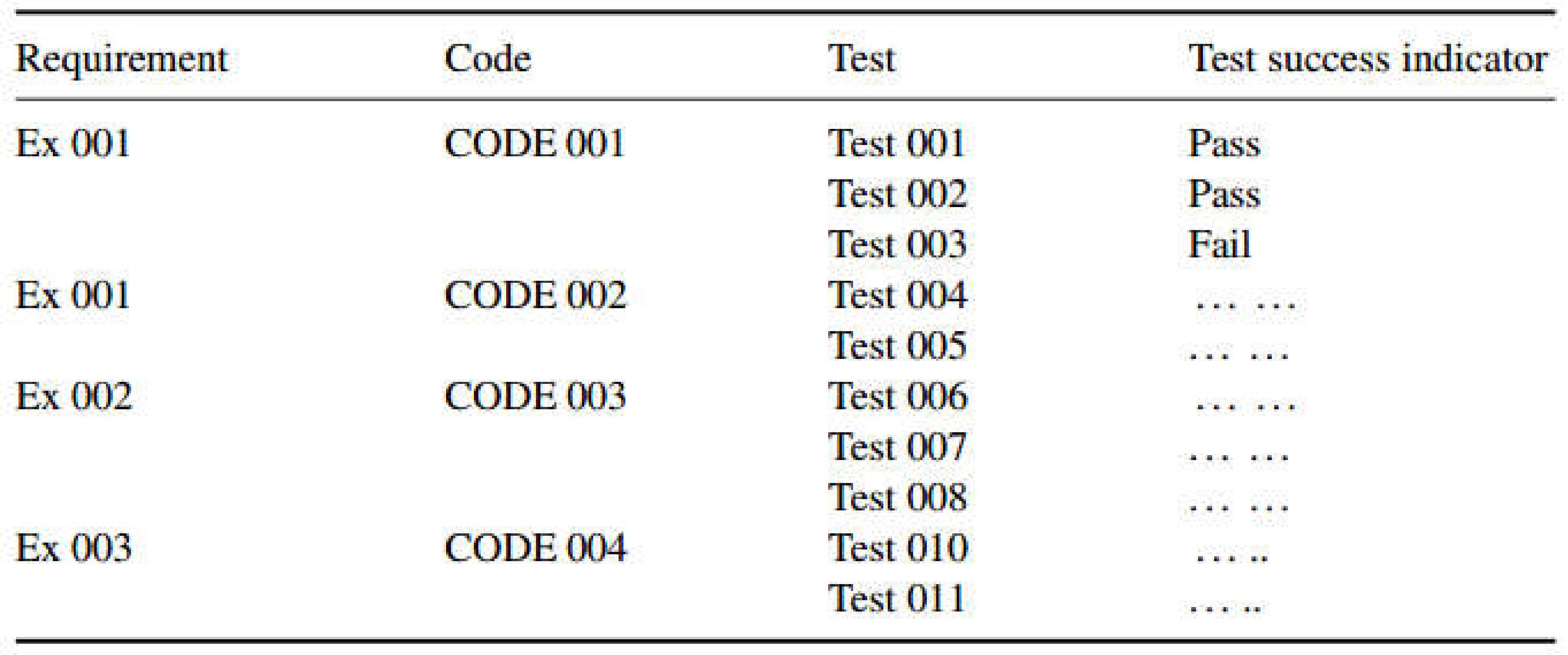
Traceability

- Traceability
  - Ability to trace the history, application or location of an object.
- Bidirectional Traceability
  - An association among two or more logical entities that is discernable in either direction (i.e., to and from an entity).
- Requirements Traceability
  - A discernable association between requirements and related requirements, implementations, and verifications.

Traceability Matrix

- A simple tool that can be developed to facilitate traceability.
- A software traceability matrix is a simple tool that can be developed to facilitate traceability.
- This matrix is completed at each phase of the development life cycle.
- It requires user requirements that have been well-defined, documented, and reviewed

Example of a Simple Traceability Matrix



Requirement	Code	Test	Test success indicator
Ex 001	CODE 001	Test 001	Pass
		Test 002	Pass
		Test 003	Fail
Ex 001	CODE 002	Test 004	… …
		Test 005	… …
Ex 002	CODE 003	Test 006	… …
		Test 007	… …
		Test 008	… …
Ex 003	CODE 004	Test 010	… ..
		Test 011	… ..

Testing

- Test
  - An activity in which a system or component is executed, under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system [ISO 24765].
- Development Testing
  - Formal or informal testing conducted during the development of a system or component, usually in the development environment by the developer [ISO 24765].
- Acceptance Testing
  - Testing conducted to determine whether a system satisfes its acceptance criteria and to enable the customer to determine whether to accept the system [IEEE 829].

Testing

- Qualifcation Testing
  - Testing, conducted by the developer and witnessed by the acquirer (as appropriate), to demonstrate that a software product meets its specifications and is ready for use in its target environment or integration with its containing system [ISO 12207]
- Operational Testing
  - Testing conducted to evaluate a system or component in its operational environment [IEEE 829]

Program testing

- Can reveal the presence of errors NOT their absence
- A successful test is a test which discovers one or more errors
- The only validation technique for non-functional requirements
- Should be used in conjunction with static verification to provide full V&V coverage

Types of testing

- Defect testing
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
  - Covered in Chapter 20
- Statistical testing
  - tests designed to reflect the frequence of user inputs. Used for reliability estimation.
  - Covered in Chapter 21

Some V&V Techniques

- Algorithms Analysis Technique
    - examines the logic and accuracy of the configuration of a software by the transcription of the algorithms in a structured language or format.
- Interface Analysis Technique
    - a technique used to demonstrate that program interfaces do not contain errors that can lead to failures.
    - include external interfaces to the software, internal interfaces between components, interfaces with hardware between software and system, between software and hardware, and between the software and a database

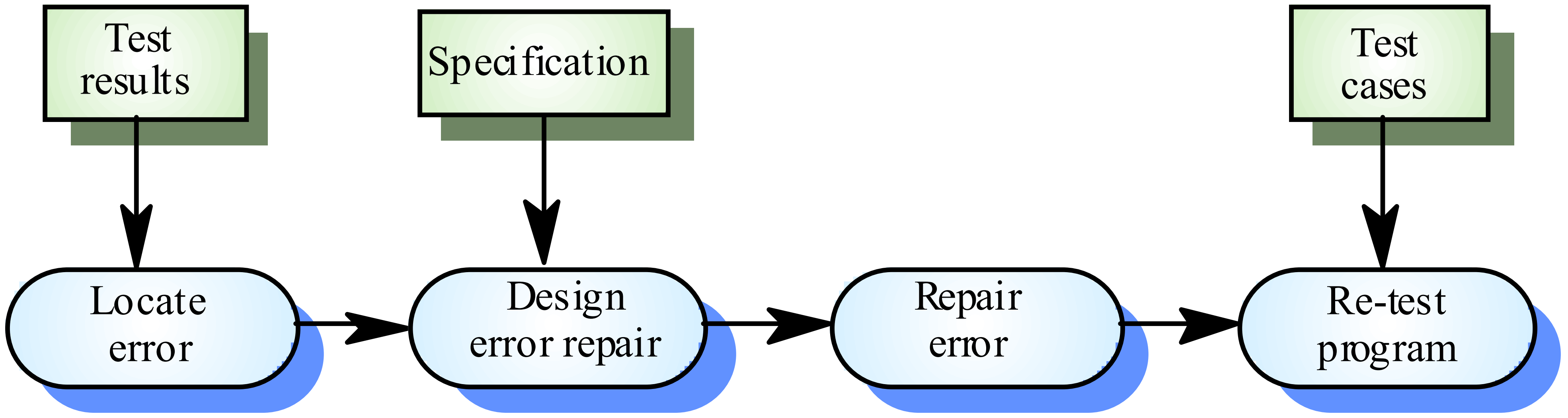
Some V&V Techniques

- Prototyping Technique
  - Prototyping demonstrates the likely results of implementing software requirements, especially the user interfaces.
  - Identify incomplete or incorrect software requirements and reveal whether the requirements will not result in undesirable system behavior.
- Simulation Technique
  - Used to evaluate the interactions between large complex systems composed of hardware, software, and users.
  - The simulation uses an “executable” model to examine the behavior of the software.

Testing and debugging

- Defect testing and debugging are distinct processes
- Verification and validation is concerned with establishing the existence of defects in a program
- Debugging is concerned with locating and repairing these errors
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error

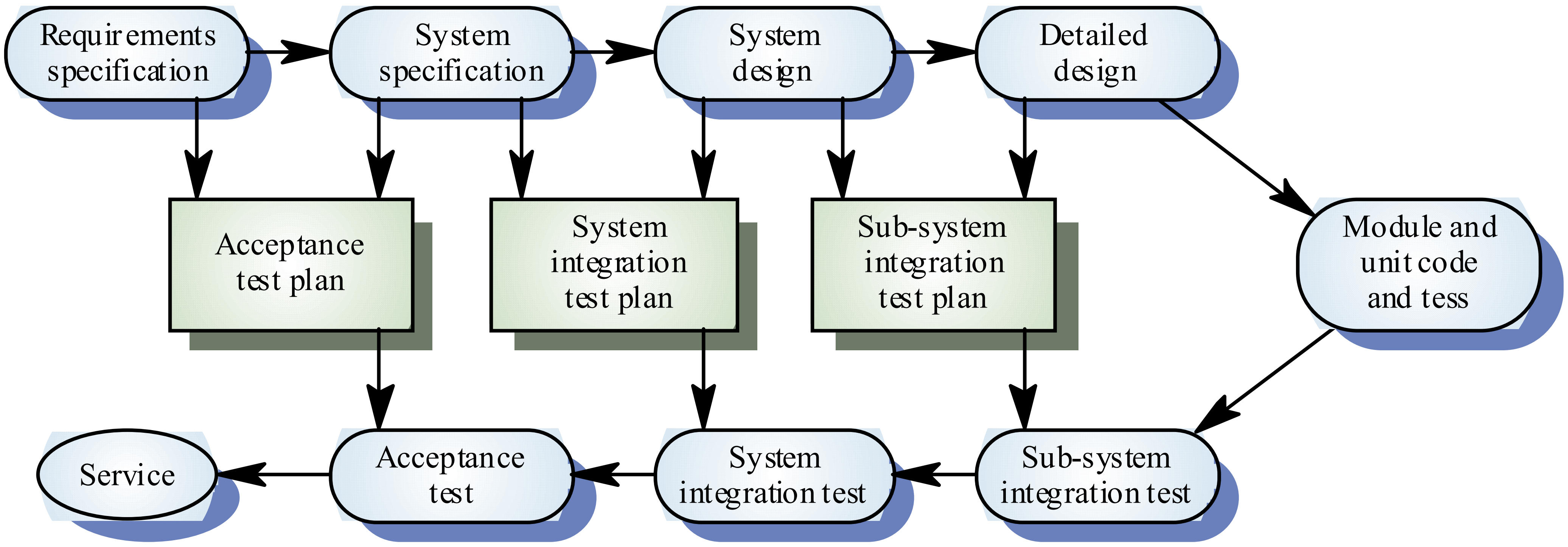
The debugging process



V & V planning

- Careful planning is required to get the most out of testing and inspection processes
- Planning should start early in the development process
- The plan should identify the balance between static verification and testing
- Test planning is about defining standards for the testing process rather than describing product tests

The V-model of development


Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and tess
Service
Acceptance test
System integration test
Sub-system integration test

The structure of a software test plan

- The testing process
- Requirements traceability
- Tested items
- Testing schedule
- Test recording procedures
- Hardware and software requirements
- Constraints

