Unit 5

Software Evolution as Entity

- **Software Evolution** is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc.

Software Evolution as Entity

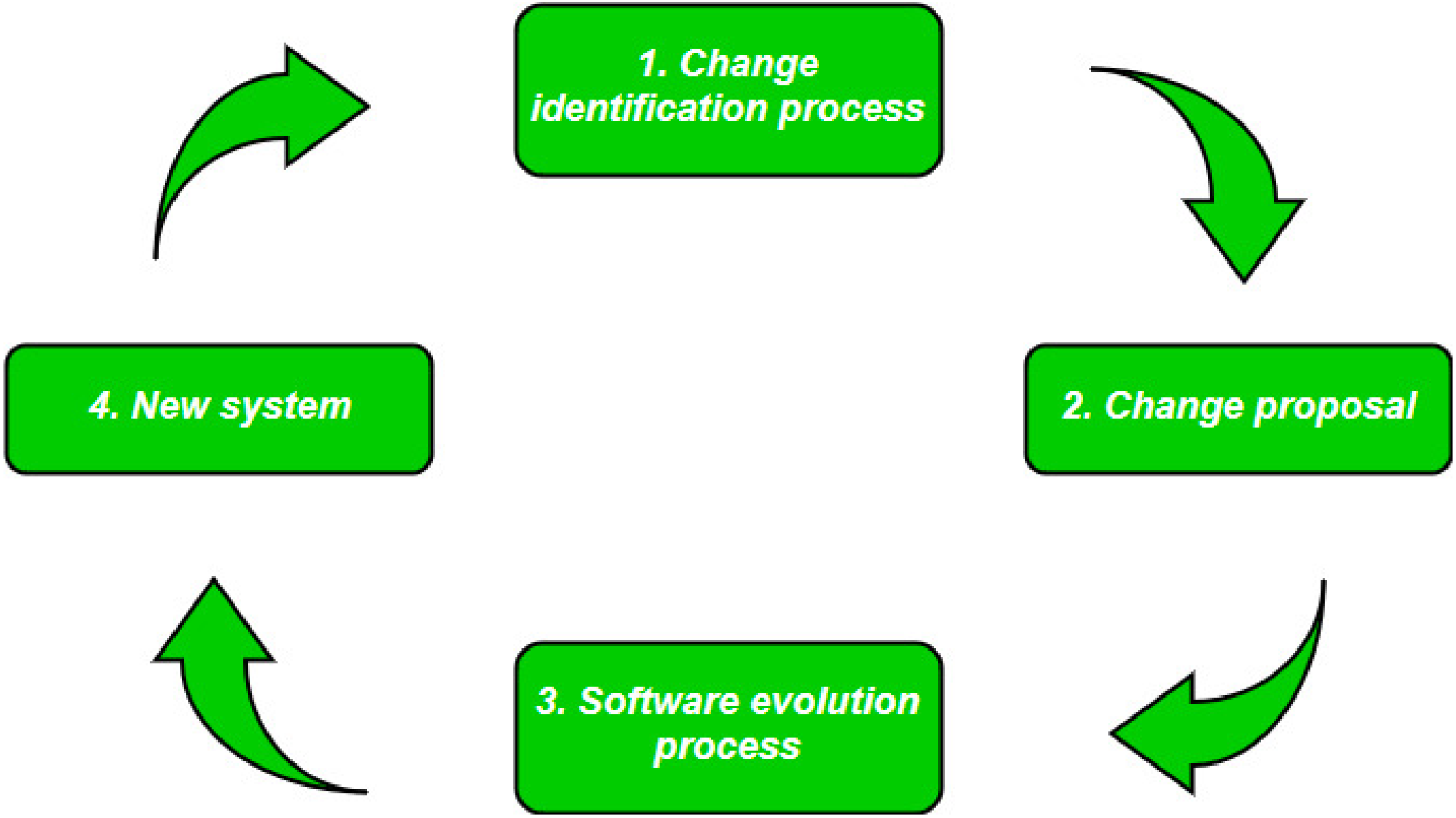
- The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.
- The cost an impact of these changes are accessed to see how much system is affected by the change and how much it might cost to implement the change.

Software Evolution as Entity

- If the proposed changes are accepted, a new release of the software system is planned.
- During release planning, all the proposed changes (fault repair, adaptation, and new functionality) are considered.

Software Evolution as Entity

- A design is then made on which changes to implement in the next version of the system.
- The process of change implementation is an iteration of the development process where the revisions to the system are designed, implemented and tested.


1. Change identification process
2. Change proposal
3. Software evolution process
4. New system

Software Maintenance

- Software Maintenance is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities, and optimization.
- The main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.

