Software Testing Techniques

- Testing fundamentals
- White-box testing
- Black-box testing

Testing

- Software testing is a process of executing a program or application with the intent of finding the software bugs.
- Activity to check whether the actual results match the expected results and to ensure that the software system is Defect free.
- Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements.
- done manually or using automated tools
  - Static testing- during verification process
  - Dynamic testing- during validation process

Testing Techniques

- Black-box testing
  - Knowing the specified function that a product has been designed to perform, test to see if that function is fully operational and error free
  - Includes tests that are conducted at the software interface
  - Not concerned with internal logical structure of the software

Testing Techniques contd..

- White-box testing
  - Knowing the internal workings of a product, test that all internal operations are performed according to specifications and all internal components have been exercised
  - Involves tests that concentrate on close examination of procedural detail
  - Logical paths through the software are tested
  - Test cases exercise specific sets of conditions and loops

White-box Testing

White-box Testing

- Uses the control structure part of component-level design to derive the test cases
- These test cases
  - Guarantee that <u>all independent paths</u> within a module have been exercised at least once
  - Exercise all logical decisions on their true and false sides
  - Execute all loops at their boundaries and within their operational bounds
  - Exercise internal data structures to ensure their validity

Basis Path Testing

- White-box testing technique proposed by Tom McCabe
- Enables the test case designer to derive a logical complexity measure of a procedural design
- Uses this measure as a guide for defining a basis set of execution paths
- Test cases derived to exercise the basis set are guaranteed to execute <u>every statement</u> in the program <u>at least one time</u> during testing

Flow Graph Notation

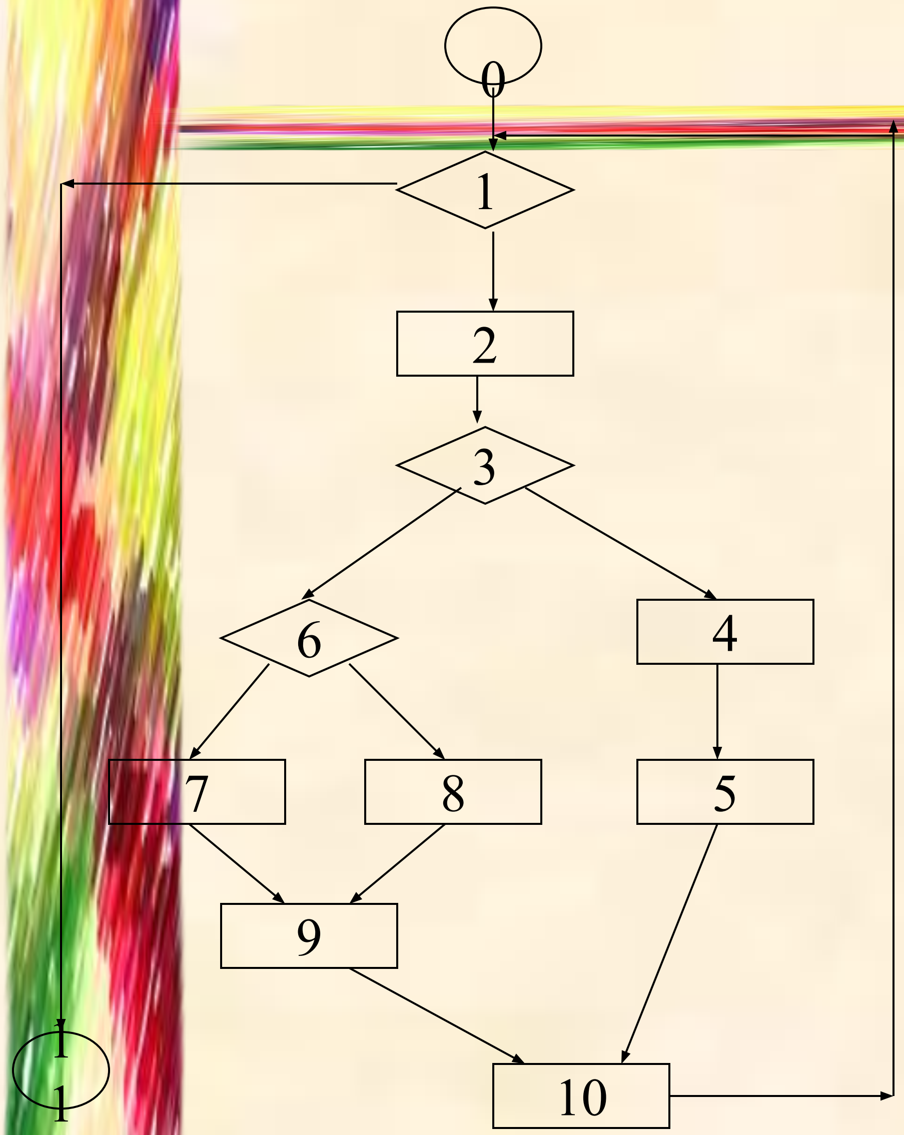
- A circle in a graph represents a <u>node</u>, which stands for a <u>sequence</u> of one or more procedural statements
- A node containing a simple conditional expression is referred to as a <u>predicate node</u>
  - Each <u>compound condition</u> in a conditional expression containing one or more Boolean operators (e.g., and, or) is represented by a separate predicate node
  - A predicate node has <u>two</u> edges leading out from it (True and False)

Flow Graph Notation

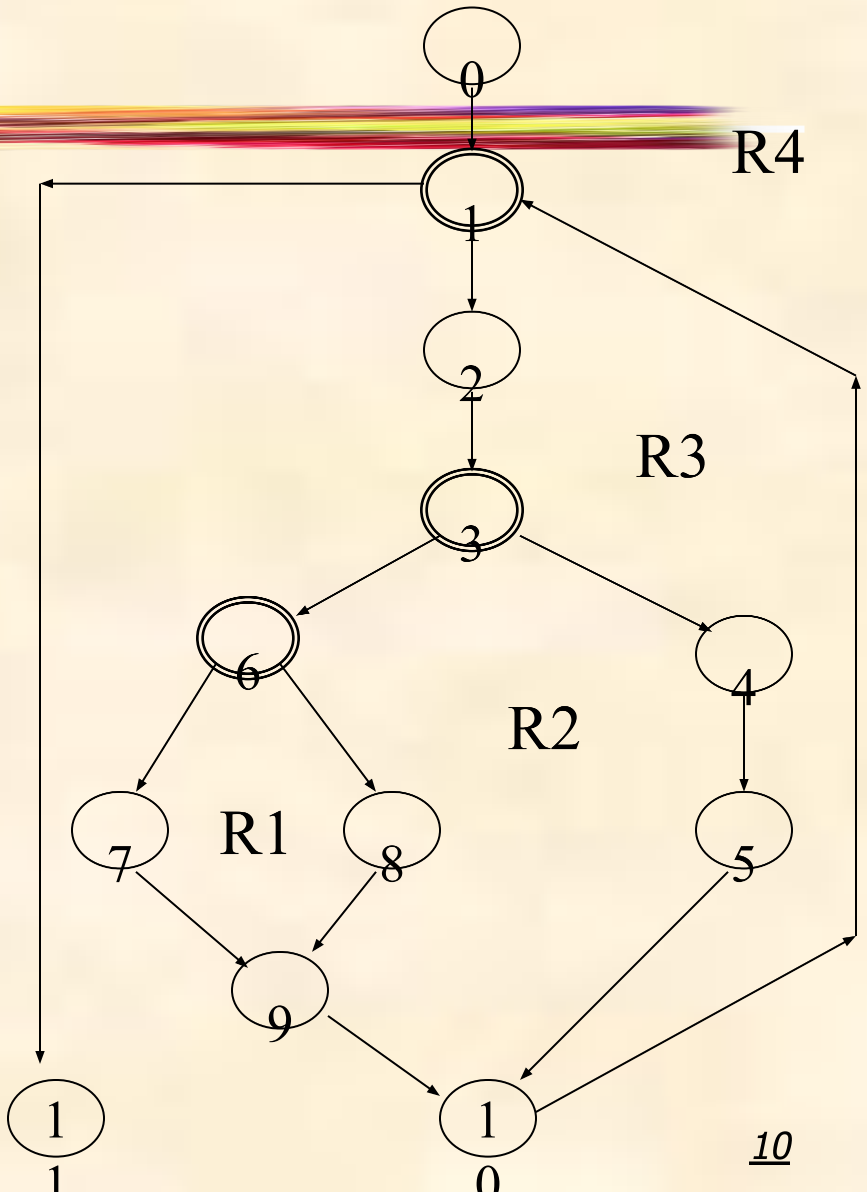
- An edge, or a link, is a an arrow representing flow of control in a specific direction
  - An edge must start and terminate at a node
  - An edge does not intersect or cross over another edge
- Areas bounded by a set of edges and nodes are called regions
- When counting regions, include the area outside the graph as a region, too

Flow Graph Example

FLOW CHART



FLOW GRAPH



Independent Program Paths

- Defined as a path through the program from the start node until the end node that introduces at least one new set of processing statements or a new condition (i.e., new nodes)
- Must move along <u>at least one</u> edge that has not been traversed before by a previous path
- Basis set for flow graph on previous slide
  - Path 1: 0-1-11
  - Path 2: 0-1-2-3-4-5-10-1-11
  - Path 3: 0-1-2-3-6-8-9-10-1-11
  - Path 4: 0-1-2-3-6-7-9-10-1-11
- The <u>number of paths</u> in the basis set is determined by the <u>cyclomatic complexity</u>