Software inspections

- Involve people examining the source representation with the aim of discovering anomalies and defects
- Do not require execution of a system so may be used before implementation
- May be applied to any representation of the system (requirements, design, test data, etc.)
- Very effective technique for discovering errors

Inspection success

- Many diffreent defects may be discovered in a single inspection. In testing, one defect ,may mask another so several executions are required
- The reuse domain and programming knowledge so reviewers are likely to have seen the types of error that commonly arise

Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques
- Both should be used during the V & V process
- Inspections can check conformance with a specification but not conformance with the customer’s real requirements
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

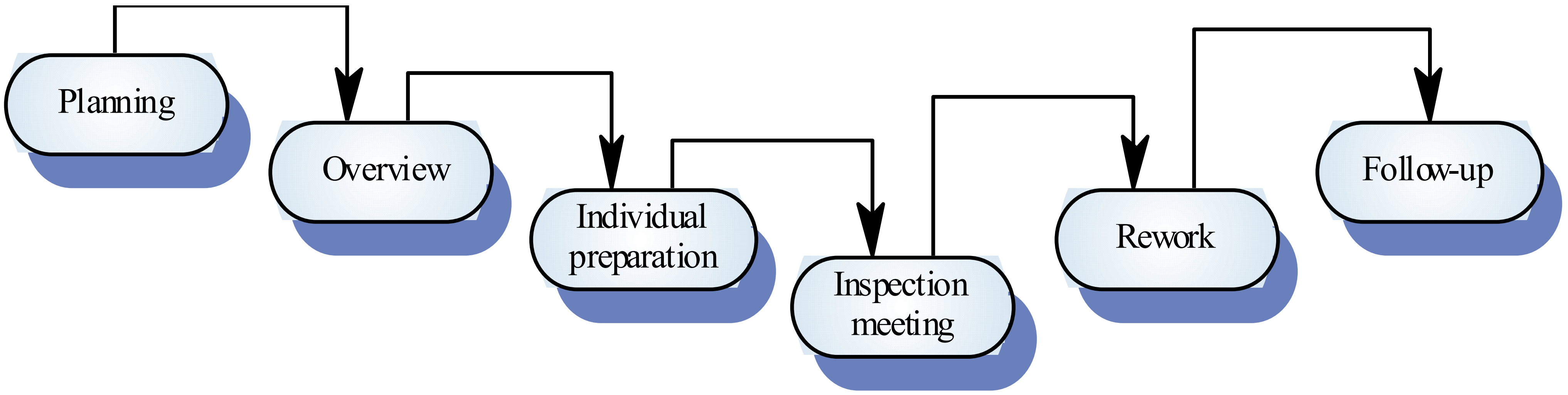
Program inspections

- Formalised approach to document reviews
- Intended explicitly for defect DETECTION (not correction)
- Defects may be logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an uninitialised variable) or non-compliance with standards

Inspection pre-conditions

- A precise specification must be available
- Team members must be familiar with the organisation standards
- Syntactically correct code must be available
- An error checklist should be prepared
- Management must accept that inspection will increase costs early in the software process
- Management must not use inspections for staff
  appraisal

The inspection process



Inspection procedure

- System overview presented to inspection team
- Code and associated documents are distributed to inspection team in advance
- Inspection takes place and discovered errors are noted
- Modifications are made to repair discovered errors
- Re-inspection may or may not be required

Inspection teams

- Made up of at least 4 members
- Author of the code being inspected
- Inspector who finds errors, omissions and inconsistencies
- Reader who reads the code to the team
- Moderator who chairs the meeting and notes discovered errors
- Other roles are Scribe and Chief moderator

Inspection checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type checking, the larger the checklist
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

Inspection checks

Fault class	**Inspection check**
Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the lower bound of arrays be 0, 1, or something else? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output?
Interface faults	Do all function and procedure calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

Inspection rate

- 500 statements/hour during overview
- 125 source statement/hour during individual preparation
- 90-125 statements/hour can be inspected
- Inspection is therefore an expensive process
- Inspecting 500 lines costs about 40 man/hours effort = £2800

Automated static analysis

- Static analysers are software tools for source text processing
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team
- Very effective as an aid to inspections. A supplement to but not a replacement for inspections

Static analysis checks

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Stages of static analysis

- *Control flow analysis.* Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- *Data use analysis.* Detects uninitialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- *Interface analysis.* Checks the consistency of routine and procedure declarations and their use

Stages of static analysis

- *Information flow analysis.* Identifies the dependencies of output variables. Does not detect anomalies itself but highlights information for code inspection or review
- *Path analysis.* Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process
- Both these stages generate vast amounts of information. Must be used with care.

```
138% more lint_ex.c

#include <stdio.h>
printarray (Anarray)
  int Anarray;
{
  printf(“%d”,Anarray);
}
main ()
{
  int Anarray[5]; int i; char c;
  printarray (Anarray, i, c);
  printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(11)
printf returns value which is always ignored
```

LINT static analysis

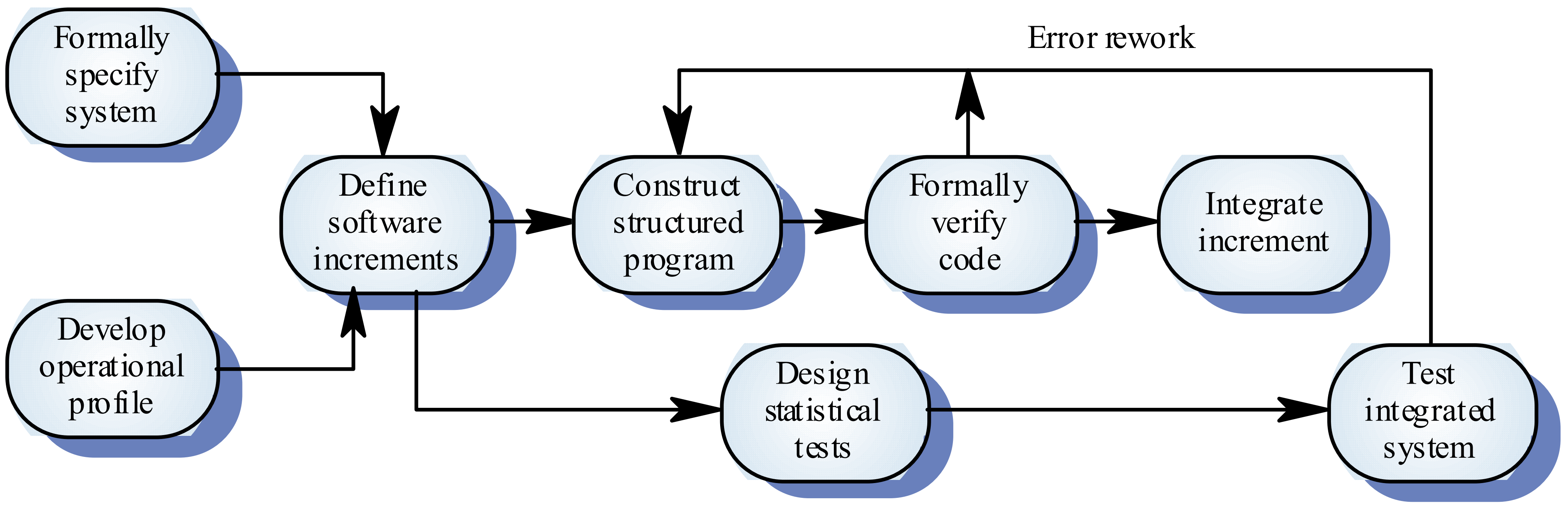
Use of static analysis

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler
- Less cost-effective for languages like Java that have strong type checking and can therefore detect many errors during compilation

Cleanroom software development

- The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal
- Software development process based on:
  - Incremental development
  - Formal specification.
  - Static verification using correctness arguments
  - Statistical testing to determine program reliability.

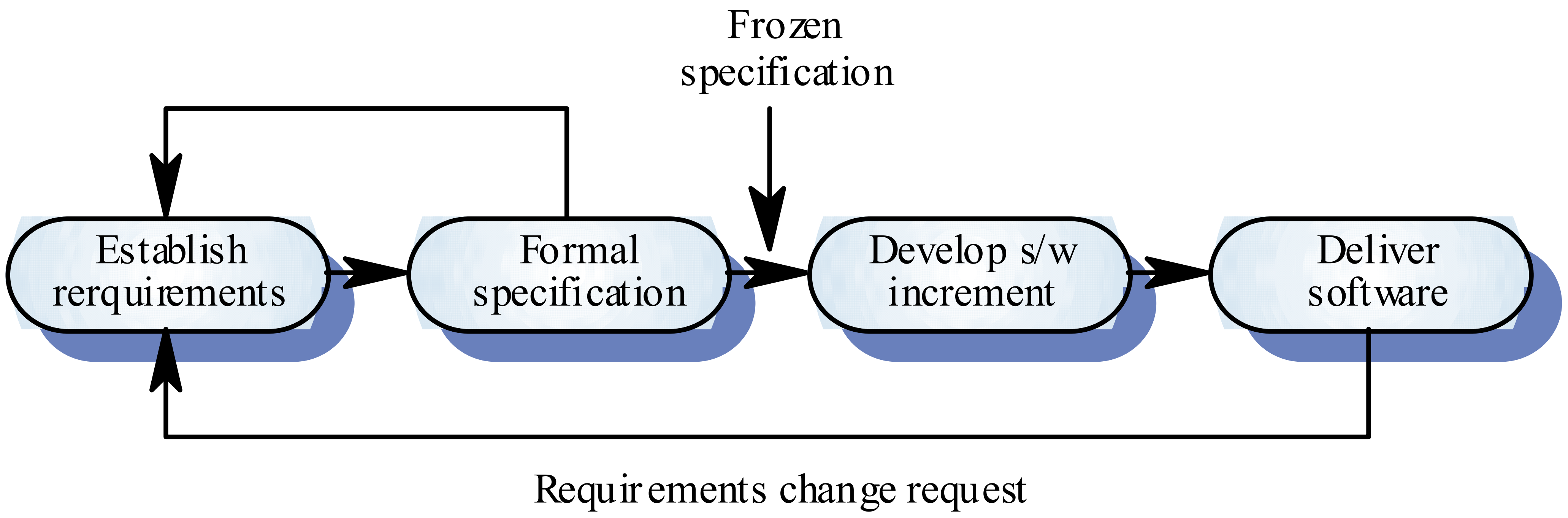
The Cleanroom process



Cleanroom process characteristics

- Formal specification using a state transition model
- Incremental development
- Structured programming - limited control and abstraction constructs are used
- Static verification using rigorous inspections
- Statistical testing of the system (covered in Ch. 21).

Incremental development



Formal specification and inspections

- The state based model is a system specification and the inspection process checks the program against this model
- Programming approach is defined so that the correspondence between the model and the system is clear
- Mathematical arguments (not proofs) are used to increase confidence in the inspection process

Cleanroom process teams

- *Specification team.* Responsible for developing and maintaining the system specification
- *Development team.* Responsible for developing and verifying the software. The software is NOT executed or even compiled during this process
- *Certification team.* Responsible for developing a set of statistical tests to exercise the software after development. Reliability growth models used to determine when reliability is acceptable

Cleanroom process evaluation

- Results in IBM have been very impressive with few discovered faults in delivered systems
- Independent assessment shows that the process is no more expensive than other approaches
- Fewer errors than in a 'traditional' development process
- Not clear how this approach can be transferred to an environment with less skilled or less highly motivated engineers

Key points

- Verification and validation are not the same thing. Verification shows conformance with specification; validation shows that the program meets the customer’s needs
- Test plans should be drawn up to guide the testing process.
- Static verification techniques involve examination and analysis of the program for error detection

Key points

- Program inspections are very effective in discovering errors
- Program code in inspections is checked by a small team to locate software faults
- Static analysis tools can discover program anomalies which may be an indication of faults in the code
- The Cleanroom development process depends on incremental development, static verification and statistical testing

Software Configuration Management

Pertemuan 9

Jaminan Kualitas Perangkat Lunak

Outline

- Software Configuration Management (SCM)
  - What is SCM
  - What is software configuration
  - Sources of Change
  - People in typical SCM Scenario
  - Baselines
  - Configuration objects
  - SCM Repository
  - SCM Process
  - Change Control
  - Version Control
  - CSR

What is SCM?