Need for Maintenance

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

Categories of Software Maintenance

- Corrective Maintenance
- Adaptive Maintenance
- Preventive Maintenance
- Perfective Maintenance

Types of Maintenance

- **Corrective Maintenance:** A software product is necessary to rectify the bugs observed while the system is in use or to enhance the performance of the system.
- **Adaptive Maintenance**:
  - Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.

Types of Maintenance

- This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.
- Adaptive maintenance is generally not requested by the client, rather it is imposed by the external environment

Types of Maintenance (Contd.)

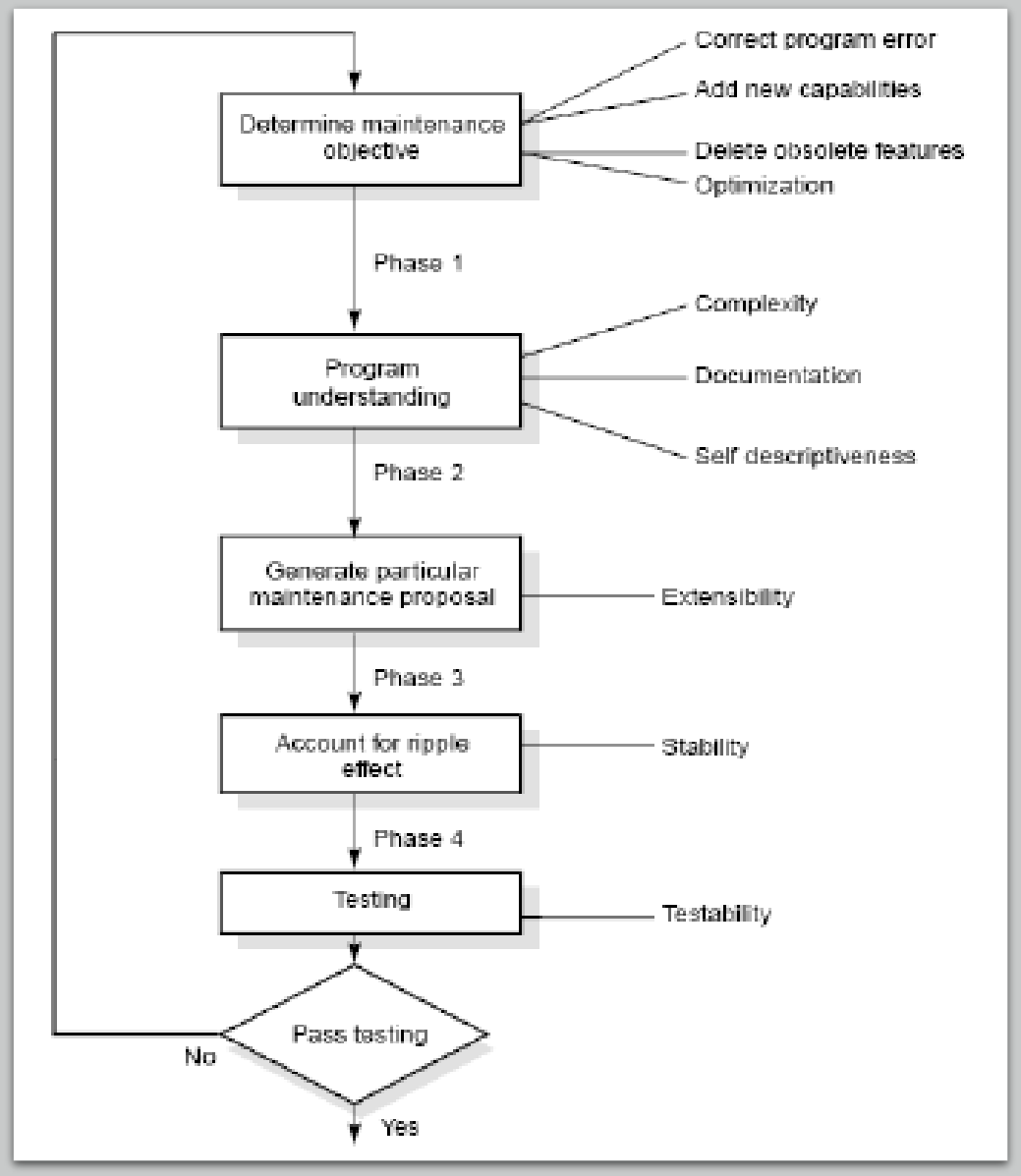
- Preventive Maintenance
  - This type of maintenance includes modifications and updations to prevent future problems of the software.
  - It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

Types of Maintenance (Contd.)

- Perfective Maintenance
  - A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

Causes of Software Maintenance

- Lack of Traceability: Codes are not traceable according to requirement and design specification.
- Lack of Code Comments
- Lack of Domain Knowledge
- Staff Problem due to limited understanding



Maintenance Process

Maintenance Process Contd.