Cyclomatic Complexity

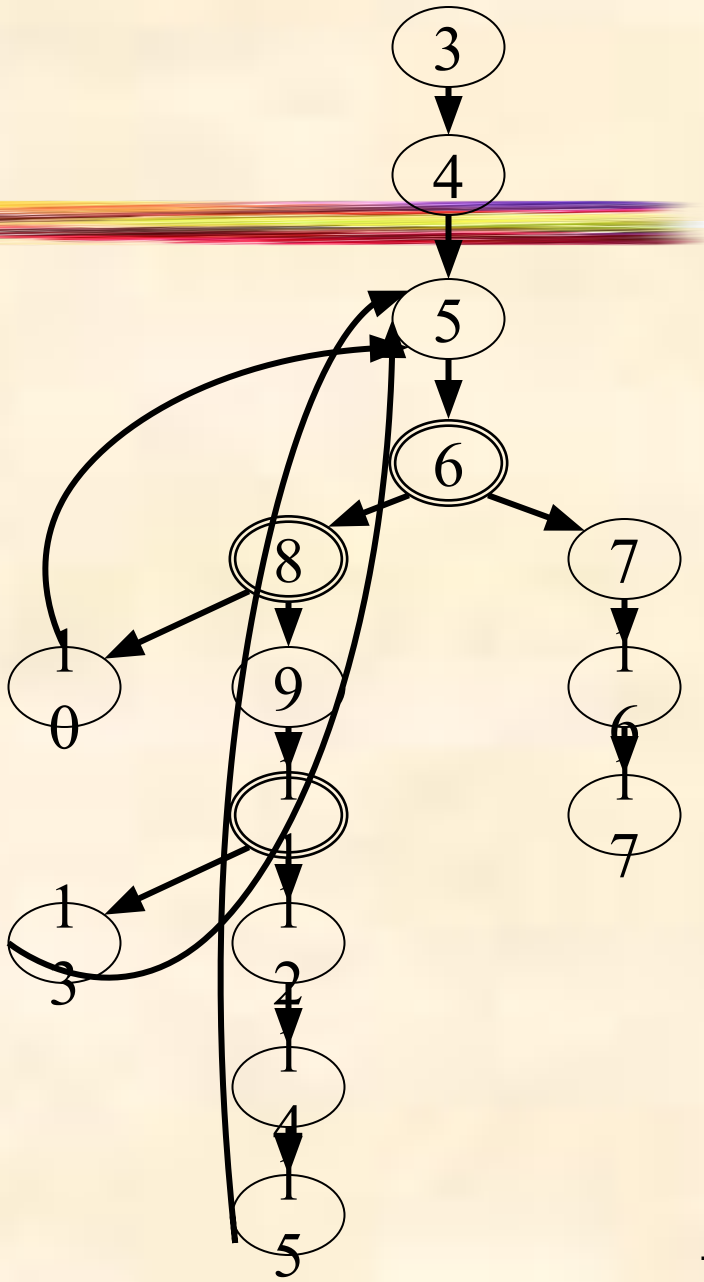
- Provides a quantitative measure of the <u>logical complexity</u> of a program
- Defines the <u>number of independent paths</u> in the basis set
- Provides an <u>upper bound</u> for the number of tests that must be conducted to ensure <u>all statements</u> have been executed <u>at least once</u>
- Can be computed <u>three</u> ways
  - The number of regions
  - V(G) = E – N + 2, where E is the number of edges and N is the number of nodes in graph G
  - V(G) = P + 1, where P is the number of predicate nodes in the flow graph G
- Results in the following equations for the example flow graph
  - Number of regions = 4
  - V(G) = 14 edges – 12 nodes + 2 = 4
  - V(G) = 3 predicate nodes + 1 = 4

Deriving the Basis Set and Test Cases

1) **Using the design or code as a foundation, draw a corresponding flow graph**
2) **Determine the cyclomatic complexity of the resultant flow graph**
3) **Determine a basis set of linearly independent paths**
4) **Prepare test cases that will force execution of each path in the basis set**

A Second Flow Graph Example

```
 1  int functionY(void)
 2  {
 3      int x = 0;
 4      int y = 19;

 5  A: x++;
 6      if (x > 999)
 7          goto D;
 8      if (x % 11 == 0)
 9          goto B;
10      else goto A;

11  B: if (x % y == 0)
12          goto C;
13      else goto A;

14  C: printf("%d\n", x);
15      goto A;

16  D: printf("End of list\n");
17      return 0;
18  }
```

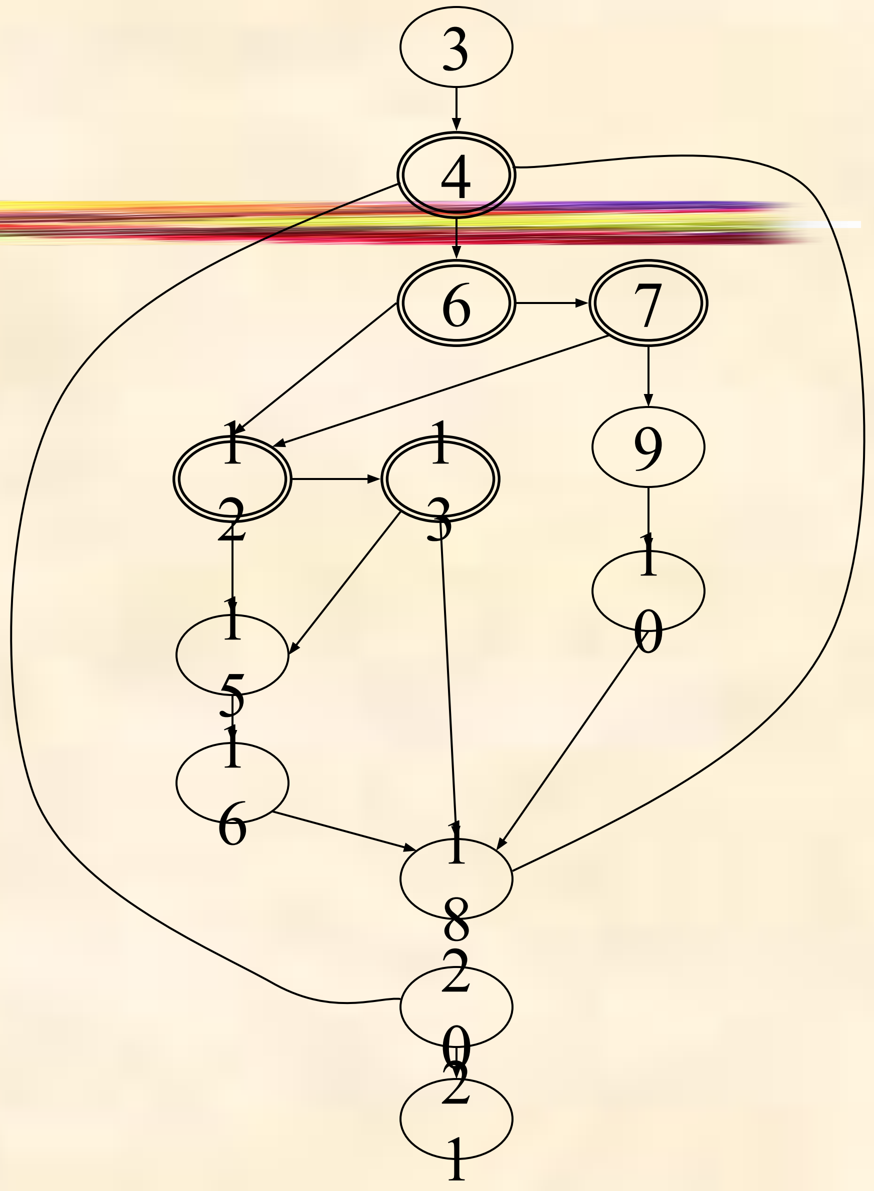


A Sample Function to Diagram and Analyze

```
 1  int functionZ(int y)
 2  {
 3  int x = 0;

 4  while (x <= (y * y))
 5     {
 6     if ((x % 11 == 0) &&
 7          (x % y == 0))
 8        {
 9        printf(“%d”, x);
10        x++;
11        } // End if
12     else if ((x % 7 == 0) ||
13               (x % y == 1))
14        {
15        printf(“%d”, y);
16        x = x + 2;
17        } // End else
18     printf(“\n”);
19     } // End while

20  printf("End of list\n");
21  return 0;
22  } // End functionZ
```

A Sample Function to Diagram and Analyze

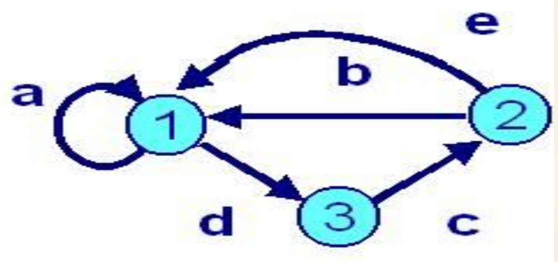
```
1    int functionZ(int y)
2    {
3    int x = 0;

4    while (x <= (y * y))
5       {
6       if ((x % 11 == 0) &&
7             (x % y == 0))
8          {
9          printf(“%d”, x);
10         x++;
11         } // End if
12      else if ((x % 7 == 0) ||
13               (x % y == 1))
14         {
15         printf(“%d”, y);
16         x = x + 2;
17         } // End else
18      printf(“\n”);
19      } // End while

20   printf("End of list\n");
21   return 0;
22   } // End functionZ
```



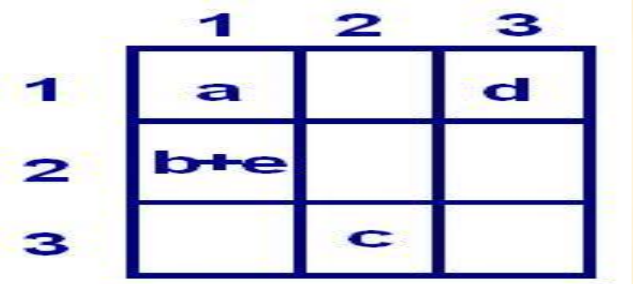
Graph Matrices

- To develop s/w tool that assist in basis path testing, a DS called graph matrix is quite useful.
- Is a square matrix whose size is equal to no of nodes on the flow graph
- Each node is identified by nos. & each edge is identified by letters
- Letter entry is made to correspond to a connection between two nodes.
- Link weights is a 1 or 0. (connection matrix)
- Provides other way to determine cyclomatic complexity

Graph Matrices contd..

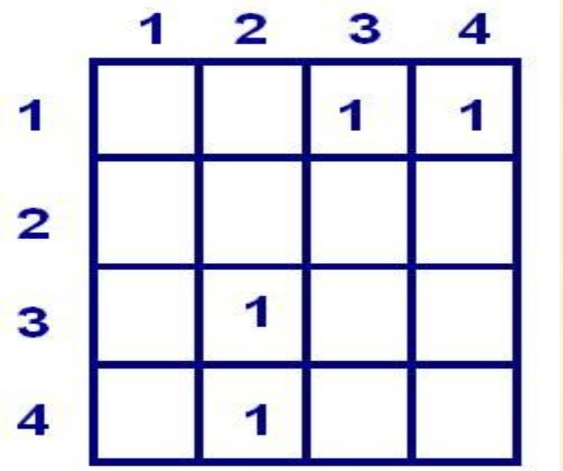


Flow Graph



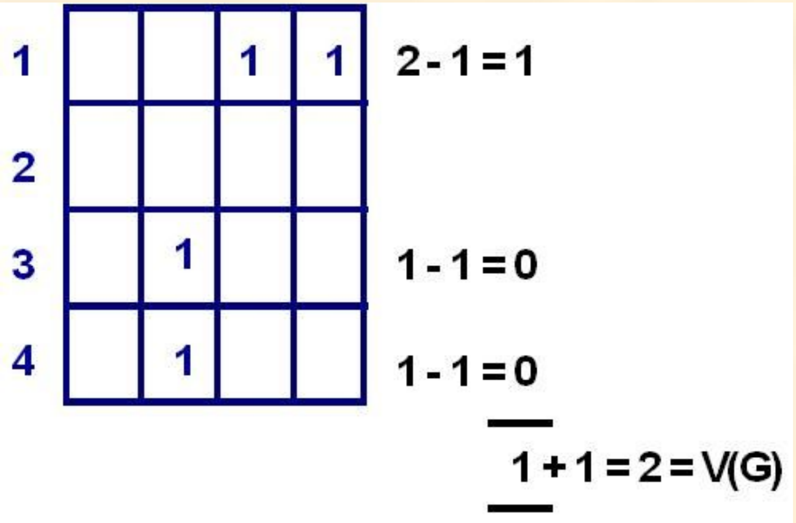
	1	2	3
1	a		d
2	b+e		
3		c	

Graph Matrix



	1	2	3	4
1			1	1
2				
3		1		
4		1		

Connection Matrix



1			1	1	2 - 1 = 1
2					
3		1			1 - 1 = 0
4		1			1 - 1 = 0

$1 + 1 = 2 = V(G)$

Calculation of V(G)