- Change management is commonly called *software configuration management* (SCM or CM)
- *Babich* defines Configuration Management as:
  - the art of identifying, organizing and controlling modifications to the software being built by a programming team.
- SCM activities have been developed to manage change throughout the life cycle of computer software
- SCM can be viewed as a quality assurance activity

What is SCM? (Contd.)

- SCM is a set of activities designed to manage change by:
  - Identifying s/w work products that are likely to change,
  - Establishing relationships among them,
  - Defining mechanisms for managing different versions of these work products
  - Controlling changes
  - And auditing and reporting on the changes made

Difference between Software Support and SCM

- Support is a set of s/w engg. activities that occur after software has been delivered to the customer and put into operation.

- While SCM is a set of tracking and control activities that initiate when the s/w engg project begins & end only when the s/w is taken out of operation

What is software configuration?

- The items that comprise all information produced as part of the software process are collectively called a ***software configuration***
- This information (also termed as output of software process) is broadly divided into:
  - Computer programs (both source level & executable forms)
  - Work products that describe the computer programs
  - Data (contained within program or external to it)

Sources of Changes

- **New business or market conditions**
  - dictate changes in product requirements or business rules.
- **New customer needs**
  - demand modification of data produced by information systems, functionality delivered by products, or services delivered by a computer-based system.
- **Reorganization or business growth/ downsizing**
  - causes changes in project priorities or software engg team structure.
- **Budgetary or scheduling constraints**
  - cause a redefinition of the system or product.

People involved in a typical SCM Scenario

- Project Manager
  - In charge of s/w group
  - Ensures that product is developed within time frame
  - Monitors development, recognizes & reacts to problems
    - by generating & analyzing reports about system status
    - By performing reviews
- Configuration Manager
  - In charge of CM procedures
  - Ensures that procedures & policies for creating, changing & testing of code are followed
  - To control changes in code, introduces mechanisms for official change requests, their evaluation & authoriza-tion.

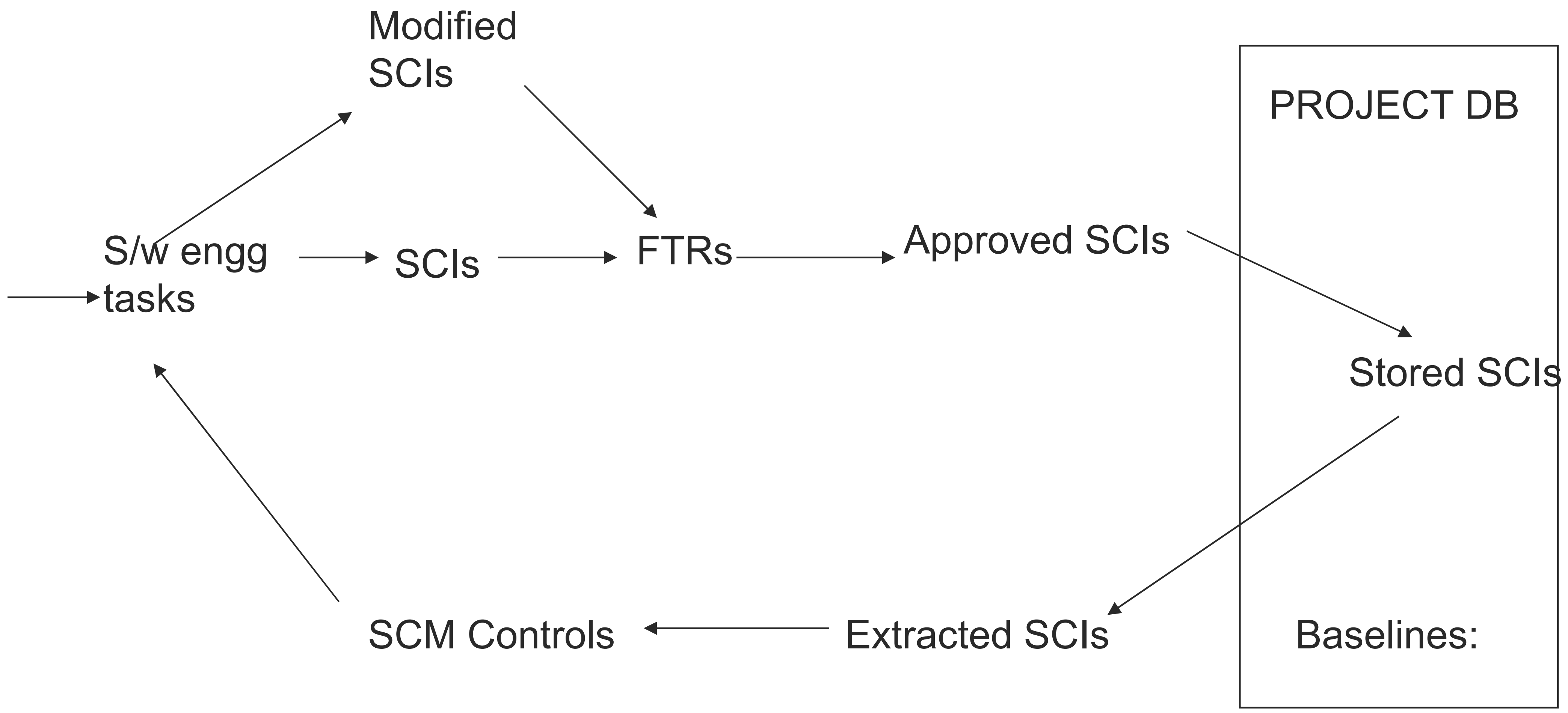
People involved in a typical SCM Scenario

- Software engineers
  - Work efficiently without interfering in each other’s code creation, testing & supporting documentation.
  - But at the same time communicate & coordinate efficiently by
    - Updating each other about tasks required & tasks completed
    - Propagating changes across each others work by merging files
- Customers
  - Use the product
  - Follows formal procedures of requesting changes and for indicting bugs in the product

Baselines

- IEEE defines baseline as:
  - A specification or product that has been formally reviewed and agreed upon, that thereafter serves as basis for further development, and that can be changed only through formal change control procedures.
- Before a s/w configuration item becomes a baseline, changes may be made quickly & informally.
- However, once it becomes a baseline, formal procedures must be applied to evaluate & verify every change.
- Refer to figure 27.1 to see steps for forming a baseline.

Baselined SCIs and Project DB (fig. 27.1)



Configuration Objects

- SCIs are organized to form configuration objects.
- A config. object has a name, attributes and is related to other objects.
- Example:
  - Test Specification – Config. Object
  - It comprises of SCIs like Test Plan, Test Procedure, Test Cases
  - It is related to config. object SourceCode
- Config. Objects are related to other cong. objects, the relationship is either compositional (part-of relationship) or interrelated.
- In related config. objects, if one of them changes, the other one has to be changed.

SCM Repository

- In early days of s/w engg, software configuration items were maintained as paper documents (or punch cards), placed in files and folders.
- The problems of this approach were:
  - Finding a configuration item when it was needed was often difficult
  - Determining which items were changed, when and by whom was often challenging
  - Constructing a new version of existing program was error-prone & time consuming
  - Describing complex relationships among configuration items was virtually impossible
- Today SCIs are maintained in a project database or repository.
- It acts as the center for accumulation and storage of SCIs

Role of repository

- SCM Repository is a set of mechanisms & data structures that allow a software team to manage change in an effective manner.
- It provides functions of a DBMS but in addition has the following functions:
  - Data integrity
  - Info sharing
  - Tool integration
  - Data integration
  - Methodology enforcement
  - Document standardization

    - See page 778 for details of the above

SCM Process

- SCM process defines a series of tasks having 4 primary objectives
  - To identify all items that collectively define the s/w configuration.
  - To manage changes to one or more of these items.
  - To facilitate construction of different versions of an application.
  - To ensure that s/w quality is maintained as configuration evolves over time.

Version Control

- *Combines procedures and tools to manage the different versions of configuration objects created during the software process.*
- Version
  - A particular version is an instance of a system that differs in some way from other instances (difference may be enhanced performance, functionality or repaired faults).
  - A software component with a particular version is a collection of objects.
  - A new version is defined when major changes have been made to one or more objects
- Variant
  - A variant is a collection of objects at the same revision level and coexists with other variants.
  - Variants have minor differences among each other.

Version Control

- Object 1.0 undergoes revision
  - Object 1.0 -> Object 1.1
- If minor changes/corrections introduced to Object 1.1
  - Object 1.1.1 & 1.1.2 (variants)
- Functional changes, new modules added to Object 1.1 can lead to
  - Object 1.1 ->Object 1.2
- Major Technical Change
  - Object 1.0 ->Object 2.0

Version Control System

- A number of automated tools are available for version control.
- CVS (Concurrent version System) is an example which is based on RCS (Revision Control System)
- CVS features
  - Establishes a simple repository.
  - Maintains all versions of a file in a single named file by storing only the differences between progressive versions of the original file.
  - Protects against simultaneous changes to a file by establishing different directions for each developer

Change Control

- A Change request is submitted and evaluated to assess technical merit and impact on the other configuration objects and budget
- Change report contains the results of the evaluation
- Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report
- Engineering change order (ECO) is generated for each change approved (describes change, lists the constraints, and criteria for review and audit)

Change Control (Contd.)

- Object to be changed is checked-out of the project database subject to access control parameters for the object
- Modified object is subjected to appropriate SQA and testing procedures
- Modified object is checked-in to the project database and version control mechanisms are used to create the next version of the software
- Synchronization control is used to ensure that parallel changes made by different people don't overwrite one another

Software Configuration Audit

- To ensure that a change has been properly implemented, Formal Technique Reviews (FTR) and software config. audits are used.
- A Software Config. Audit complements the FTR by addressing the following questions:
  - Has the change specified by the ECO been made without modifications?
  - Has an FTR been conducted to assess technical correctness?
  - Was the software process followed and software engineering standards applied?
  - Do the attributes of the configuration object reflect the change?
  - Have the SCM standards for recording and reporting the change been followed?
  - Were all related SCI's properly updated?

Configuration Status Reporting

- CSR addresses the following questions:
  - What happened?
  - Who did it?
  - When did it happen?
  - What else will be affected by the change?
- CSR entries are made when:
  - A SCI is assigned new or updated identification
  - Change is approved by CCA
  - Config. Audit is conducted
- CSR is generated on regular basis.

THE END ….

- Thank You

Software Quality Metrics

Pertemuan 10

Jaminan Kualitas Perangkat Lunak

OHT 21.2

If you can’t measure it, you can’t manage it

Tom DeMarco, 1982

Measurement, Measures, Metrics

- Measurement
  - is the act of obtaining a measure
- Measure
  - provides a quantitative indication of the size of some product or process attribute, E.g., Number of errors
- Metric
  - is a quantitative measure of the degree to which a system, component, or process possesses a given attribute *(IEEE Software Engineering Standards 1993) : Software Quality* - E.g., Number of errors found per person hours expended

What to measure

- Process

  Measure the efficacy of processes. What works, what doesn't.

- Project

  Assess the status of projects. Track risk. Identify problem areas. Adjust work flow.