Program Understanding –

- This is the first phase. Analyze the program in order to understand it.
- Attributes such as complexity of the program, the documentation, and the self- descriptiveness of the program contribute to the ease of understanding the program.

Maintenance Process Contd.

- The complexity is a measure of the effort required to understand the program and is usually based on the control or data flow of the program.

Maintenance Process Contd.

Generating Maintenance Proposal

- This is the 2nd phase consists of generating a particular maintenance proposal to accomplish the implementation of the maintenance objective.
- The ease of generating maintenance proposal for a program is primarily affected by the attribute extensibility.
- The extensibility of the program is a measure of the extent to which the program can support extensions of critical functions.

Maintenance Process Contd.

Ripple Effect

- The 3rd phase consists of accounting for all of the ripple effect as a consequence of program modification. In software, the effect of a modification may not be local to the modification, but may also affect other portions of the program. There is a ripple effect from the location of the modification to the other parts of the program that are affected by the modification.

One aspect if this effect is Logical or functional

Maintenance Process Contd.

Second is performance requirements

Modified Program Testing

- The 4th phase consists of testing the modified program to ensure that the modified program has at least the reliability level as before.

Cost of Maintenance

- Models for maintenance cost estimation
  - Belady and Lehman Model
  - Boehm Model.

Belady and Lehman Model

- Lehman and Belady have studied the characteristics of evolution of several software products [1980]. They have expressed their observations in the form of laws.
- **Lehman's first law:** A software product must change continually or become progressively less useful. Every software product continues to evolve after its development through maintenance efforts.

Belady and Lehman Model

- **Lehman's second law:** The structure of a program tends to degrade as more and more maintenance is carried out on it. The reason for the degraded structure is that when you add a function during maintenance, you build top of an existing program, often in a way that the existing program was not intended to support. If you do not redesign the system, the additions will be more complex that they should be.

Belady and Lehman Model

- **Lehman's third law:** Over a program's lifetime, its rate of development is approximately constant.
- The rate of development can be quantified in terms of the lines of code written or modified.
- Therefore, this law states that the rate at which code is written or modified is approximately the same during development and maintenance.

Belady and Lehman Model

- This model indicates that the effort and cost can increase exponentially if poor software development approach is used and the person or group that used the approach is no longer available to perform maintenance.
- The basic equation is

$$M = P + Ke^{(c-d)}$$

Where

M – Total effort expended

P – Productive effort

K – An empirically determined constant

c – Complexity measure due to lack of good design and documentation

d – Degree to which maintenance team is familiar with the software.

Belady and Lehman Model

Example – The development effort for a software project is 500 person-months. The empirically determined constant K is 0.3. The complexity of the code is quite high and is equal to 8. calculate the total effort expended (M) if

i) Maintenance team has good level of understanding of the project (d = 0.9)

ii) Maintenance team has poor understanding of project (d = 0.1)

Belady and Lehman Model

Solution –

Development effort (P) = 500 pm

K = 0.3

c = 8

i) d = 0.9

$M = P + Ke^{(c-d)}$

$= 500 + 0.3e^{(8-0.9)} = 863.59$ pm

i) d= 0.1

$M = P + Ke^{(c-d)}$

$= 500 + 0.3e^{(8-0.1)} = 1309.18$ pm

Belady and Lehman Model

- The development effort for a project is 600 PMs. The empirically determined constant (K) of Belady and Lehman model is 0.5. The complexity of code is quite high and is equal to 7. Calculate the total effort expended (M) if maintenance team has reasonable level of understanding of the project (d=0.7).

Boehm Model

Boehm used a quantity called Annual Change Traffic(ACT) which is defined as

"The fraction of a software products source instructions which undergo change during a year either through addition, deletion or modification".

$$ACT = (KLOC_{added} + KLOC_{deleted}) / KLOC_{total}$$

The Annual Maintenance Effort (AME) in person-month can be calculated as:

$$AME = ACT * SDE$$

Where SDE – Software development effort in person-month

ACT – Annual Change Traffic

Bohem Model

The formula can be modified further by using some Effort Adjustment Factors(EAF).

The modified equation is given as :

AME = ACT * SDE * EAF

Bohem Model

Example – Annual change traffic (ACT) for a software system is 15 % per year. The development effort is 600 pms. Compute an estimate for annual maintenance effort(AME). If life time of the project is 10 years, what is the total effort of the project.

Soln – AME = ACT * SDE

= 0.15 * 600 = 90 pm

For 10 years AME= 10 * 90 = 900pm

Total effort = 600 + 900 = 1500 pm.