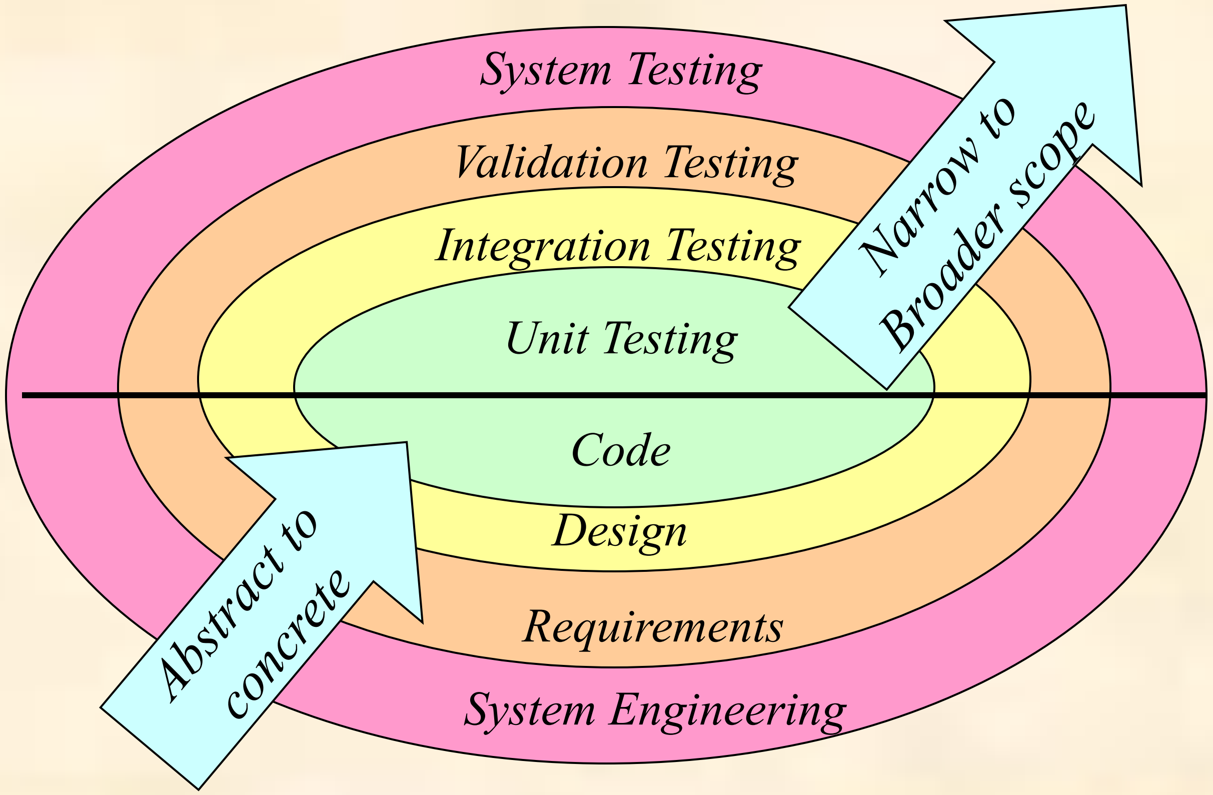
Black-box Testing

Black-box Testing Categories

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external data base access
- Behavior or performance errors
- Initialization and termination errors

A Strategic Approach to Testing

A Strategy for Testing Conventional Software



Levels of Testing for Conventional Software

- Unit testing
  - Concentrates on each component/function of the software as implemented in the source code
- Integration testing
  - Focuses on the design and construction of the software architecture
- Validation testing
  - Requirements are validated against the constructed software
- System testing
  - The software and other system elements are tested as a whole

Unit Testing

- Focuses testing on the function or software module
- Concentrates on the internal processing logic and data structures
- Is simplified when a module is designed with high cohesion
  - Reduces the number of test cases
  - Allows errors to be more easily predicted and uncovered
- Concentrates on critical modules and those with high cyclomatic complexity when testing resources are limited

Integration Testing

- Defined as a systematic technique for constructing the software architecture
  - At the same time integration is occurring, conduct tests to uncover errors associated with interfaces
- Objective is to take unit tested modules and build a program structure based on the prescribed design
- Two Approaches
  - Non-incremental Integration Testing
  - Incremental Integration Testing

Sample Integration Test Cases for the following scenario:

Application has 3 modules say 'Login Page', 'Mail box' and 'Delete mails' and each of them are integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	**Test Case Objective**	**Test Case Description**	**Expected Result**
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mail box select the an email and click delete button	Selected email should appear in the Deleted/Trash folder

Non-incremental Integration Testing

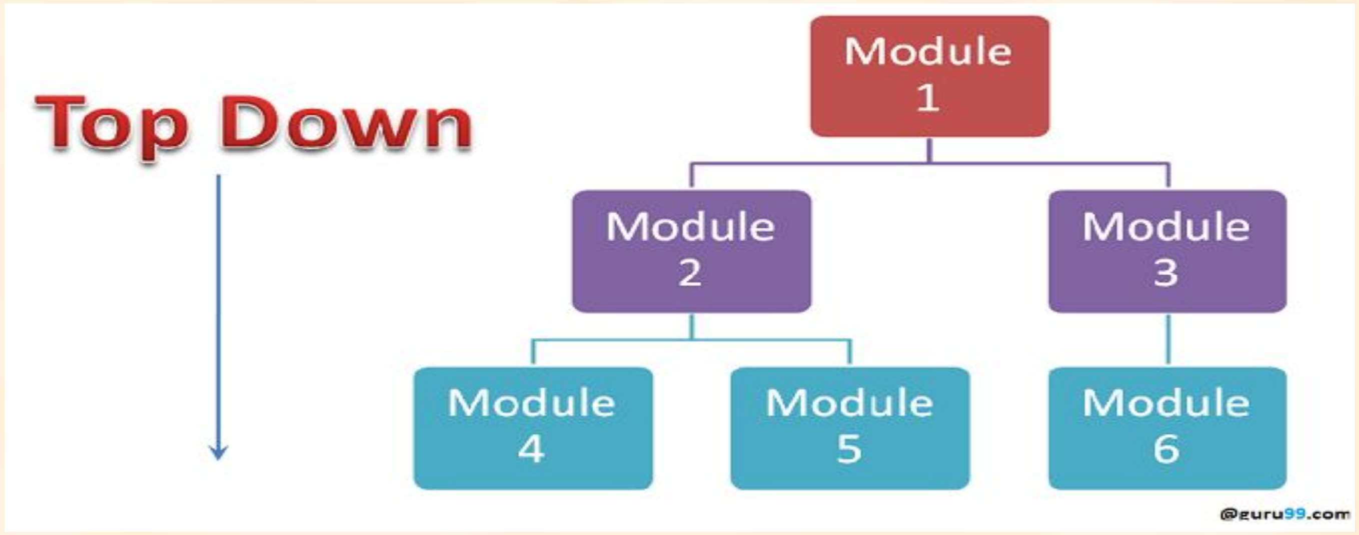
- Commonly called the "Big Bang" approach
- All components are combined in advance
- The entire program is tested as a whole
- Chaos results
- Many seemingly-unrelated errors are encountered
- Correction is difficult because isolation of causes is complicated
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

Incremental Integration Testing

- Three kinds
  - Top-down integration
  - Bottom-up integration
  - Sandwich integration
- The program is constructed and tested in small increments
- Errors are easier to isolate and correct
- Interfaces are more likely to be tested completely
- A systematic test approach is applied

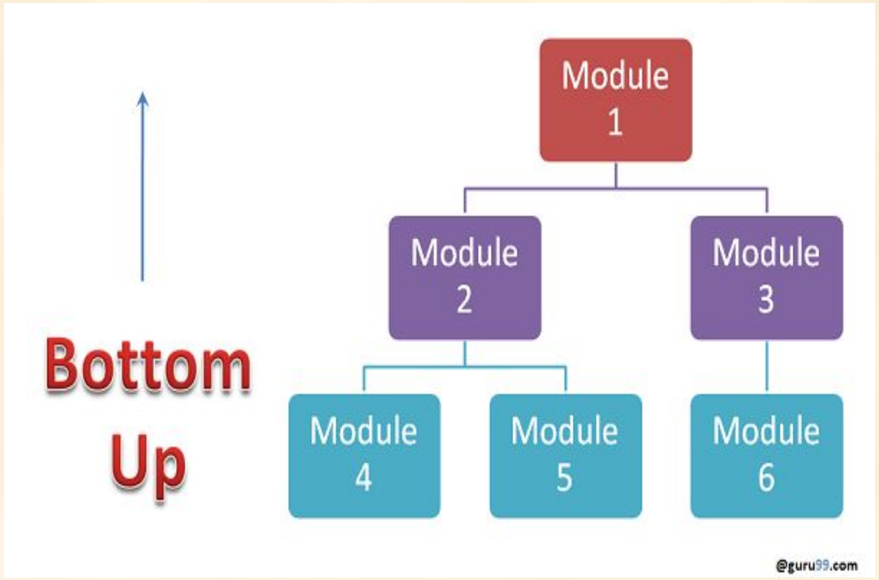
Top-down Integration

- Modules are integrated by moving downward through the control hierarchy, beginning with the main module
- Subordinate modules are incorporated in either a depth-first or breadth-first fashion



Bottom-up Integration

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing



Bottom-up Integration

Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

- Early prototype is not possible

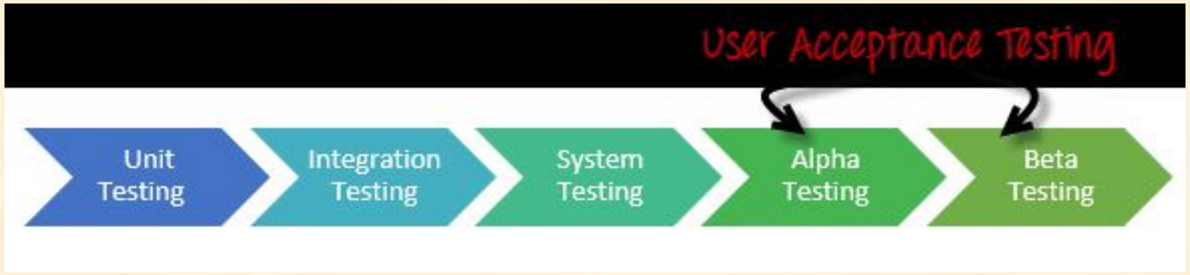
Regression Testing

- **Each new addition or change to baselined software may cause problems with functions that previously worked flawlessly**
- **Regression testing re-executes a small subset of tests that have already been conducted**
  - **Ensures that changes have not propagated unintended side effects**
  - **Helps to ensure that changes do not introduce unintended behavior or additional errors**
  - **May be done manually or through the use of automated capture/playback tools**
- **Regression test suite contains three different classes of test cases**
  - **A representative sample of tests that will exercise all software functions**
  - **Additional tests that focus on software functions that are likely to be affected by the change**
  - **Tests that focus on the actual software components that have been changed**