- Product

  Measure predefined product attributes

What to measure

- Process

  Measure the efficacy of processes. What works, what doesn't.
- Code quality
- Programmer productivity
- Software engineer productivity
  - Requirements,
  - design,
  - testing
  - and all other tasks done by software engineers
- Software
  - Maintainability
  - Usability
  - And all other quality factors
- Management
  - Cost estimation
  - Schedule estimation, Duration, time
  - Staffing

Process Metrics

- Process metrics are measures of the software development process, such as
  - Overall development time
  - Type of methodology used
- Process metrics are collected across all projects and over long periods of time.
- Their intent is to provide indicators that lead to long-term software process improvement.

Project Metrics

- Project Metrics are the measures of Software Project and are used to monitor and control the project. Project metrics usually show how project manager is able to estimate schedule and cost
- They enable a software project manager to:
  - Minimize the **development time** by making the adjustments necessary to avoid delays and potential problems and risks.
  - Assess **product cost** on an ongoing basis & modify the technical approach to improve cost estimation.

Product metrics

- Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure:
  - How easy is the software to use
  - How easy is the user to maintain
  - The quality of software documentation
  - And more ..

Why do we measure?

- Determine quality of piece of software or documentation
- Determine the quality work of people such software engineers, programmers, database admin, and most importantly MANAGERS
- Improve quality of a product/project/ process

Why Do We Measure?

- To assess the benefits derived from new software engineering methods and tools
- To close the gap of any problems (E.g training)
- To help justify requests for new tools or additional training

Examples of Metrics Usage

- Measure estimation skills of project managers (Schedule/ Budget)
- Measure software engineers requirements/analysis/design skills
- Measure Programmers work quality
- Measure testing quality

And much more …

IEEE definitions of software quality metrics

(1) **A quantitative measure** of the degree to which an item possesses a given quality attribute.

(2) **A function** whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

Main objectives of software quality metrics

1. **Facilitate management control, planning and managerial intervention**.
   Based on:
   - Deviations of actual from planned performance.
   - Deviations of actual timetable and budget performance from planned.
2. **Identify situations for development or maintenance process improvement (preventive or corrective actions).** Based on:
   - Accumulation of metrics information regarding the performance of teams, units, etc.

Software size (volume) measures

- **KLOC** — classic metric that measures the size of software by thousands of code lines.
- **Number of function points (NFP)** — a measure of the development resources (human resources) required to develop a program, based on the functionality specified for the software system.

Process metrics categories

- **Software process quality metrics**
  - Error density metrics
  - Error severity metrics
- **Software process timetable metrics**
- **Software process error removal effectiveness metrics**
- **Software process productivity metrics**

Is KLOC enough ?

- What about number of errors (error density)?
- What about types of errors (error severity) ?
- A mixture of KLOC, density, and severity is an ideal quality metric to programmers quality of work and performance

An example

- 2 different project programmers are working on two similar modules
- Programmer A– produced 342 errors during software process before release
- Programmer B- Produced 184 errors

- Which Programmer do you think is better ?

An example

- It really depends on the types of errors found (severity) and not only the number of errors (Density)
- One error of high severity might be more important than hundreds of other types of errors

Error density metrics

Code	Name	Calculation formula
CED	Code Error Density	$CED = \frac{NCE}{KLOC}$
DED	Development Error Density	$DED = \frac{NDE}{KLOC}$
WCED	Weighted Code Error Density	$WCDE = \frac{WCE}{KLOC}$
WDED	Weighted Development Error Density	$WDED = \frac{WDE}{KLOC}$
WCEF	Weighted Code Errors per Function Point	$WCEF = \frac{WCE}{NFP}$
WDEF	Weighted Development Errors per Function Point	$WDEF = \frac{WDE}{NFP}$

NCE = The number of code errors detected by code inspections and testing.
NDE = total number of development (design and code) errors) detected in the development process.
WCE = weighted total code errors detected by code inspections and testing.
WDE = total weighted development (design and code) errors detected in development process.

Error severity metrics

Code	Name	Calculation formula
ASCE	Average Severity of Code Errors	$ASCE = \frac{WCE}{NCE}$
ASDE	Average Severity of Development Errors	$ASDE = \frac{WDE}{NDE}$

NCE = The number of code errors detected by code inspections and testing.
NDE = total number of development (design and code) errors detected in the development process.
WCE = weighted total code errors detected by code inspections and testing.
WDE = total weighted development (design and code) errors detected in development process.

Software process timetable metrics

Code	Name	Calculation formula
TTO	Time Table Observance	$TTO = \frac{MSOT}{MS}$
ADMC	Average Delay of Milestone Completion	$ADMC = \frac{TCDAM}{MS}$

MSOT = Milestones completed on time.
MS = Total number of milestones.
TCDAM = Total Completion Delays (days, weeks, etc.) for all milestones.

Time Table Metric Example

- TTO
  - Milestones are Requirements, Analysis, Design, Implementation, and Testing
  - Milestones completed in time are Requirements and analysis only
  - TTO = 2/5

Time Table Metric Example

- ADMC
  - Requirements (One week delay, Analysis (Three weeks delay, Design (Two weeks delay, Implementation (Six weeks delay), and Testing (Two weeks delay)
  - Total Delay is 14 Weeks
  - ADMS = 14/5

Error removal effectiveness metrics

Code	Name	Calculation formula
DERE	Development Errors Removal Effectiveness	$DERE = \frac{NDE}{NDE + NYF}$
DWERE	Development Weighted Errors Removal Effectiveness	$DWERE = \frac{WDE}{WDE+WYF}$

NDE = total number of development (design and code) errors) detected in the development process.
WCE = weighted total code errors detected by code inspections and testing.
WDE = total weighted development (design and code) errors detected in development process.
NYF = number software failures detected during a year of maintenance service.
WYF = weighted number of software failures detected during a year of maintenance service.

Error removal effectiveness metrics

- DERE
  - Number of errors detected at design and coding stages is 100
  - Number of errors detected after one year of maintenance service is 500
  - DERE= 100/(100+500)

Process productivity metrics

Code	Name	Calculation formula
DevP	Development Productivity	$DevP = \frac{DevH}{KLOC}$
FDevP	Function point Development Productivity	$FDevP = \frac{DevH}{NFP}$
CRe	Code Reuse	$Cre = \frac{ReKLOC}{KLOC}$
DocRe	Documentation Reuse	$DocRe = \frac{ReDoc}{NDoc}$

DevH = Total working hours invested in the development of the software system.
ReKLOC = Number of thousands of reused lines of code.
ReDoc = Number of reused pages of documentation.
NDoc = Number of pages of documentation.

Product metrics categories

* **Help Desk (HD) quality metrics:**
    * HD calls density metrics - measured by the number of calls.
    * HD calls severity metrics - the severity of the HD issues raised.
    * HD success metrics – the level of success in responding to HD calls.
* **HD productivity metrics.**
* **HD effectiveness metrics.**
* **Corrective maintenance quality metrics.**
    * Software system failures density metrics
    * Software system failures severity metrics
    * Failures of maintenance services metrics
    * Software system availability metrics
* **Corrective maintenance productivity and effectiveness metrics**.

HD calls density metrics

Code	Name	Calculation Formula
HDD	HD calls density	$HDD = \frac{NHYC}{KLMC}$
WHDD	Weighted HD calls density	$WHYC = \frac{WHYC}{KLMC}$
WHDF	Weighted HD calls per function point	$WHDF = \frac{WHYC}{NMFP}$

NHYC = the number of HD calls during a year of service.
KLMC = Thousands of lines of maintained software code.
WHYC = weighted HD calls received during one year of service.
NMFP = number of function points to be maintained.

Severity of HD calls metrics

Code	Name	Calculation Formula
ASHC	Average severity of HD calls	$\text{ASHC} = \dfrac{\text{WHYC}}{\text{NHYC}}$

NHYC = the number of HD calls during a year of service.

WHYC = weighted HD calls received during one year of service.

HD success metrics

Code	Name	Calculation Formula
HDS	HD service success	$HDS = \frac{NHYOT}{NHYC}$

NHYNOT = Number of yearly HD calls completed on time during one year of service.
NHYC = the number of HD calls during a year of service.

OHT 21.31

HD productivity and effectiveness metrics

Code	Name	Calculation Formula
HDP	HD Productivity	$HDP = \frac{HDYH}{KLNC}$
FHDP	Function Point HD Productivity	$FHDP = \frac{HDYH}{NMFP}$
HDE	HD effectiveness	$HDE = \frac{HDYH}{NHYC}$

HDYH = Total yearly working hours invested in HD servicing of the software system.
KLMC = Thousands of lines of maintained software code.
NMFP = number of function points to be maintained.
NHYC = the number of HD calls during a year of service.

Software system failures density metrics

Code	Name	Calculation Formula
SSFD	Software System Failure Density	$\text{SSFD} = \frac{\text{NYF}}{\text{KLMC}}$
WSSFD	Weighted Software System Failure Density	$\text{WFFFD} = \frac{\text{WYF}}{\text{KLMC}}$
WSSFF	Weighted Software System Failures per Function point	$\text{WSSFF} = \frac{\text{WYF}}{\text{NMFP}}$

NYF = number of software failures detected during a year of maintenance service.
WYF = weighted number of yearly software failures detected during one year of maintenance service.
NMFP = number of function points designated for the maintained software.
KLMC = Thousands of lines of maintained software code.

Failures of maintenance services metrics

Code	Name	Calculation Formula
MRepF	Maintenance Repeated repair Failure metric -	$\text{MRepF} = \dfrac{\text{RepYF}}{\text{NYF}}$

NYF = number of software failures detected during a year of maintenance service.

RepYF = Number of repeated software failure calls (service failures).

OHT 21.34

Software system availability metrics

Code	Name	Calculation Formula
FA	Full Availability	$FA = \frac{NYSerH - NYFH}{NYSerH}$
VitA	Vital Availability	$VitA = \frac{NYSerH - NYVitFH}{NYSerH}$
TUA	Total Unavailability	$TUA = \frac{NYTFH}{NYSerH}$

NYSerH = Number of hours software system is in service during one year.

NYFH = Number of hours where at least one function is unavailable (failed) during one year, including total failure of the software system.