Reverse Engineering

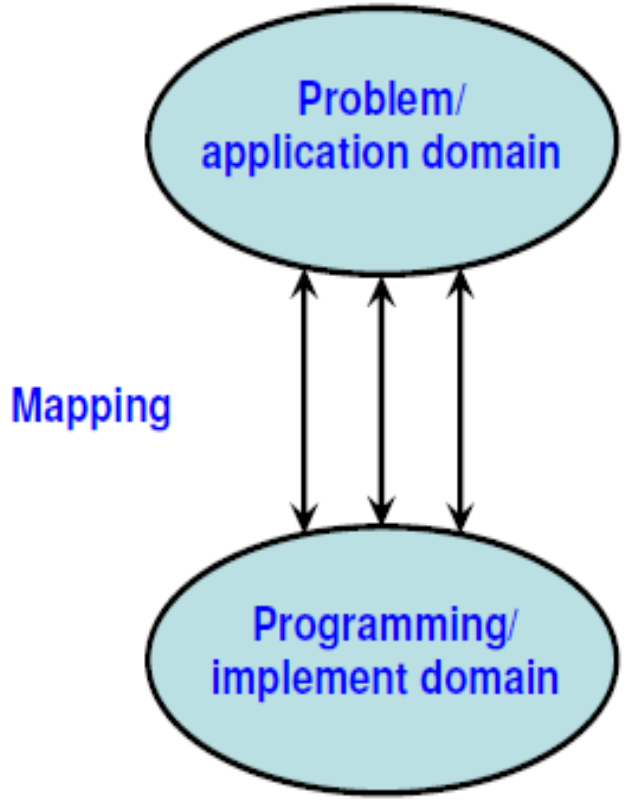
- Reverse engineering is the process followed in order to find difficult, unknown and hidden information about a software system.
- The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Reverse Engineering

- Scope and Tasks
- The areas there reverse engineering is applicable include (but not limited to):
  1. Program comprehension
  2. Redocumentation and/ or document generation
  3. Recovery of design approach and design details at any level of abstraction
  4. Identifying reusable components
  5. Identifying components that need restructuring
  6. Recovering business rules, and
  7. Understanding high level system description

Reverse Engineering

- Reverse Engineering encompasses a wide array of tasks related to understanding and modifying software system. This array of tasks can be broken into a number of classes.
- Mapping between application and program domains
- Mapping between concrete and abstract levels
- Rediscovering high level structures
- Finding missing links between program syntax and semantics
- To extract reusable component