Validation Testing

Background

- Validation testing follows integration testing
- Focuses on user-visible actions and user-recognizable output from the system
- Demonstrates conformity with requirements
- Designed to ensure that
  - All functional requirements are satisfied
  - All behavioral characteristics are achieved
  - All performance requirements are attained
  - Documentation is correct
  - Usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability)
- After each validation test
  - The function or performance characteristic conforms to specification and is accepted
  - A deviation from specification is uncovered and a deficiency list is created
- A configuration review or audit ensures that all elements of the software configuration have been properly developed, cataloged, and have the necessary detail for entering the support phase of the software life cycle


User Acceptance Testing
Unit Testing
Integration Testing
System Testing
Alpha Testing
Beta Testing

Alpha and Beta Testing

- Alpha testing
  - Conducted at the developer's site by end users
  - Software is used in a natural setting with developers watching intently
  - Testing is conducted in a controlled environment
- Beta testing
  - Conducted at end-user sites
  - Developer is generally not present
  - It serves as a live application of the software in an environment that cannot be controlled by the developer
  - The end-user records all problems that are encountered and reports these to the developers at regular intervals
- After beta testing is complete, software engineers make software modifications and prepare for release of the software product to the entire customer base

System Testing

Different Types

- Recovery testing
  - Tests for recovery from system faults
  - Forces the software to fail in a variety of ways and verifies that recovery is properly performed
  - Tests reinitialization, checkpointing mechanisms, data recovery, and restart for correctness
- Security testing
  - Verifies that protection mechanisms built into a system will, in fact, protect it from improper access
- Stress testing
  - Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance testing
  - Tests the run-time performance of software within the context of an integrated system
  - Often coupled with stress testing and usually requires both hardware and software instrumentation

Software Configuration Management

By

Purvi D. Sankhe

Software Configuration Management

- SCM is an umbrella activity i.e. applied throughout the software process, because change can occur at any time.
- SCM activities are developed to,
  1. Identify change.
  2. Control change
  3. Ensure that change is being properly implemented, and
  4. Report to others who may have an interest.

The “First Law”

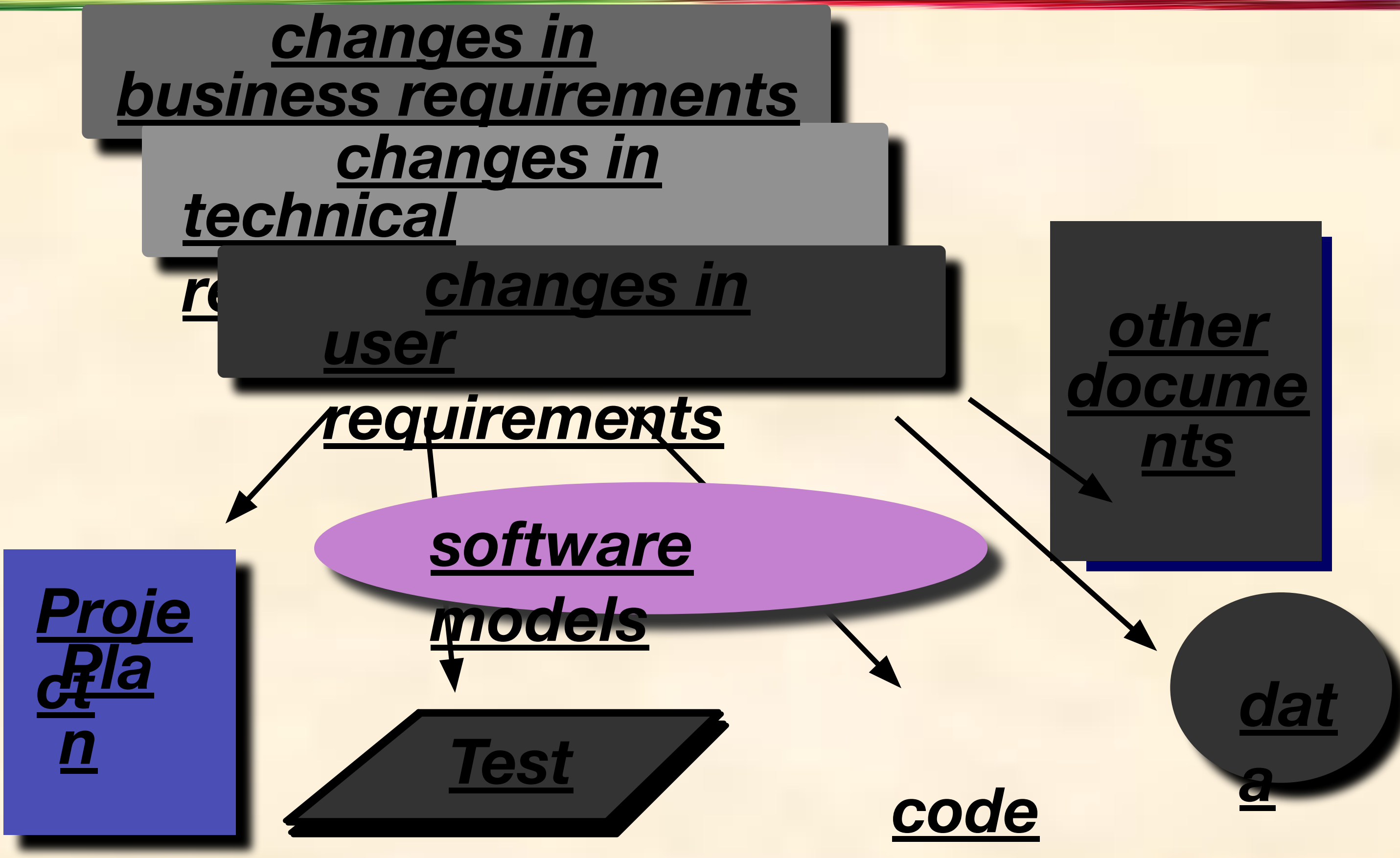
No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.

- Bersoff, et al, 1980

Fundamental Source of change

- New business or market conditions dictate changes to product requirements or business rules
- New customer needs demand modification of data, functionality, or services
- software engineering team structure
- Budgetary or scheduling constraints cause system to be redefined

What Are These Changes?


changes in business requirements
changes in technical requirements
changes in user requirements
other documents
software models
Project Plan
Test
code
data

Software Configuration Items

- It is information i.e. created as part of software engineering process.
- It may be,
  - Computer programs (both source and executable).
  - Documentation (both technical and user).
  - Data (contained within the program or external to it).

Baseline

- A work product or specification becomes a baseline only after it is reviewed and approved, that thereafter serves as the basis for further development.
- Baseline product or specification can be changed only through formal change control procedure.
- A baseline is a milestone in software development that is marked by the

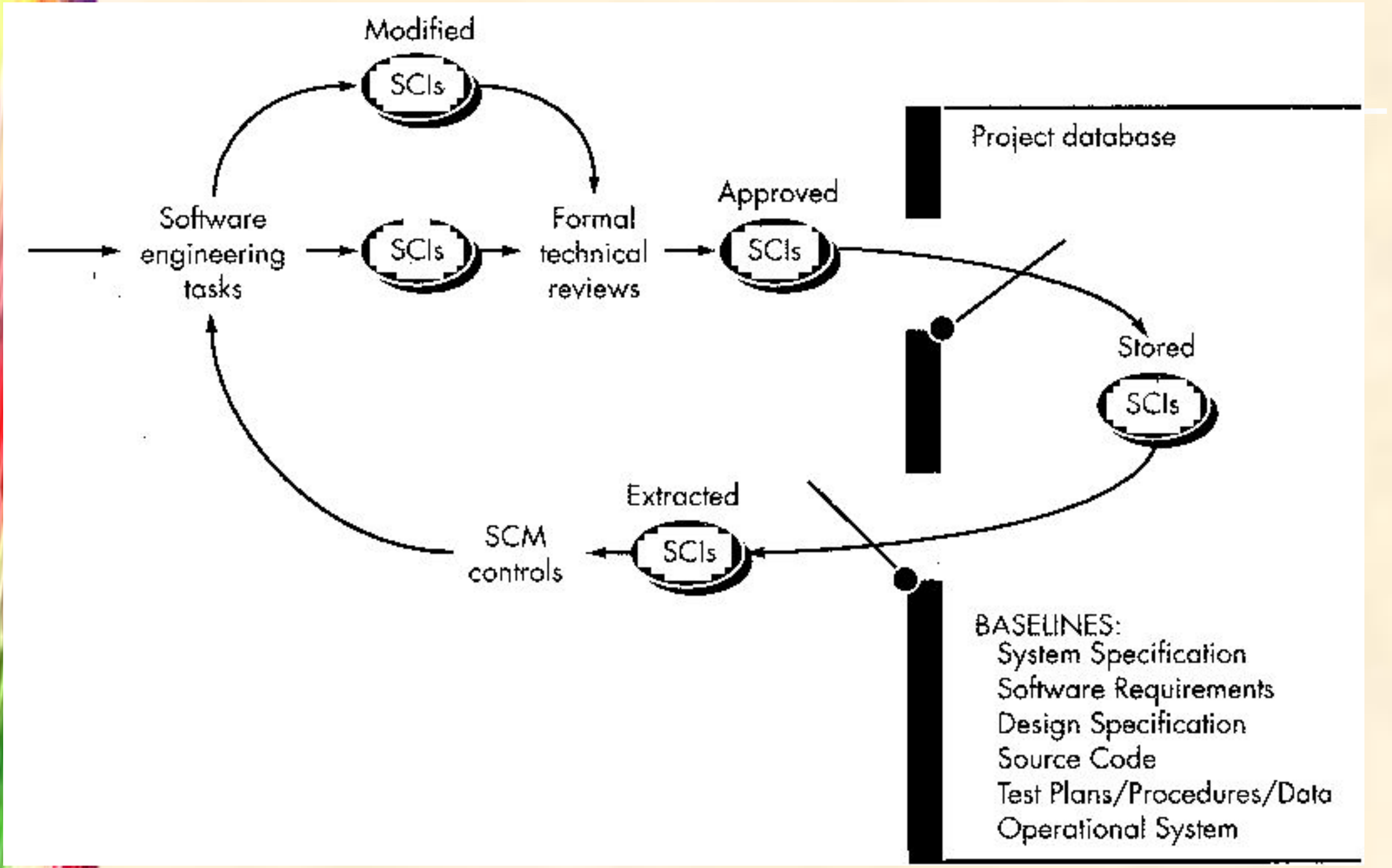
Baseline (Continue)

For Example:

The elements of Design Specification have been documented and reviewed. Errors are found & corrected. Once all parts of the specification have been reviewed, corrected and then approved, the Design specification becomes a baseline.