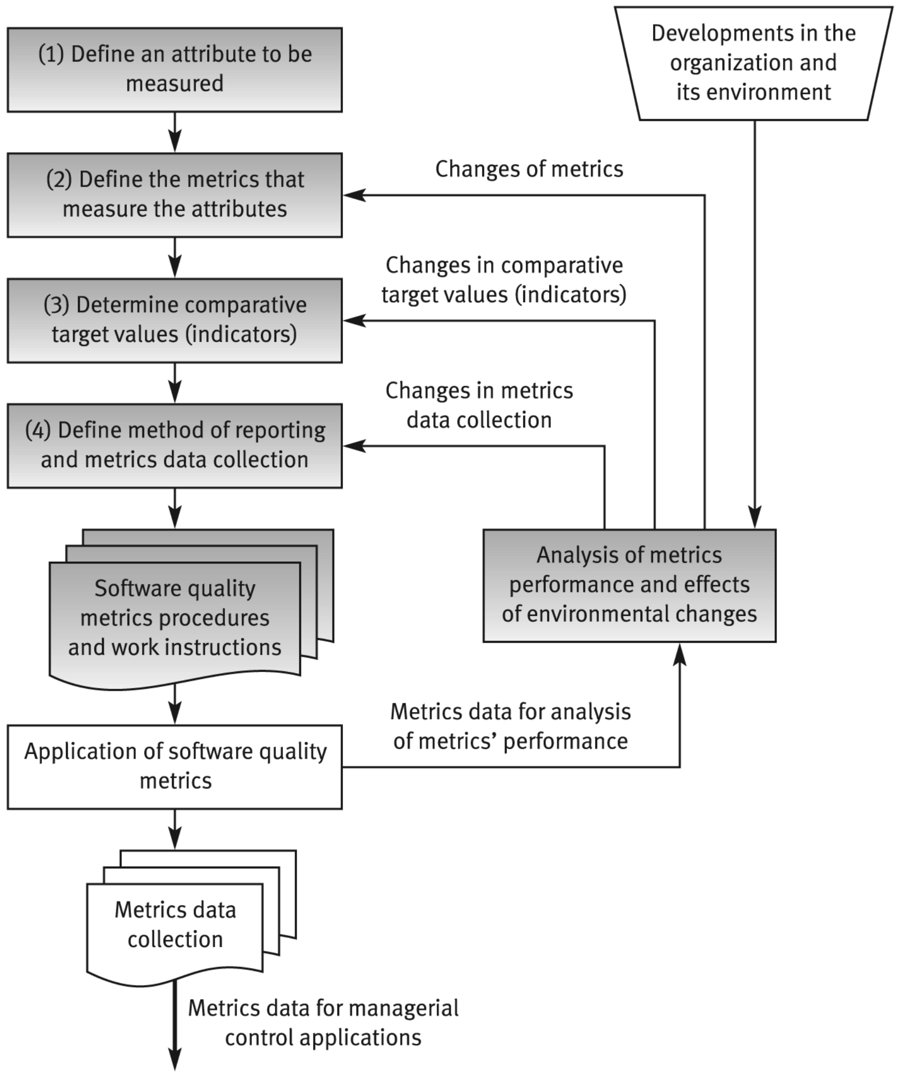
NYVitFH = Number of hours when at least one vital function is unavailable (failed) during one year, including total failure of the software system.

NYTFH = Number of hours of total failure (all system functions failed) during one year.

$NYFH \geq NYVitFH \geq NYTFH$.

$1 - TUA \geq VitA \geq FA$

The process of defining software quality metrics


(1) Define an attribute to be measured
(2) Define the metrics that measure the attributes
(3) Determine comparative target values (indicators)
(4) Define method of reporting and metrics data collection
Software quality metrics procedures and work instructions
Application of software quality metrics
Metrics data collection
Metrics data for managerial control applications
Developments in the organization and its environment
Changes of metrics
Changes in comparative target values (indicators)
Changes in metrics data collection
Analysis of metrics performance and effects of environmental changes
Metrics data for analysis of metrics' performance

General limitations of quality metrics

* **Budget** constraints in allocating the necessary resources.

* **Human factors**, especially opposition of employees to evaluation of their activities.

* **Validity** Uncertainty regarding the data's, partial and biased reporting.

Examples of metrics

Requirements

- Number of requirements that change during the rest of the software development process
  - if a large number changed during specification, design, …, something is wrong in the requirements phase and the quality of requirements engineers work
  - The less changes of requirements, the better requirements document quality

Examples of metrics

Inspection

number of faults found during inspection can be used as a metric to the quality of inspection process

Examples of metrics

Testing

- Number of test cases executed
- Number of bugs found per thousand of code
- And more possible metrics

Examples of metrics

Maintainability metrics

- Total number of faults reported
- classifications by severity, fault type
- status of fault reports (reported/fixed)
- Detection and correction times

Examples of metrics

Reliability quality factor

- Count of number of system failure (System down)
- Total of minutes/hours per week or month

An example

Programmers productivity using error severity and density

- We will consider error density and error severity
- Assume that we have three levels of severity
  - Low severity errors Wight is 1
  - Medium Severity Error Wight is 3
  - High Severity Error Wight is 15

- Two Programmers produced 3 KLOC per day
- After inspection, we found 100 low severity error, 20 medium severity errors and 10 High severity errors in the code of Programmer 1
- And 300 low severity error, 10 medium severity errors and 2 High severity errors in the code of Programmer 2

Which programmer has the highest code quality ?

Other Metrics

- Understanding
  - Learning time: Time for new user to gain basic understanding of features of the software
- Ease of learning
  - Learning time: Time for new user to learn how to perform basic functions of the software
- Operability
  - Operation time: Time required for a user to perform operation(s) of the software
- Usability
  - Human factors: Number of negative comments from new users regarding ergonomics, human factors, etc.

Other Metrics

- Number and type of defects found during requirements, design, code, and test inspections
- Number of pages of documentation delivered
- Number of new source lines of code created
- Number of source lines of code delivered
- Total number or source lines of code delivered
- Average complexity of all modules delivered
- Average size of modules
- Total number of modules
- Total number of bugs found as a result of unit testing
- Total number of bugs found as a result of integration testing
- Total number of bugs found as a result of validation testing
- Productivity, as measured by KLOC per person-hour

Examples... (continued)

- Metrics for the analysis model
- Metrics for the design model
- Metrics for the source code
- Average find-fix cycle time
- Number of person-hours per inspection
- Number of person-hours per KLOC
- Average number of defects found per inspection

Examples... (continued)

- Average find-fix cycle time
- Number of person-hours per inspection
- Number of person-hours per KLOC
- Average number of defects found per inspection
- Number of defects found during inspections in each defect category
- Average amount of rework time
- Percentage of modules that were inspected

Exercise

How can you measure the quality of:

- Project manager skills.
- Requirements Document
- Software development plan
- User manuals
- Programmer coding skills
- Design specification

ADMC Exercise

- An SDLC that consists of six phases
  - Requirements (3 weeks delay)
  - Analysis   (2 weeks delay)
  - Logical design (0 weeks delay)
  - Physical design (4 weeks delay
  - Implementation (10 weeks delay)
  - Testing (6 weeks delay)
- ADMC ?

QUALITY MANAGEMENT PLAN OVERVIEW

Course Goals

At the conclusion of this training, you will be able to:

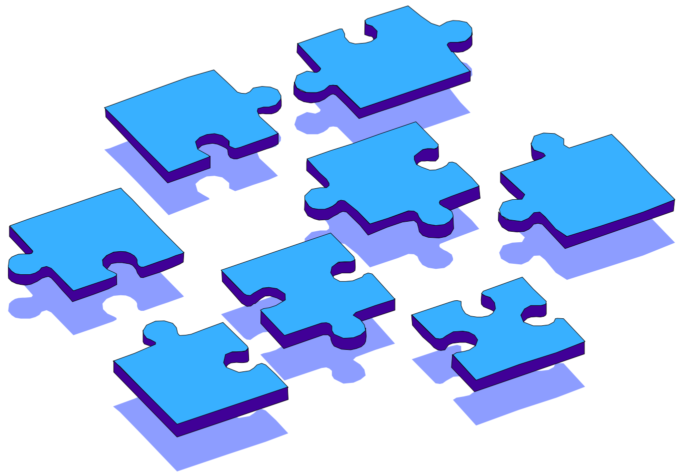
- Explain the basic considerations for Quality System design
- Explain what a Quality Management Plan (QMP) is and why it is required
- Discuss the ten elements of a QMP and the topics to cover in each
- Explain the graded approach to designing a QMP

Why Are You Here?

- Any organization conducting environmental programs by or on behalf of EPA must have a documented quality system
- The QMP documents that quality system

➡ You help develop a QMP.

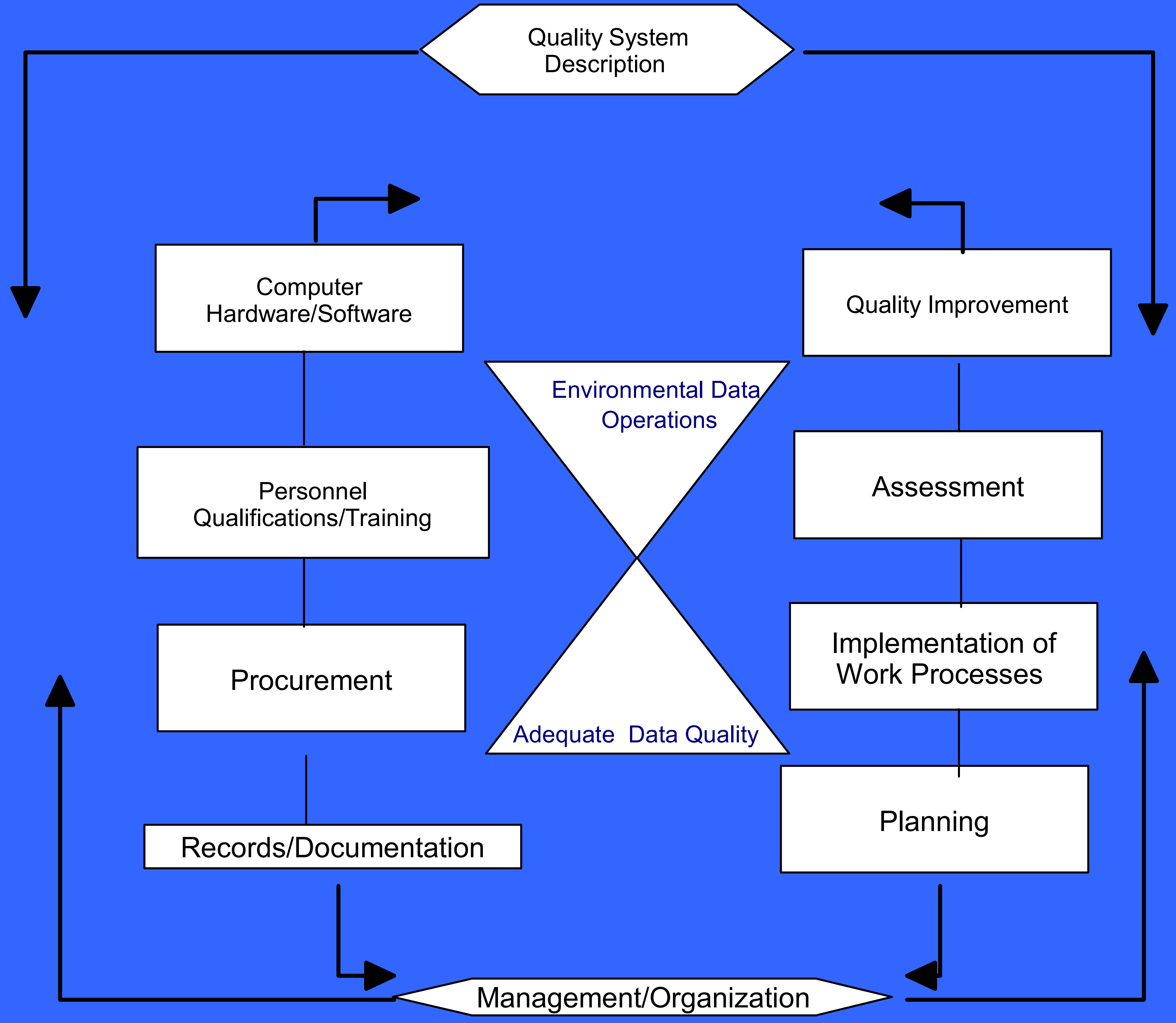
How is an organization vulnerable if inadequate data are collected?



What Is A Quality System?

A framework of management and technical practices that assure that environmental data used to support decisions are:

- Of adequate quality and usability for their intended purpose
- Defensible


Quality System Description
Computer Hardware/Software
Quality Improvement
Environmental Data Operations
Personnel Qualifications/Training
Assessment
Implementation of Work Processes
Procurement
Adequate Data Quality
Records/Documentation
Planning
Management/Organization

Common Steps in Quality System Design

- Prepare an overview of the current organization and its environmental data operations
- Identify existing Quality System components or tools
- Document existing procedures as appropriate
- Develop procedures where necessary
- Document the Quality System design

System Overview

Org.	Data Use	Technical Activities	Responsibility	Quality System Tools Used	Notes

Questions to Ask

- How does your organization use data (data generator, decision maker, regulatory)?
- What is the scope of technical activities within your organization (monitoring, regulatory actions, inspections, characterizations, program oversight)?
- Who is responsible for planning, implementing, and assessing these technical activities?
- What tools are already in use or are mandated (QAPPs, SOPs, TSAs)?