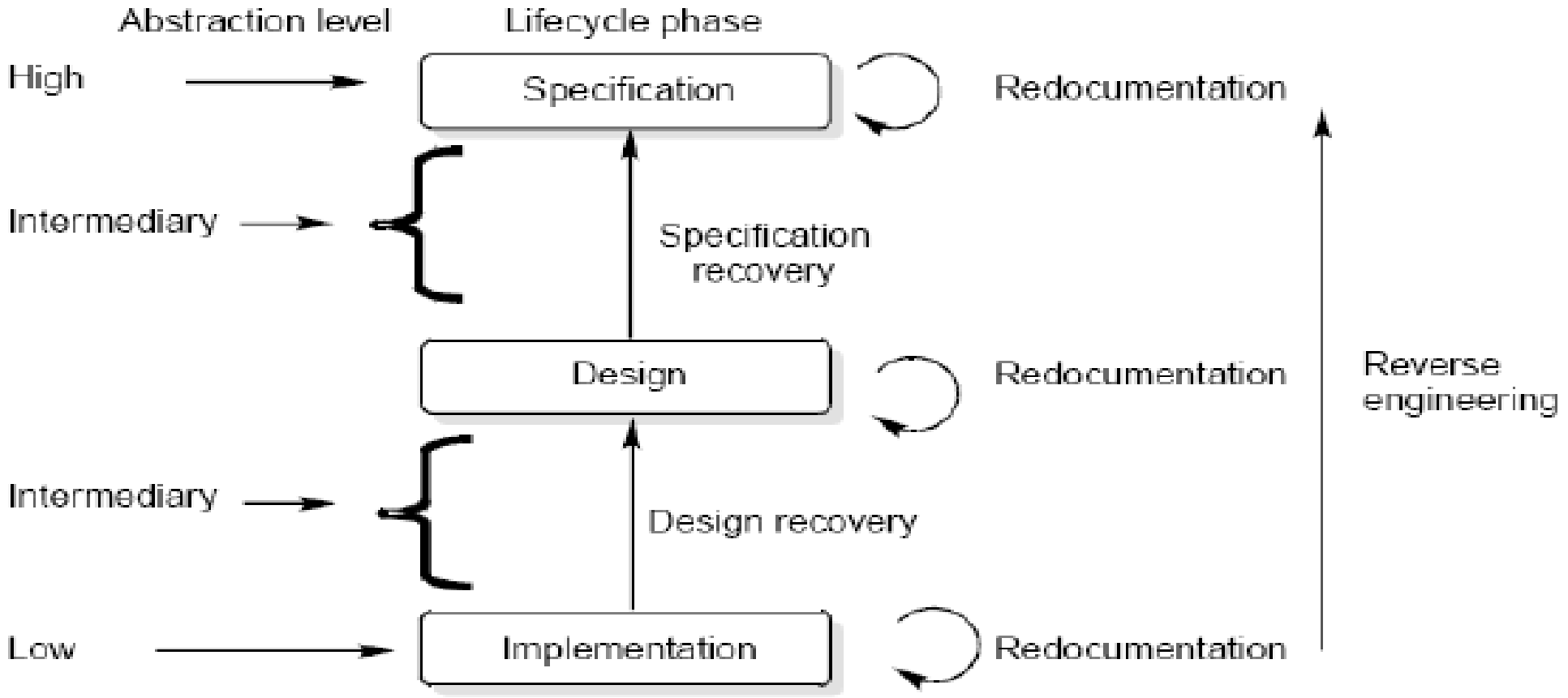


Mapping between application and domains program

Reverse Engineering

- Levels of Reverse Engineering
  - Reverse Engineers detect low level implementation constructs and replace them with their high level counterparts.
  - The process eventually results in an incremental formation of an overall architecture of the program.

Reverse Engineering: Levels of Abstraction



Reverse Engineering

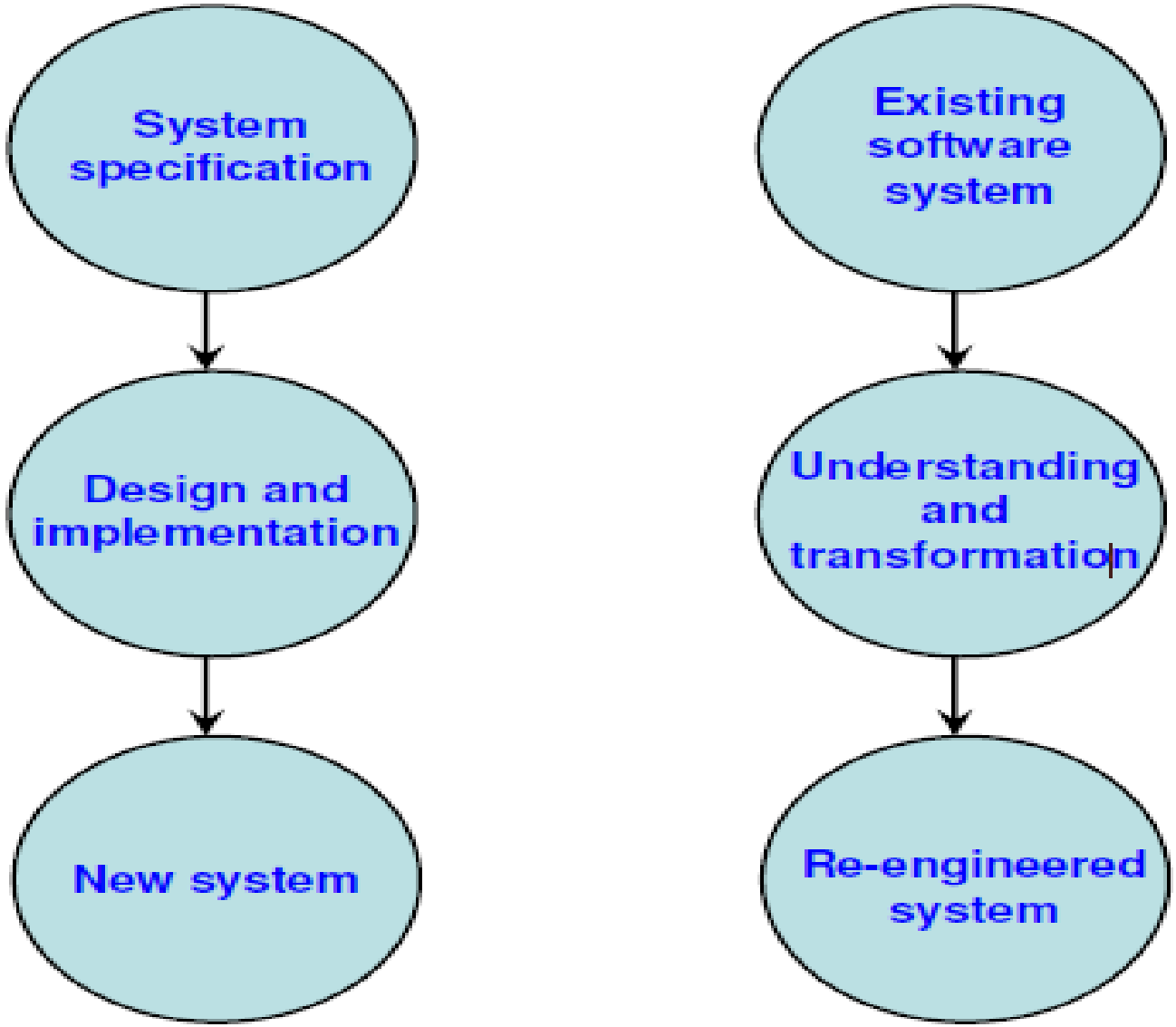
- Redocumentation
  - Redocumentation is the recreation of a semantically equivalent representation within the same relative abstraction level.