Further changes to the Design specification can be made only after each has been evaluated and approved.

Baselined SCIs & the project database.



Software Configuration Management

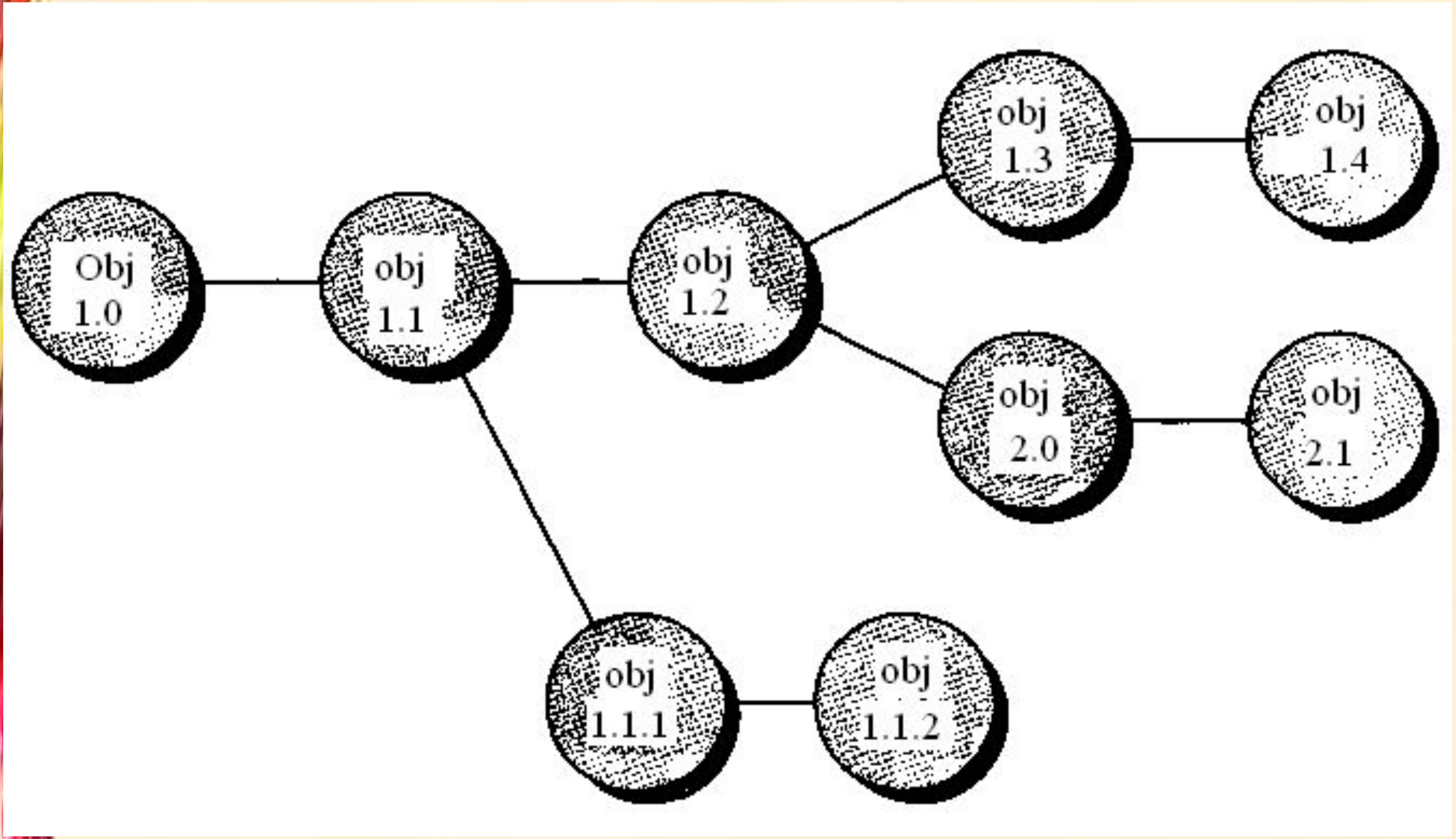
- It is an important element of software quality assurance.
- Its primary responsibility is
  - Identification (tracking multiple versions to enable efficient changes)
  - Version control (control changes before and after release to customer)
  - Change control (authority to approve and prioritize changes)
  - Configuration auditing (ensure changes made properly)
  - Reporting (tell others about changes made)

Software Configuration Objects

- To control and manage configuration items, each must be named and managed using an object-oriented approach
- Basic objects are created by software engineers during analysis, design, coding, or testing
- Aggregate objects are collections of basic objects and other aggregate objects
- Configuration object attributes: unique name, description, list of resources, and a realization
- An entity-relationship (E-R) diagram can be used to show the interrelationships among the objects

The Evolution Graph

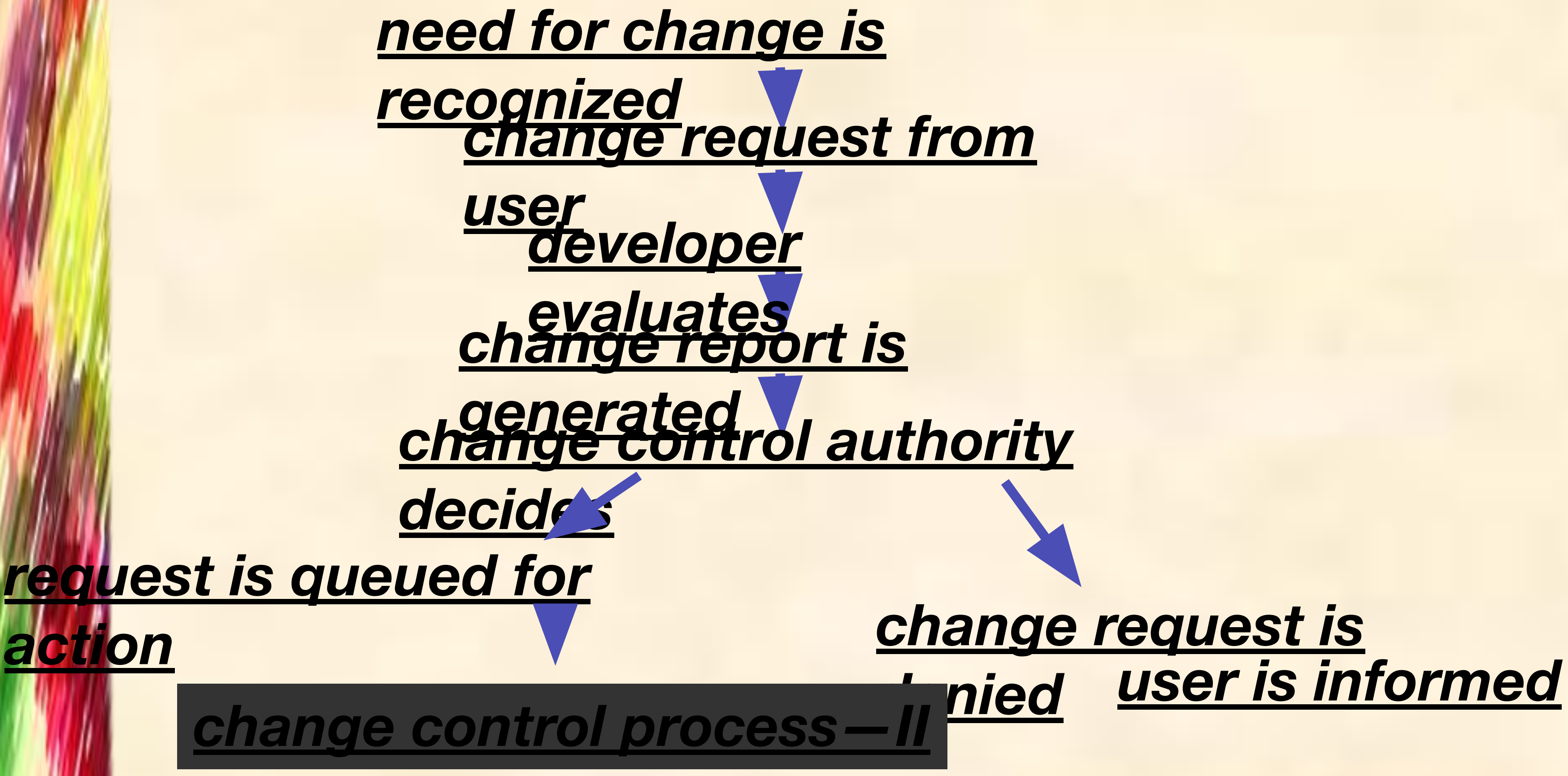
- The evolution graph describe the change history of an object.