Graded Approach

QA and QC requirements commensurate with:

- Importance of work
- Available resources
- Unique needs of organization
- Consequences of potential decision errors

Tips for Design and Implementation

- Don't delegate responsibility for success of the Quality System too low in organization
- In larger organizations, avoid communicating QMP requirements to everyone all at once
- Use value added assessments rather than rigid audits
- Build success measures into the system so that you will know when it is working well

QMP Requirements Overview

QMP Requirement

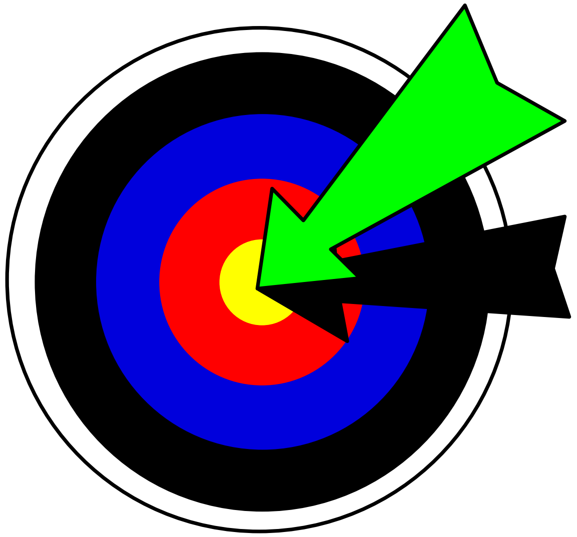
All work performed by EPA organizations and by external organizations funded by EPA that involves acquisition of environmental data generated from direct measurement activities, collected from other sources, or compiled from computerized data bases and information systems shall be covered by an Agency-approved QMP.

Internal and External Requirements

- Internal: EPA Order 5360.1 A2 2000
- External:
  - –48 CFR Part 46 (contracts)
  - –40 CFR Part 30, 31, and 35 (assistance agreements)
  - –Other (e.g., consent agreements in enforcement actions)

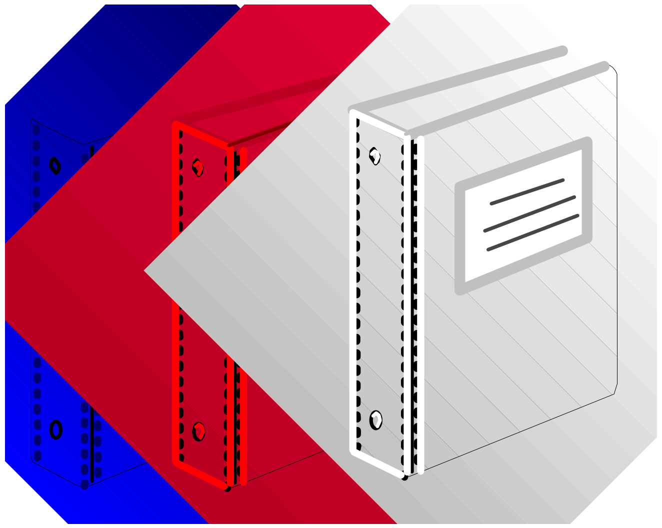
QMP Purpose

- The QMP documents what you are going to do, how you are going to do it, and how you know you did it.



Important Terms

- Quality Management
- Quality Assurance
- Quality Control



Scope of QMP

- Must reflect or emphasize actual practices, not planned practices
- Should use the graded approach
- Should be constructed and written so its effectiveness can be assessed

Revision

- QMP must be kept accurate and up-to-date in accordance with changes in QA policy and procedures
- Notify all appropriate personnel about changes to quality system and QMP

Revision (Cont’d)

- Internal: Submit revisions made during year to the Quality Staff with QA Annual Report and Work Plan.
- External: Submit revisions to the designated EPA official as a report when they occur. If changes are significant, submit entire QMP.

General Requirements

10 QMP Requirements

1. Management and Organization
2. Quality System Components
3. Personnel Qualifications and Training
4. Procurement of Items and Services
5. Documents and Records

10 QMP Requirements (Cont'd)

6. Computer Hardware and Software
7. Planning
8. Implementation of Work Processes
9. Assessment and Response
10. Quality Improvement

Approval Page

- Approval page signatures
  - Management
  - QA Manager of organization
  - Quality Staff Director (internal)
  - Senior EPA Official for Quality (internal)
  - Responsible EPA official (external)

- May be either part of title page or separate sheet following title page

If Not Applicable

If element is not applicable to an organization's quality system, a statement about why it is not must be included in QMP

Management and Organization

Overview: Management and Organization

A description of:

- QA Policy
- Organizational structure
- Staff roles, responsibilities, and authorities
- Technical activities
- Dispute resolution
- Quality System implementation

QA Policy

A statement of the organization's policy on quality assurance, including:

- Importance of QA and QC to organization and why
- General objectives/goals of Quality System
- Policy for resource allocation for the Quality System

Organization Chart

- An organization chart must:
  - Identify all components of the organization
  - Identify the position and lines of reporting for QA Manager and any QA staff
  - Show independence of the QA Manager from groups generating, compiling, and evaluating environmental data

Responsibilities and Authorities

Discuss the responsibilities and authorities of:

- QA Manager
- Any other QA staff

Technical Activities

- Discuss the technical activities or programs supported by the quality system
  - Specific programs that require quality management controls
  - Where oversight of extramural programs is needed to assure data quality
  - Where internal coordination of QA and QC activities needs to occur

Process for Resolving Disputes - Internal Only

Discuss the organization's process for resolving disputes regarding:

- Quality system requirements applicability
- QA and QC procedures application
- Assessments
- Corrective actions

Quality System Implementation

Discuss how management will ensure that applicable elements of the Quality System are understood and implemented in all environmental programs

Quality System Components

Overview: Quality System Components

1. Discuss principal components of Quality System and tools used to manage the system

   – What components comprise Quality System

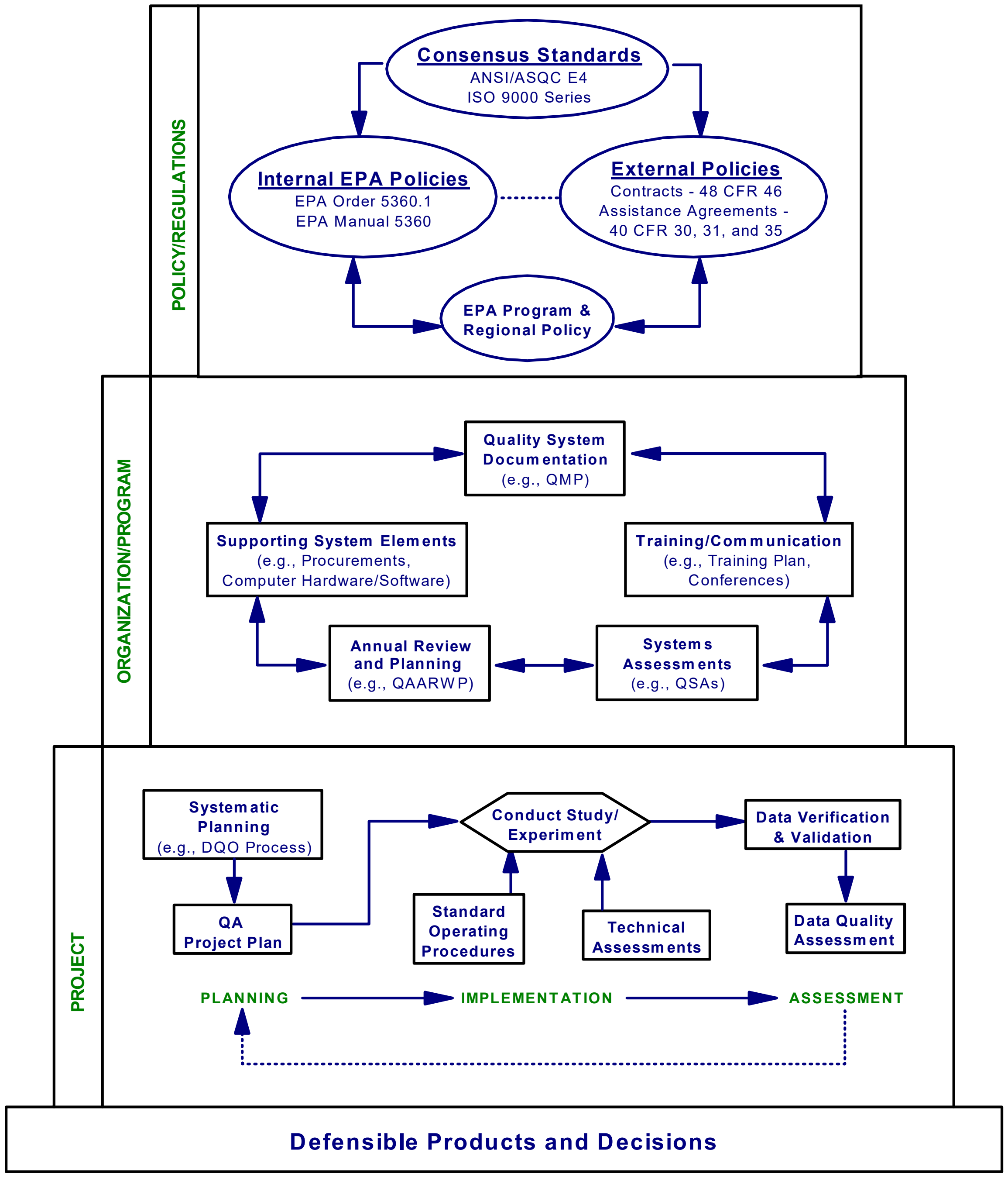
   – What tools are used to manage Quality System

   – Who uses them and how

Quality System Components (Cont'd)

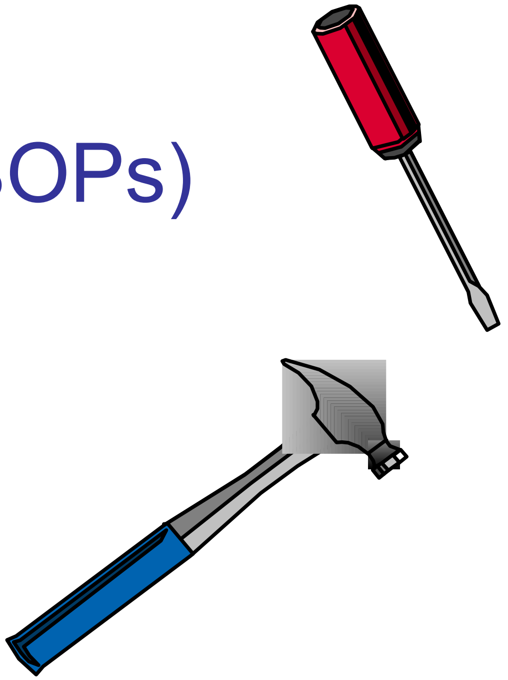
2. List parts of the organization that are required to prepare QMPs