Reverse Engineering

- Design recovery
  - Design recovery entails identifying and extracting meaningful higher- level abstractions beyond those obtained directly from examination of the source code.
  - This may be achieved from a combination of code, existing design documentation, personal experience, and knowledge of the problem and application domains.

Re-Engineering

- Software re-engineering is concerned with taking existing legacy systems and re-implementing them to make them more maintainable.
- The critical distinction between re-engineering and new software development is the starting point for the development



Comparison of new software development with re-engineering

Re-Engineering

- The following suggestions may be useful for the modification of the legacy code:
  - Study code well before attempting changes
  - Concentrate on overall control flow and not coding
  - Heavily comment internal code
  - Create Cross References
  - Build Symbol tables
  - Use own variables, constants and declarations to localize the effect
  - Keep detailed maintenance document
  - Use modern design techniques

Re-Engineering

- Source Code Translation
  - **Hardware platform update:** The organization may wish to change its standard hardware platform.

Compilers for the original language may not be available on the new platform.

Re-Engineering

- **Staff Skill Shortages:** There may be lack of trained maintenance staff for the original language. This is a particular problem where programs were written in some non-standard language that has now gone out of general use.
- **Organizational policy changes:** An organization may decide to standardize on a particular language to minimize its support software costs. Maintaining many versions of old compilers can be very expensive.

Re-Engineering

- Program Restructuring
  - **Control flow driven restructuring:** This involves the imposition of a clear control structure within the source code and can be either inter modular or intra modular in nature.
  - **Efficiency driven restructuring:** This involves restructuring a function or algorithm to make it more efficient. A simple example is the replacement of an IF-THEN-ELSE-IF-ELSE construct with a CASE construct.

Re-Engineering

- **Adaption driven restructuring:**
  - This involves changing the coding style in order to adapt the program to a new programming language or new operating environment, for instance changing an imperative program in PASCAL into a functional program in LISP.

Software Configuration Management

- *Software configuration management* (SCM) is an umbrella activity that is 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 changes

SOFTWARE CONFIGURATION MANAGEMENT

- However, there are four fundamental sources of change:
  - New business or market conditions dictate changes in product requirements or business rules.

SOFTWARE CONFIGURATION MANAGEMENT

- 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 engineering team structure.
- Budgetary or scheduling constraints cause a redefinition of the system or product.

SCM Concepts: Baselines

- A *baseline is a software configuration management concept that helps us to control* change without seriously impeding justifiable change. The IEEE (IEEE Std. No. 610.12-1990) defines a baseline as:

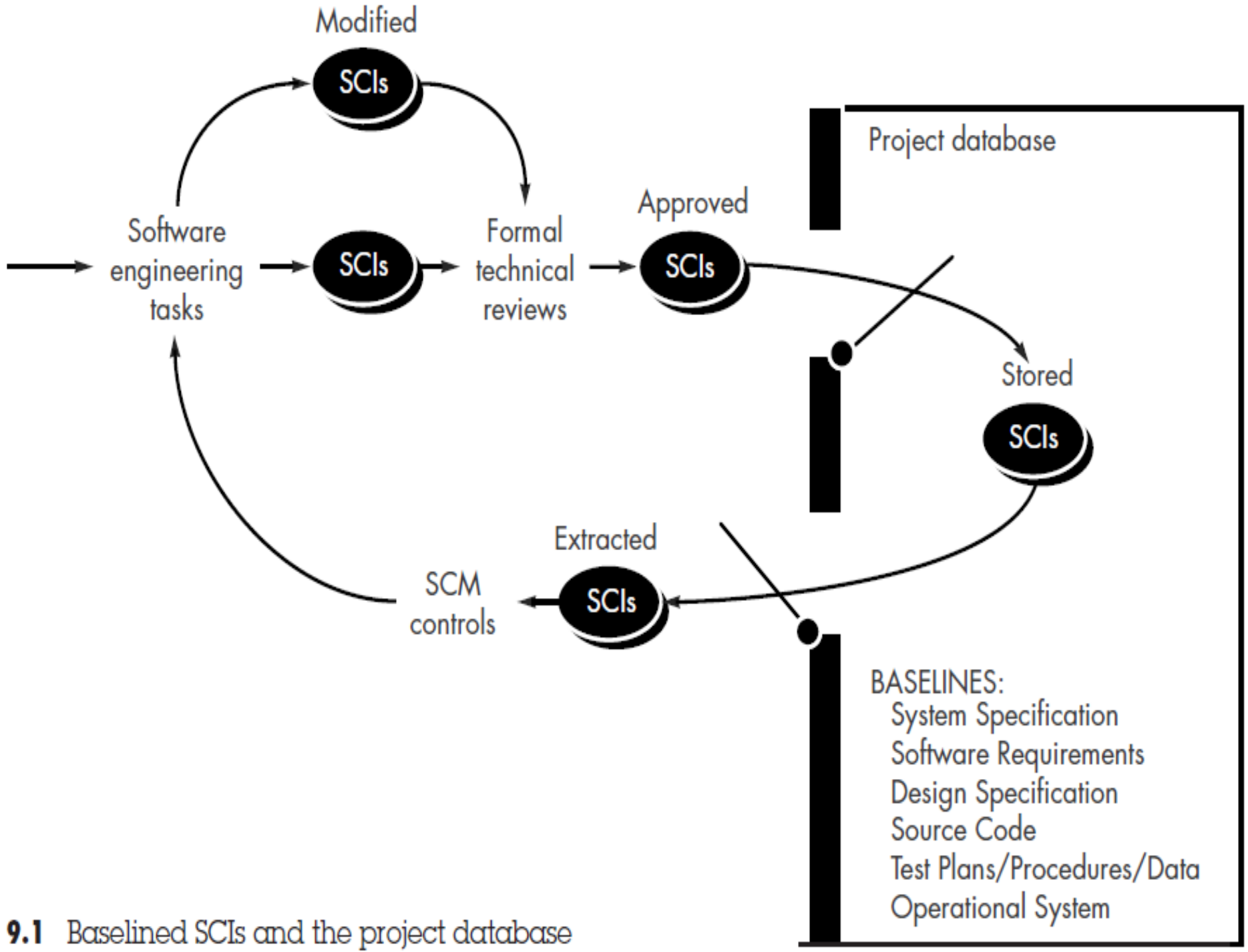
  A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

Baselines

- In the context of software engineering, a baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items (SCIs)` and the approval of these SCIs that is obtained through a formal technical review.
- For example, the elements of a *Design Specification have been* documented and reviewed.

Baselines

- Errors are found and corrected. Once all parts of the specification have been reviewed, corrected and then approved, the *Design Specification* becomes a baseline.
- Further changes to the program architecture (documented in the *Design Specification) can be made only after each has been evaluated and approved.*
- Although baselines can be defined at any level of detail, the most common software baselines are shown in Figure.



E 9.1 Baselined SCIs and the project database

Baselines

- Software engineering tasks produce one or more SCIs.
- After SCIs are reviewed and approved, they are placed in a *project database (also called a project library or software repository).*

Baselines

- *When a member of a software engineering team wants to make a* modification to a baselined SCI, it is copied from the project database into the engineer's private work space.
- However, this extracted SCI can be modified only if SCM controls are followed.
- The arrows in figure illustrate the modification path for a baselined SCI.