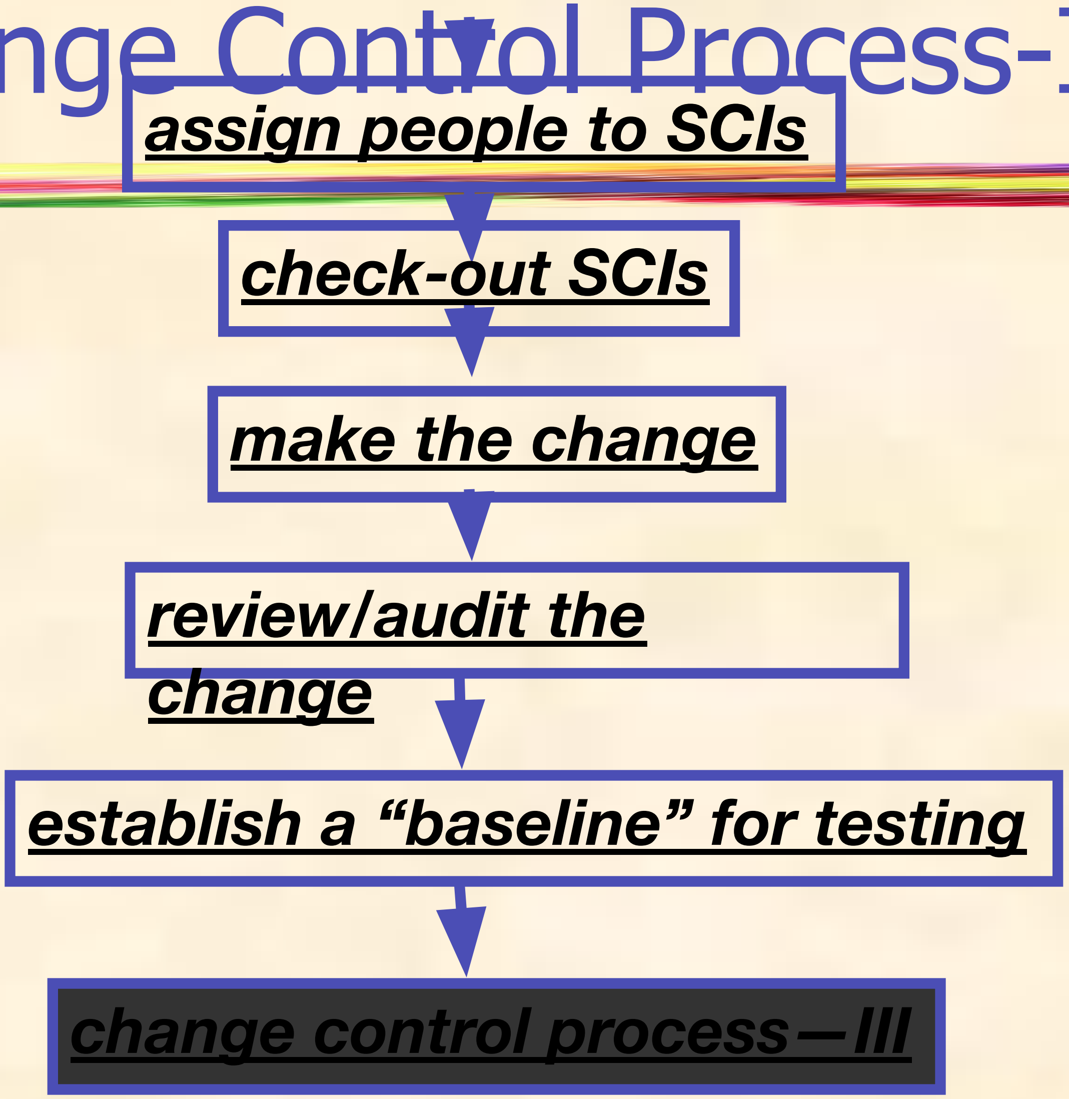
Version Control

- Combines procedures and tools to manage the different versions of configuration objects created during the software process.
- Configuration management allows a user to specify alternative configuration of the software system through the selection of appropriate versions.
- The evolution graph can be used to describe different versions of a system.
- Each version of the software is a collection of SCIs.

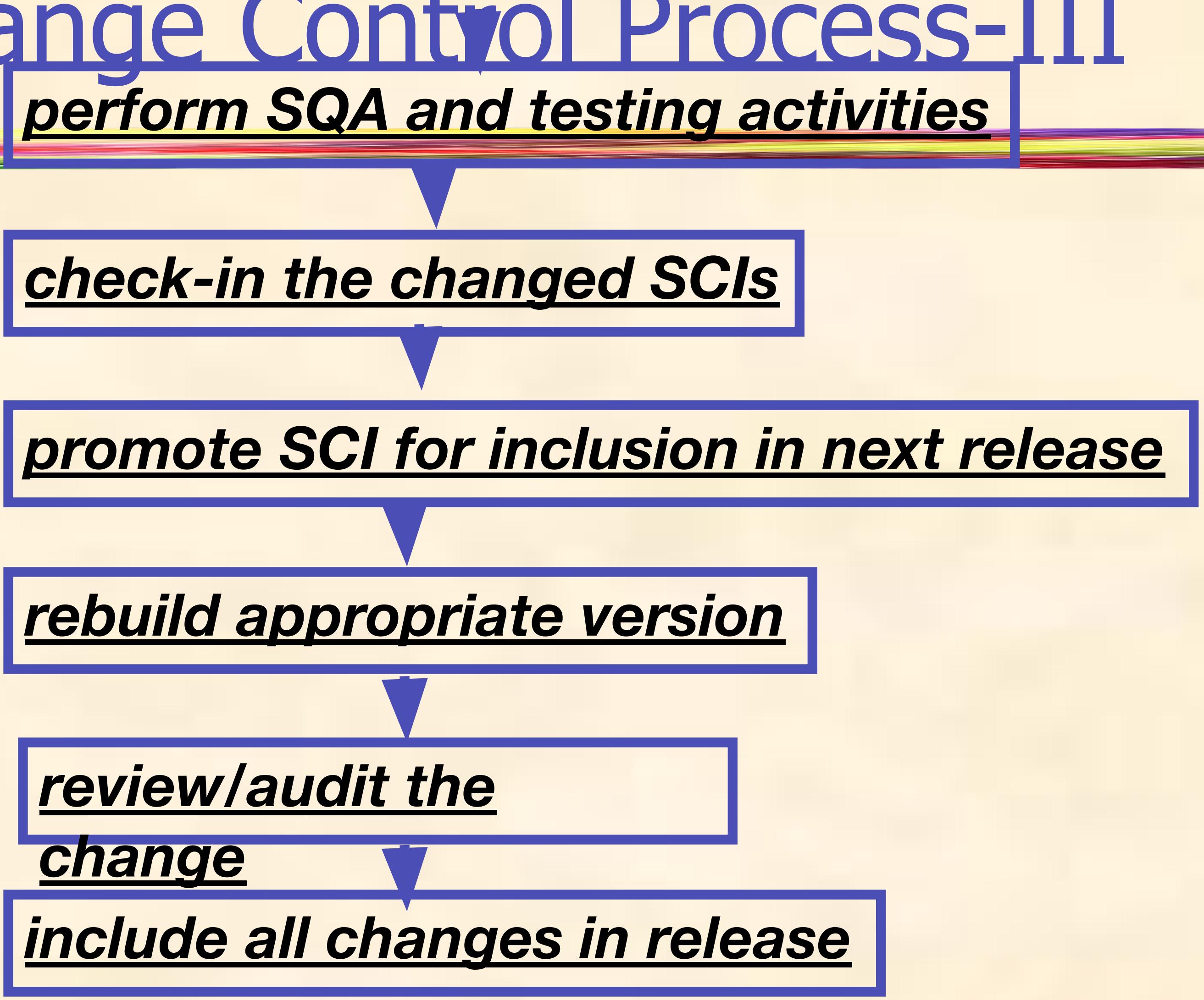
Change Control Process—I


need for change is recognized
change request from user
developer evaluates
change report is generated
change control authority decides
request is queued for action
change request is denied
user is informed
change control process—II

Change Control Process-II



Change Control Process-III


perform SQA and testing activities
check-in the changed SCIs
promote SCI for inclusion in next release
rebuild appropriate version
review/audit the change
include all changes in release

Configuration audit

- ***How to ensure change has been properly implemented.***
  - ***Formal Technical review***
  - ***Software Configuration audit.***
- ***The FTR focuses on the technical correctness of the configuration object has been modified.***
- ***In FTR, the reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects.***

Configuration audit (Continue)

- **The audit asks and answers the following questions:**
  - **Has the change specified by the ECO (Engineering change order) been made without modifications?**
  - **Has an FTR been conducted to assess technical correctness?**
  - **Was the software process followed and software engineering standards applied?**
  - **Have the SCM standards for recording and reporting the change been followed?**
  - **Were all related SCI's properly updated?**

Configuration Status Reporting

Configuration status reporting (or status accounting) is an SCM task that answers following questions:

- **What happened?**
- **Who did it?**
- **When did it happen?**
- **What else will be affected by the change?**

Configuration Status Reporting contd..

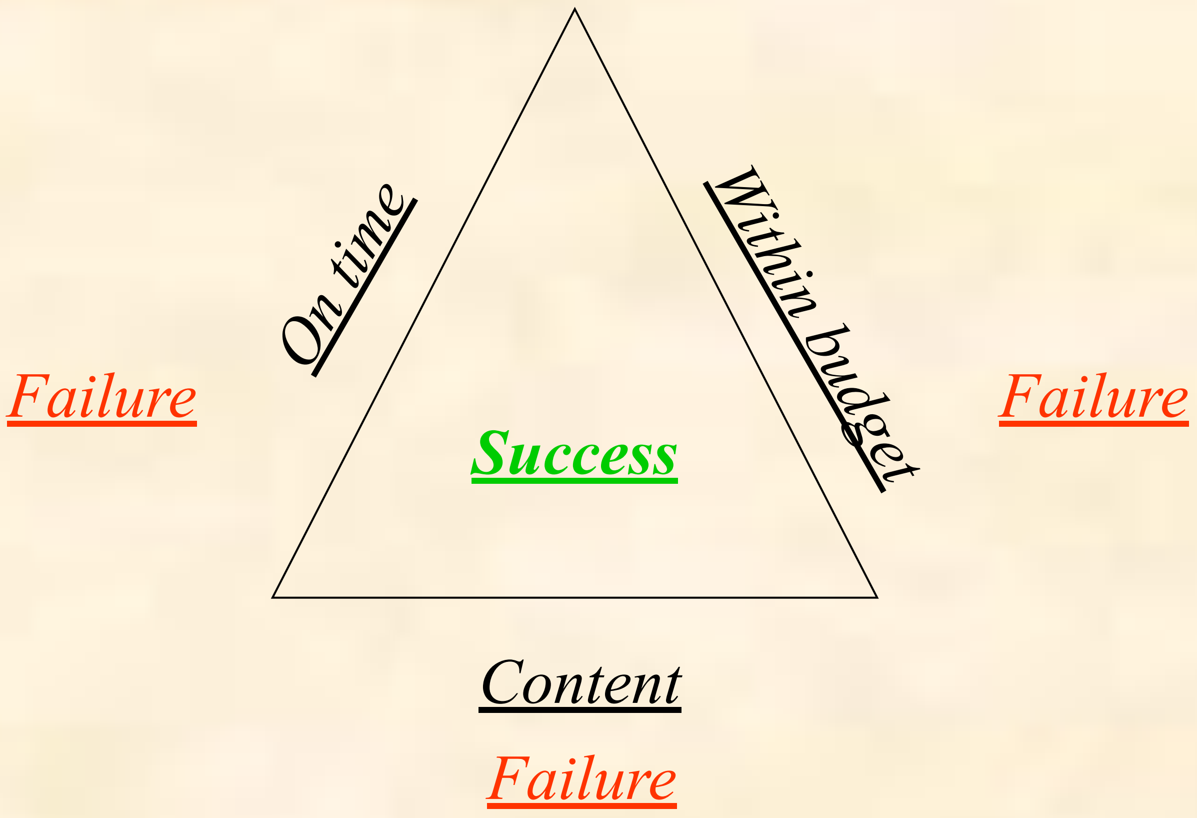
- each time SCI is assigned a new or updated identification , A CSR entry is made.
- After CA. results are reported as a part of the CSR task.

What is Quality Assurance ?

- Quality =
  - ...meeting the customer's requirements,
  - ...at the agreed cost,
  - ...within the agreed timescales.
- Quality = "Fitness for purpose"
- Quality = Customer satisfaction !

“Success” v “Failure”

Quality Assurance helps us avoid failure !