3. Discuss the review and approval process for QMPs from external organizations


POLICY/REGULATIONS
Consensus Standards
ANSI/ASQC E4
ISO 9000 Series
Internal EPA Policies
EPA Order 5360.1
EPA Manual 5360
External Policies
Contracts - 48 CFR 46
Assistance Agreements - 40 CFR 30, 31, and 35
EPA Program & Regional Policy
ORGANIZATION/PROGRAM
Quality System Documentation (e.g., QMP)
Supporting System Elements (e.g., Procurements, Computer Hardware/Software)
Training/Communication (e.g., Training Plan, Conferences)
Annual Review and Planning (e.g., QAARWP)
Systems Assessments (e.g., QSAs)
PROJECT
Systematic Planning (e.g., DQO Process)
QA Project Plan
Conduct Study/ Experiment
Standard Operating Procedures
Technical Assessments
Data Verification & Validation
Data Quality Assessment
PLANNING
IMPLEMENTATION
ASSESSMENT
Defensible Products and Decisions

Quality System "Tools"

- QMPs
- Management Assessments - Self and Independent
- Technical Assessments
- Systematic Planning
- Standard Operating Procedures (SOPs)
- Quality Assurance Project Plans (QAPPs)
- Data Quality Assessments



Personnel Qualifications and Training

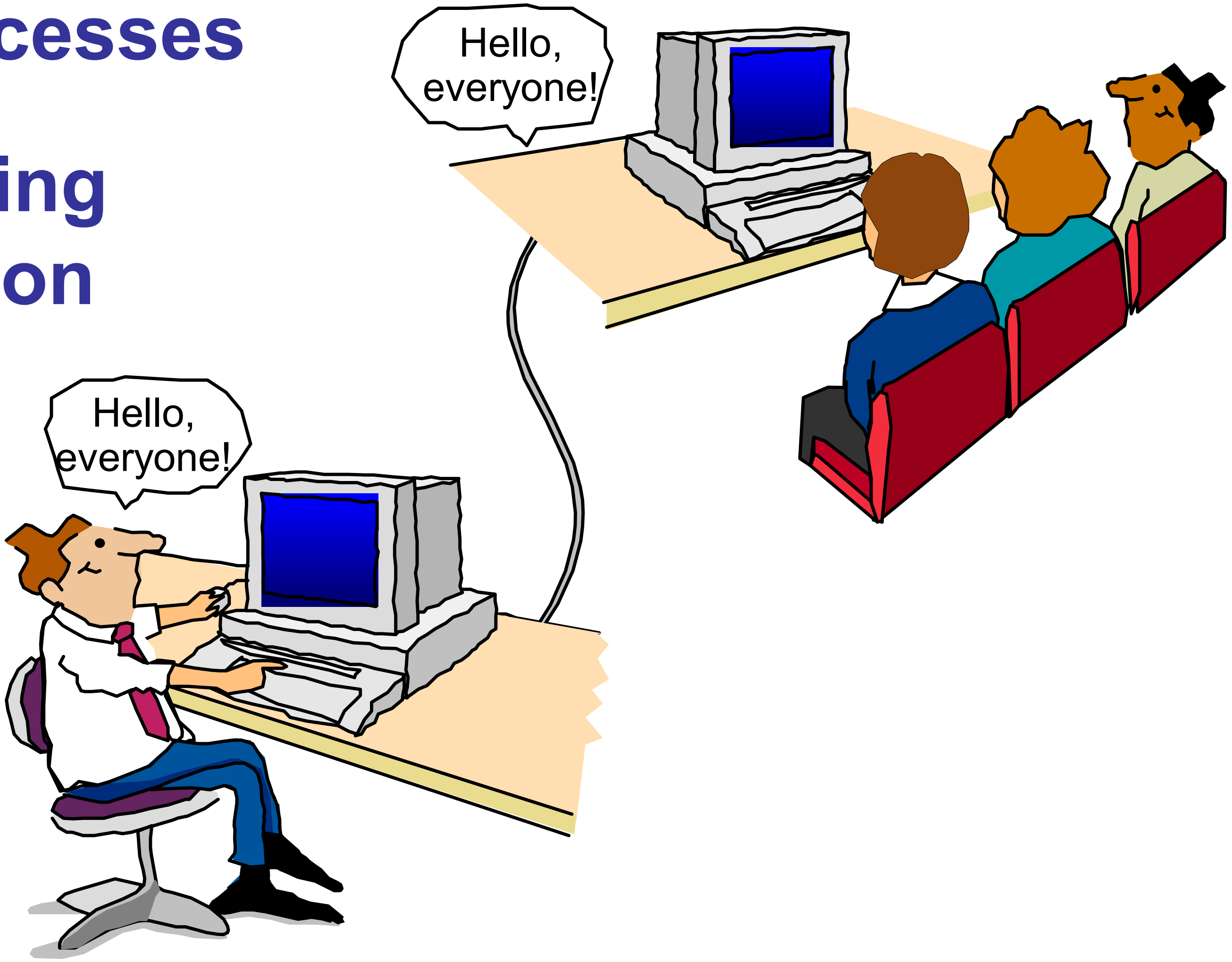
Overview: Personnel Qualification and Training

Describe organization's process for:

- Establishing training requirements
- Identifying training needs
- Assigning training priorities
- Meeting training needs

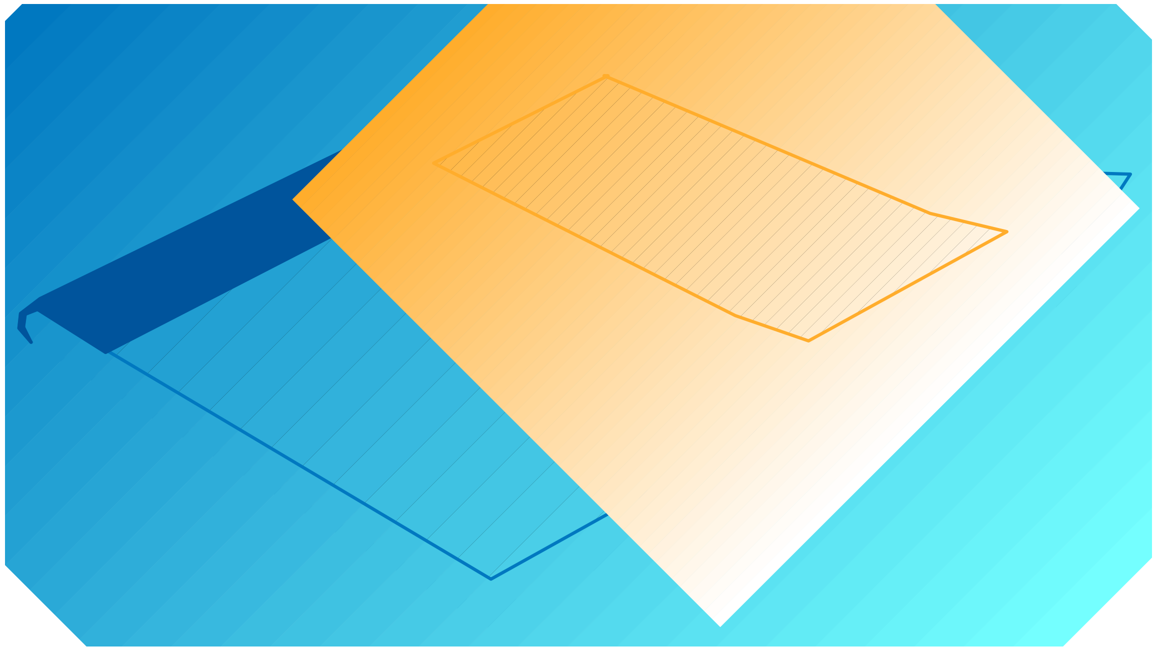
Three Areas to Cover

1. Training policy
2. Training processes
3. Critical training documentation



Policy

State the policy regarding training for management and staff



Processes

- Describe processes and management/staff responsible for:
  - Identifying statutory, regulatory, or professional certifications required
  - Identifying, designing, performing, and documenting training
    - Technical
    - Quality
    - Project management

Document Training

Describe how staff proficiency in critical technical disciplines is maintained and documented

Procurement of Items and Services

Overview: Procurement of Items and Services

- Describe organization's process for ensuring that suppliers provide items and services that:
  - Are of known and documented quality
  - Meet the technical requirements

Four Areas to Cover

1. Review and approval of procurement documents

2. Review and approval of QA responses to solicitations

3. Objective evidence of adequate quality from suppliers

4. QA document review and approval

Review and Approval of Procurement Documents

- Describe process and responsibilities for ensuring that documentation clearly states:
  - Item or service required
  - Technical and quality requirements
  - Quality System elements that the supplier is responsible for
  - How supplier's conformance to requirements will be verified

Review and Approval of QA Responses to Solicitations

- Describe roles, responsibilities, and process for review and approval of responses to solicitations to ensure:
  - That all technical and quality requirements are satisfied
  - That there is adequate evidence of supplier's capability to satisfy EPA Quality System requirements

Quality From Suppliers

- Describe process for determining adequate quality from suppliers, including:
  - How procurement sources are evaluated and selected
  - How and when source inspections are used
  - How deliverables are examined for conformance to specifications

QA Document Review and Approval

- Describe roles, responsibilities, processes and policies for:
  - Review and approval of mandatory quality related documentation
  - Delegation of review and approval of mandatory quality-related documentation
  - Ensuring that EPA quality-related contracting policies are satisfied

Documents and Records

Overview: Documents and Records

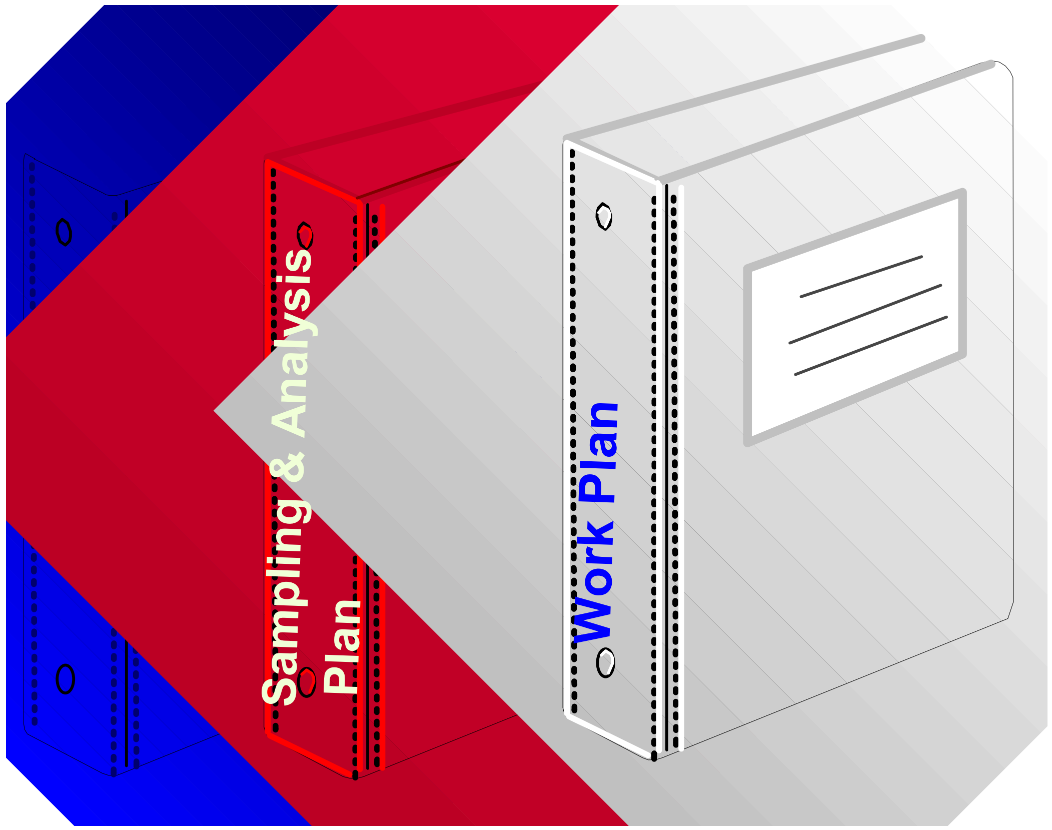
Discuss procedures for documents and records:

- Timely preparation
- Review
- Approval
- Issuance
- Use
- Control
- Revision
- Maintenance

Document

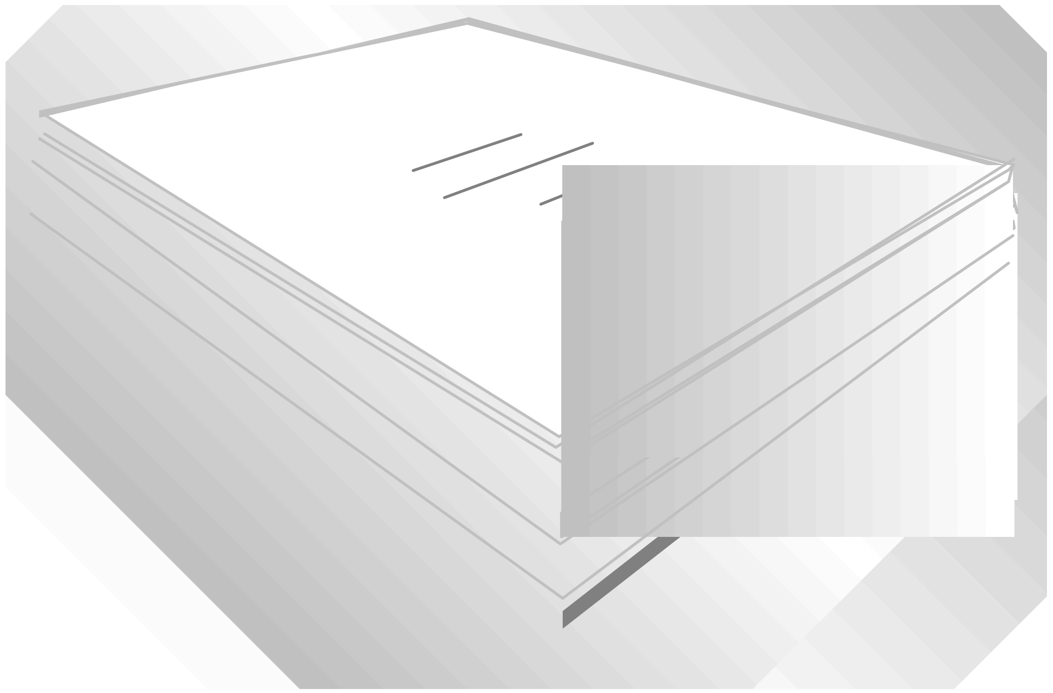
Any compilation of information that:

- Describes
- Defines
- Specifies
- Reports
- Certifies



Record

A completed document that provides objective evidence of an item or process.



Identify Documents and Records

Describe or reference process for identifying quality-related documents and records requiring version and release control

QA-Related Documents and Records

Describe or reference process by which all QA-related documents and records are:

– Accessible
– Protected from damage and deterioration
– Subject to retention requirements
– Prepared, reviewed, and approved
– Issued and revised

Compliance

Describe or reference process for ensuring compliance with all statutory, contractual, and assistance agreement requirements for records from environmental programs

Computer Hardware and Software

Overview: Computer Hardware and Software

The QMP must describe QA and QC processes for the use of computer hardware and software to support environmental data operations

General Requirements

- Computer hardware must be appropriate for its intended application
- Computer programs must be developed using an approved software development methodology

Examples of Hardware Covered

- Data acquisition/logging systems
- Computers used with analytical instruments
- Automated sampling systems in which system failure would adversely affect quality and potential usability of collected data for decisions

Examples of Software Covered

- Programs
- Models
- Data bases
- Data analysis systems

Hardware QA and QC Processes

- Describe or reference process for ensuring that computer hardware used in environmental programs meets technical requirements and quality expectations

- Describe or reference how changes to hardware will be controlled to assess impact of change on performance

Software Development QA and QC Processes

- Describe or reference process for:
  - Developing computer software
  - Validating, verifying, and documenting software for its use
  - Assuring that software meets requirements of user

IRM Policies

- Describe or reference process for ensuring:

1. That data and information produced and collected by computers meet applicable information resource management requirements and standards

IRM Policies

2. That applicable EPA requirements for information resources management are addressed, including:

   - Year 2000 compliance (Chapter 5)
   - Security (Chapter 8)
   - Privacy requirements (Chapter 11)

Planning

Overview: Planning

- Description of process for planning environmental programs
- Discussion of QAPP process
- Discussion of organization's secondary data policy

Systematic Planning

- Use systematic planning process based on scientific method
- Apply the principle of graded approach and common sense

Systematic Planning Requirements

- Describe process used for general project planning, including:
  - How it is accomplished
  - How planning will be documented
  - Who uses planning "tools"
  - Roles and responsibilities of management and staff

Quality Assurance Project Plans (QAPPs)

<u>**Describe process for:**</u>

- Developing
- Reviewing
- Approving
- Implementing
- Revising

<u>**the QAPP**</u>

Secondary Data

- Discuss process for evaluating and qualifying secondary data
- Describe application of any statistical methods used

Implementation of Work Processes

Overview: Implementation of Work Processes

- Describe process for implementing work within organization
  - Procedures for ensuring work is performed according to plan
  - Procedures for routine, standardized, special, or critical operations
  - Procedures for controlling and documenting the release, change, and use of planned procedures

Developing and Implementing Procedures

- Describe process for:
  - Ensuring that work is performed according to plan
  - Developing and implementing procedures for appropriate routine, standardized or special operations, including:
    - Identifying operations needing procedures
    - Preparation of procedures
  - Reviewing and approving procedures

Planned Procedures

- Describe measures for controlling planned work processes or procedures:
  - Their release
  - Their change
  - Their use

Controlling Measures

- Measures should provide for:
  - Necessary approvals
  - Specific times and points for implementing changes
  - Removal of obsolete documentation from work area
  - Verification that changes are made as prescribed

Assessment and Response

Overview: Assessment and Response

- Describe:
  - How and by whom assessments of environmental programs are planned, conducted, and evaluated
  - Process by which management determines:
    - Assessment activities and tools appropriate for a particular project
    - Expected frequency of use

Examples of Assessment Tools

- Management systems reviews
- Surveillance
- Technical systems audits
- Performance evaluations
- Audits of data quality
- Peer reviews and technical reviews
- Readiness reviews
- Data quality assessments

Two General Types of Assessment

1. Management

2. Technical

Assessment Activities

- Describe process:
  - For planning, conducting, and evaluating assessments
  - By which management chooses the appropriate assessment for a particular project

Assessment Activities (Cont'd)

- Describe:
  - How frequency of assessments is determined
  - Process for reporting and responding to results
  - Responsibilities, levels of participation, and authorities for all management and staff participating in assessment process

Assessor Qualifications

- Assessor qualifications
  - Technically knowledgeable
  - No real or perceived conflict of interest

Assessor Authority

- Describe how personnel conducting assessments will have:
  - Sufficient authority
  - Access to programs and managers
  - Access to documents and records
  - Organizational freedom

Assessment Results

- Describe process for review of assessment results
- Include:
  - Documentation of results
  - Reports to management
  - Management review procedures

Response Actions

- Describe process for responding to findings:
  - How corrective actions will be taken
  - When they will be taken
  - By whom they will be taken
- Describe how effectiveness of response will be determined and documented

Quality Improvement

Overview: Quality Improvement

- Describe:
  - How organization will detect and prevent quality problems and ensure continual quality improvement
  - Communication of expectations about quality improvement to staff

Quality Problems

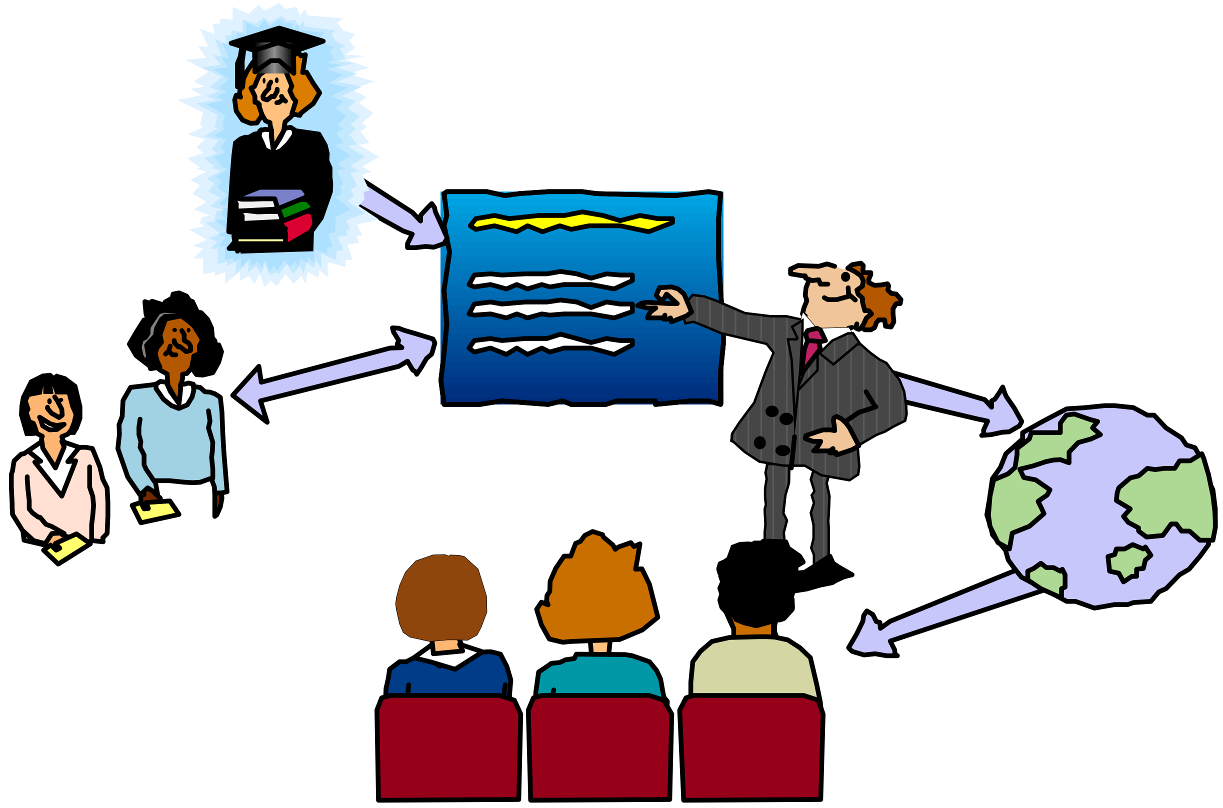
- Often inherent in existing management and technical systems
- Workers by themselves have little control over eliminating these problems or improving performance
- Action is required from management to commit to quality improvement

Quality Improvement Activities

- Describe management process and responsibilities for:
  - Identifying, planning, implementing, and evaluating quality improvement activities
  - Ensuring that conditions adverse to quality are identified promptly and corrected

Communicating Quality Improvement

- Describe how expectations for quality improvement are communicated



Implementing Quality Improvement

- Identify process improvement opportunities
- Identify problems
- Offer solutions to problems

Now You Can ...

- ✓ Explain what a Quality Management Plan (QMP) is and why it is required
- ✓ State the benefit of the QMP
- ✓ Describe the process and responsibilities for developing QMP within your program

Now You Can Also ...

✓ List 10 requirements for the QMP

✓ Describe what must be included in each requirement

Remember

Use

the

graded

approach!