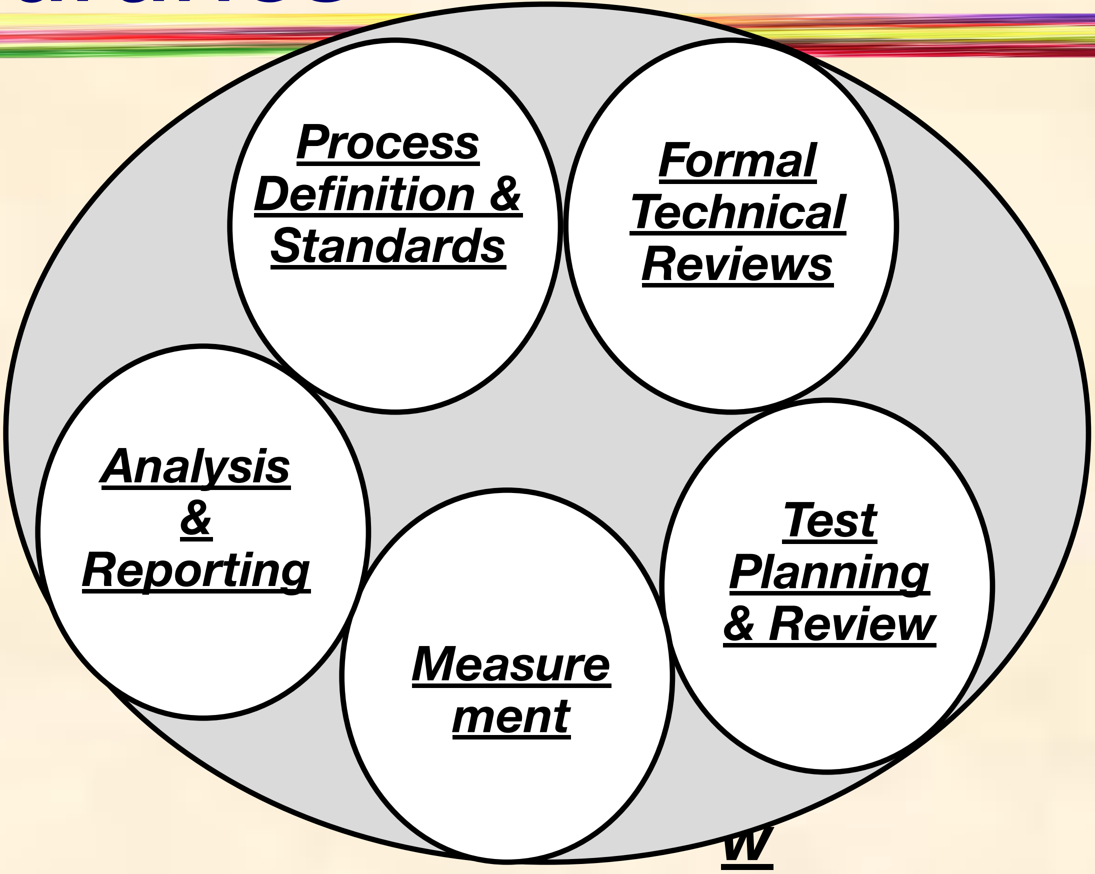
Quality Concepts

- **Quality control:** a series of inspections, reviews, tests
- **Quality assurance:** analysis, auditing and reporting activities
- **cost of quality**

Software Quality Assurance

- **Software quality is defined as :**
- **conformance to explicitly stated functional and performance requirements,**
- **explicitly document development standards**
- **and implicit characteristics that are expected of all professionally developed software.**

Software Quality Assurance



Software Quality Assurance

1. **SQA is an umbrella activity that is applied throughout the software process.**
2. **SQA encompasses :**
   1. **A quality management approach,**
   2. **Effective software engineering technology,**
   3. **Formal Technical Reviews that are applied throughout the software process,**
   4. **A multitier testing strategy,**
   5. **Control of software documentation & the change made to it,**
   6. **A procedure to ensure compliance with software development standards (when applicable) and**
   7. **Measurement & reporting mechanisms**

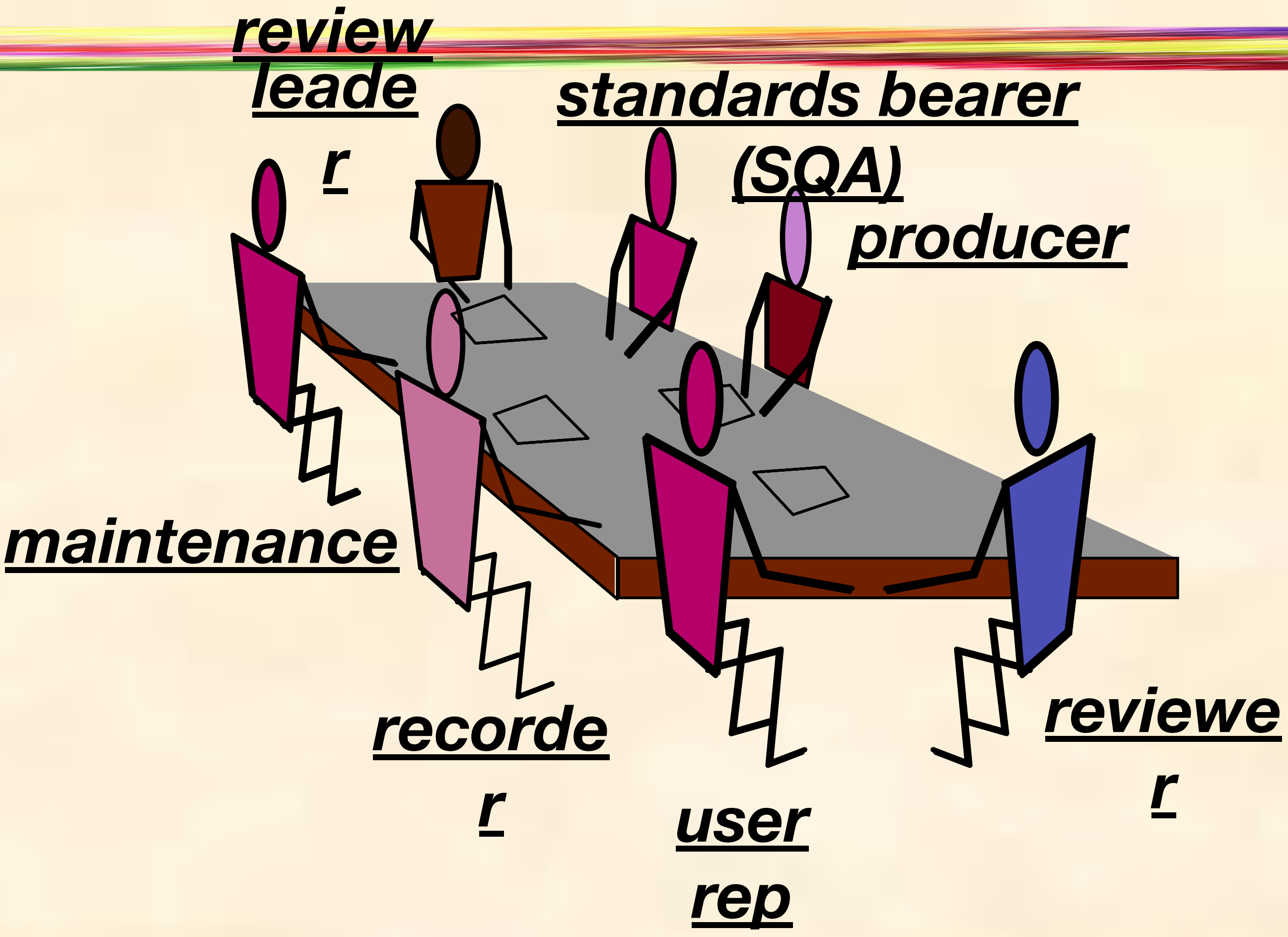
SQA ACTIVITIES

1. Prepare an SQA plan for a project
   - evaluations to be performed
   - Audits and reviews to be performed
   - Documents to be produced by the SQA group
   - procedures for error reporting and tracking
2. Participates in the development of the project's software process description.
3. Audits designated software work products to verify compliance with those defined as part of the software process.
4. Reviews software engineering activities to verify compliance with the defined software process
5. Ensures that deviations in software work and work products are documented and handled according to a documented procedure.

What Are Reviews?

- a meeting conducted by technical people for technical people
- a technical assessment of a work product created during the software engineering process
- a software quality assurance mechanism

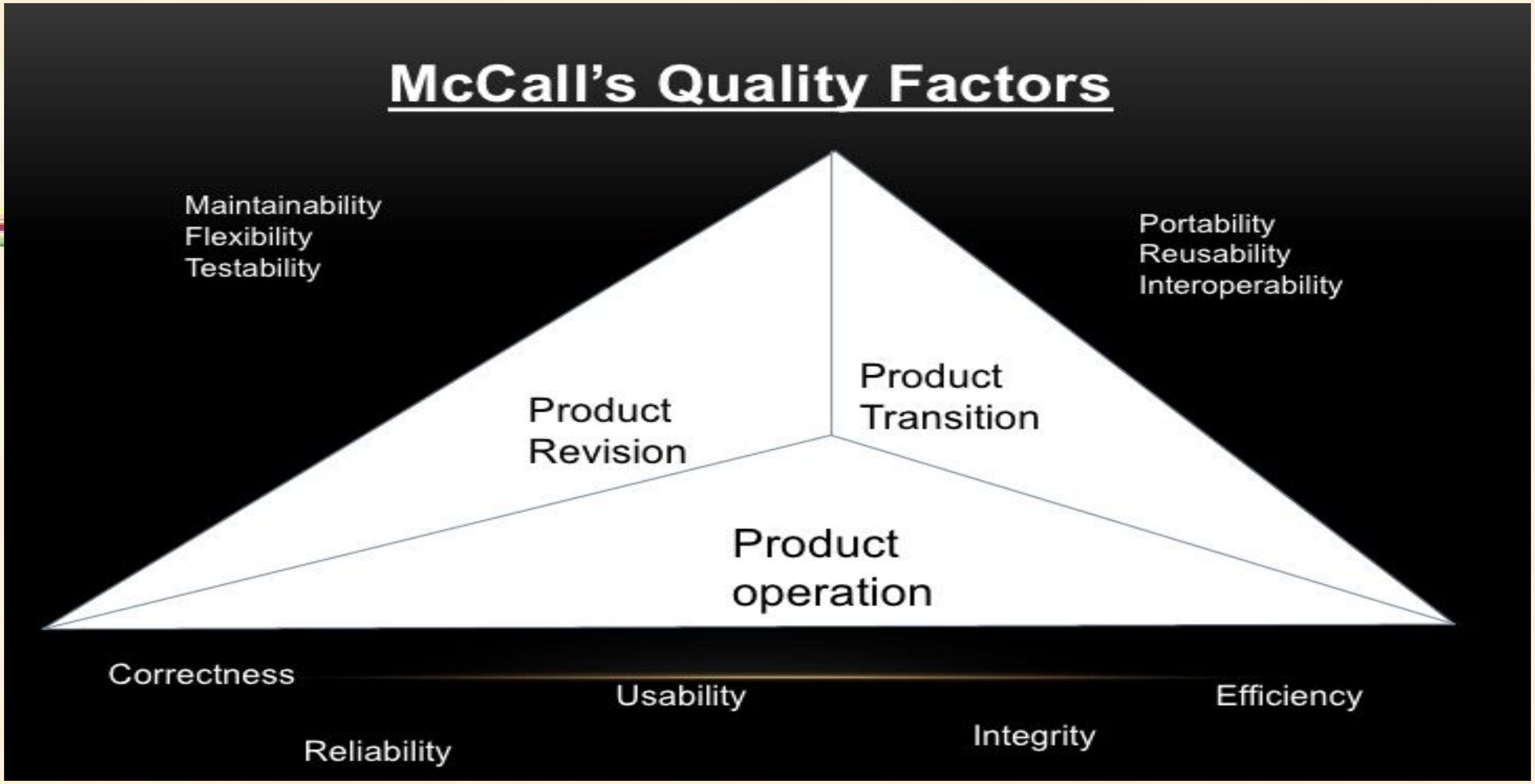
The Players



review leader
standards bearer (SQA)
producer
maintenance
recorder
user rep
reviewer

Conducting the Review

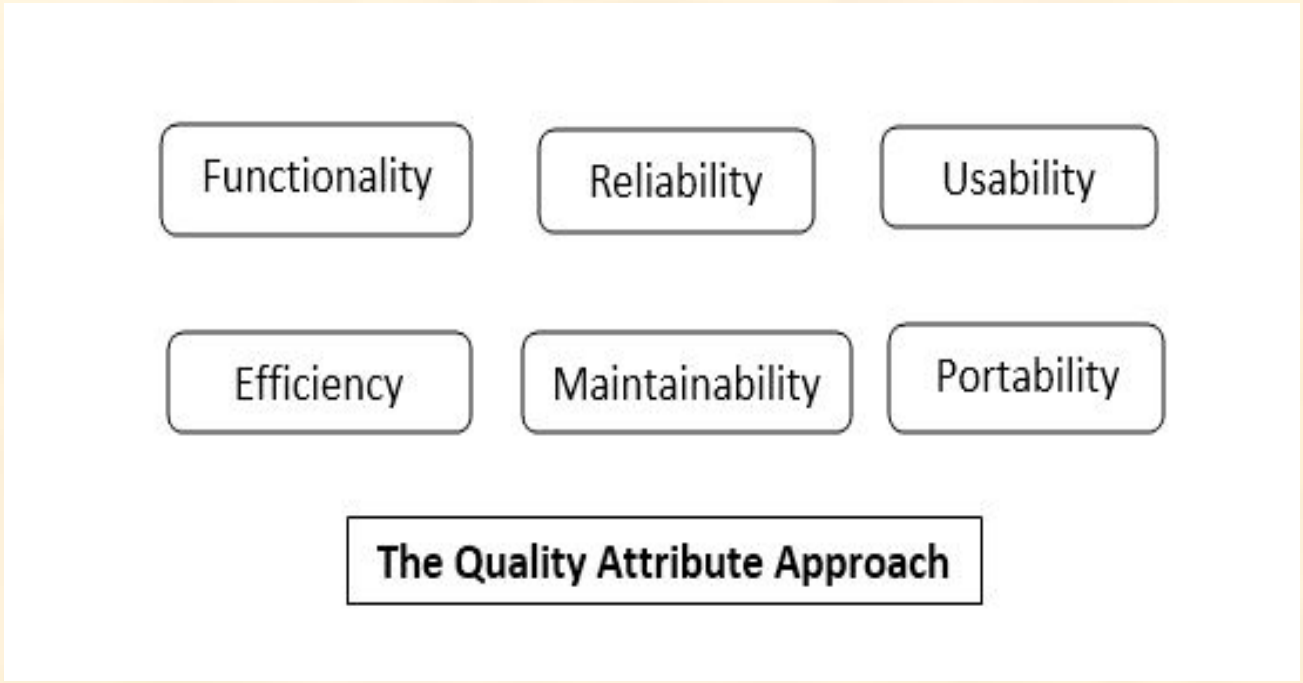
1. be prepared—evaluate product before the review
2. review the product, not the producer
3. keep your tone mild, ask questions instead of making accusations
4. stick to the review agenda
5. raise issues, don't resolve them
6. avoid discussions of style—stick to technical correctness
7. schedule reviews as project tasks
8. record and report all review results



- Product revision (ability to change).
- Product transition (adaptability to new environments).
- Product operations (basic operational characteristics)

The SQA Attribute

- There is a list of attributes which describes the step by step approach to obtain Software Quality Assurance. The attributes are given as in the diagram below:



The SQA Attribute contd..

- **Functionality: The attributes considers the set of all the functions used in the software.**
  - **Suitability:** Ensures the functions of the software are appropriate.
  - **Accuracy:** Ensures the accurate usage of the functions.
  - **Interoperability:** Ensure the effective interaction of the software with other components.
  - **Security:** Ensure the software is capable of handling any security issues

The SQA Attribute contd..

- **Reliability:** The purpose of the attribute is to check the capability of the system to perform without delay during any conditions
  - Maturity: Less possibility of failure of the software in any activities.
  - Recoverability: The rate of recovery ability once a failure occurs.
- A simple measure of reliability is mean time between failure (MTBF), where

  MTBF = MTTF + MTTR

  MTTF = Mean time to failure.
  MTTR = Mean time to repair.

Software Availability

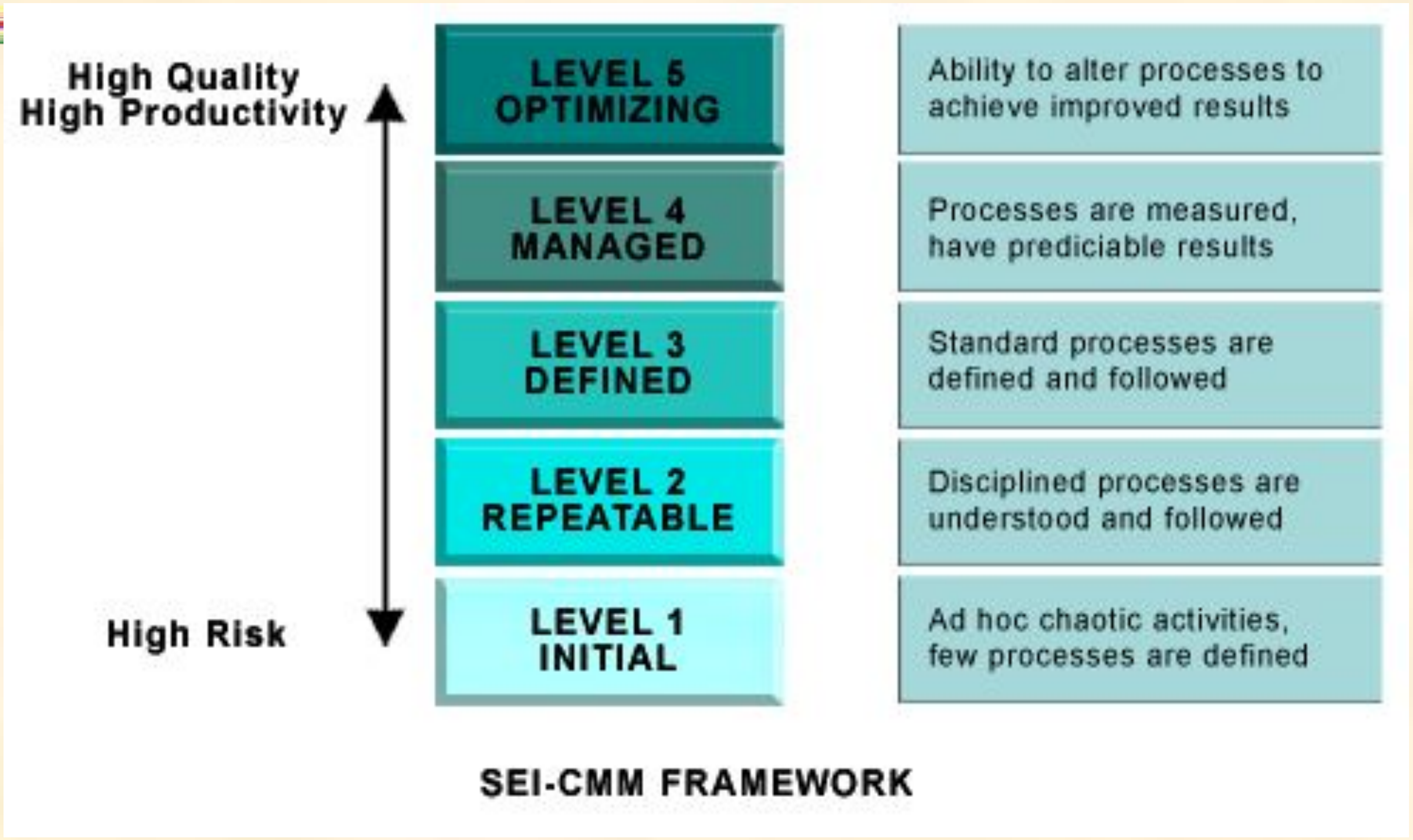
- Software Availability is the probability that a program is operating according to requirements at a given point in time and is defined as

  Availability = [MTTF / (MTTF + MTTR)] * 100%
- **Usability:** The purpose is to ensure the use of a function
  - Understandability: How much effort a user needs to understand the functions.

The SQA Attribute contd..

- **Maintainability:** The way to analyze and fix a fault/issue in the software
  - Analyzability: Finding out the cause of failure.
  - Changeability: How the system response to necessary changes.
  - Stability: How stable the system is when the changes made.
  - Testability: Testing efforts
- **Adaptability:** Ability of the system to adopt the changes in its environment.

Capability Maturity Model


High Quality
High Productivity
High Risk
LEVEL 5
OPTIMIZING
LEVEL 4
MANAGED
LEVEL 3
DEFINED
LEVEL 2
REPEATABLE
LEVEL 1
INITIAL
Ability to alter processes to achieve improved results
Processes are measured, have prediciable results
Standard processes are defined and followed
Disciplined processes are understood and followed
Ad hoc chaotic activities, few processes are defined
SEI-CMM FRAMEWORK

CMM

- The CMM is organized into five maturity levels:

1) Initial. At this level, an organization does not have effective management procedures or project plans. The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2) Repeatable. At this level, an organization has formal management, quality assurance and configuration control procedures in place. It is called the repeatable level because the organization can successfully repeat projects of the same type.

3) Defined. At this level, an organization has defined its process and thus has a basis for qualitative process improvement. Formal procedures are in place to ensure that the defined process is followed in all software projects.

CMM

4) **Managed.** At this level organization has a defined process and a formal programme of quantitative data collection. Process and product metric are collected and fed into the process improvement activity.

5) **Optimizing.** At this level, an organization is committed to continuous process improvement. Process improvement is budgeted and planned and is an integral part of the organization's process.

ISO 9000 Quality Standards

- ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of product or services offered.
- 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.

ISO 9000 Quality Standards

- ISO 9000 describes the elements of quality assurance system in general terms.
- These elements include the organizational structure, procedure, processes and resources needed to implement quality planning, quality control, quality assurance and quality improvement.
- However, ISO 9000 does not describe how an organization should implement these quality system elements.

The ISO 9001 Standard

- ISO 9001 is the quality assurance standard that applies to software engineering.
- The standard contains 20 requirements that must be present for an effective quality assurance system. [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 and statistical techniques.]