SCM Concepts: Software Configuration Items

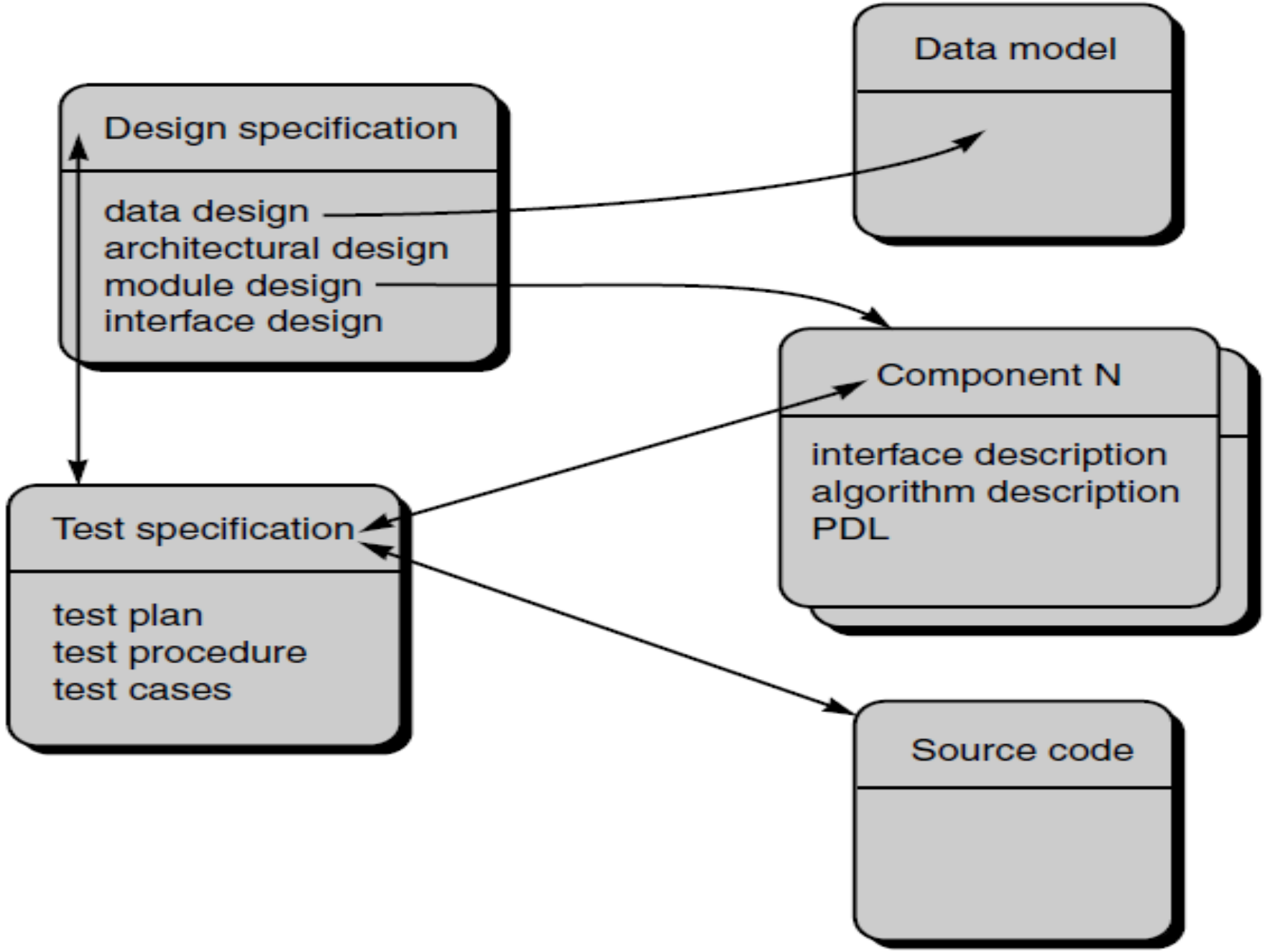
- SCIs are organized to form *configuration objects that may be catalogued* in the project database with a single name.
- A configuration object has a name, attributes, and is "connected" to other objects by relationships.
- Referring to Figure in next slide, the configuration objects, **Design Specification, data model, component N, source code and Test Specification are each defined separately.**

SCM Concepts: Software Configuration Items

- **However, each of the** objects is related to the others as shown by the arrows.

SCM Concepts: Software Configuration Items

- A curved arrow indicates a *compositional relation*. *That is,* ***data model and component N are part of the object*** **Design Specification.**
- **A double-headed straight arrow indicates an interrelationship.** If a change were made to the source code object, the interrelationships enable a software engineer to determine what other objects (and SCIs) might be affected.


Data model
Design specification
data design
architectural design
module design
interface design
Component N
interface description
algorithm description
PDL
Test specification
test plan
test procedure
test cases
Source code

THE SCM PROCESS

- Software configuration management is an important element of software quality assurance.
- Its primary responsibility is the control of change. However, SCM is also responsible for the identification of individual SCIs and various versions of the software, the auditing of the software configuration to ensure that it has been properly developed, and the reporting of all changes applied to the configuration.

THE SCM PROCESS

- Any discussion of SCM introduces a set of complex questions:
  - How does an organization identify and manage the many existing versions of a program in a manner that will enable change to be accommodated efficiently?

THE SCM PROCESS

- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?

THE SCM PROCESS

- How can we ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made?

- These questions lead us to the definition of five SCM tasks: *identification, version control, change control, configuration auditing, and reporting.*

SCM Process: IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

- To control and manage software configuration items, each must be separately named and then organized using an object-oriented approach.

SCM Process: IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

- Two types of objects can be identified : *basic objects* and *aggregate objects.*
- A basic object is a "unit of text" that has been created by a software engineer during analysis, design, code, or test. For example, a basic object might be a section of a requirements specification, a source listing for a component, or a suite of test cases that are used to exercise the code.

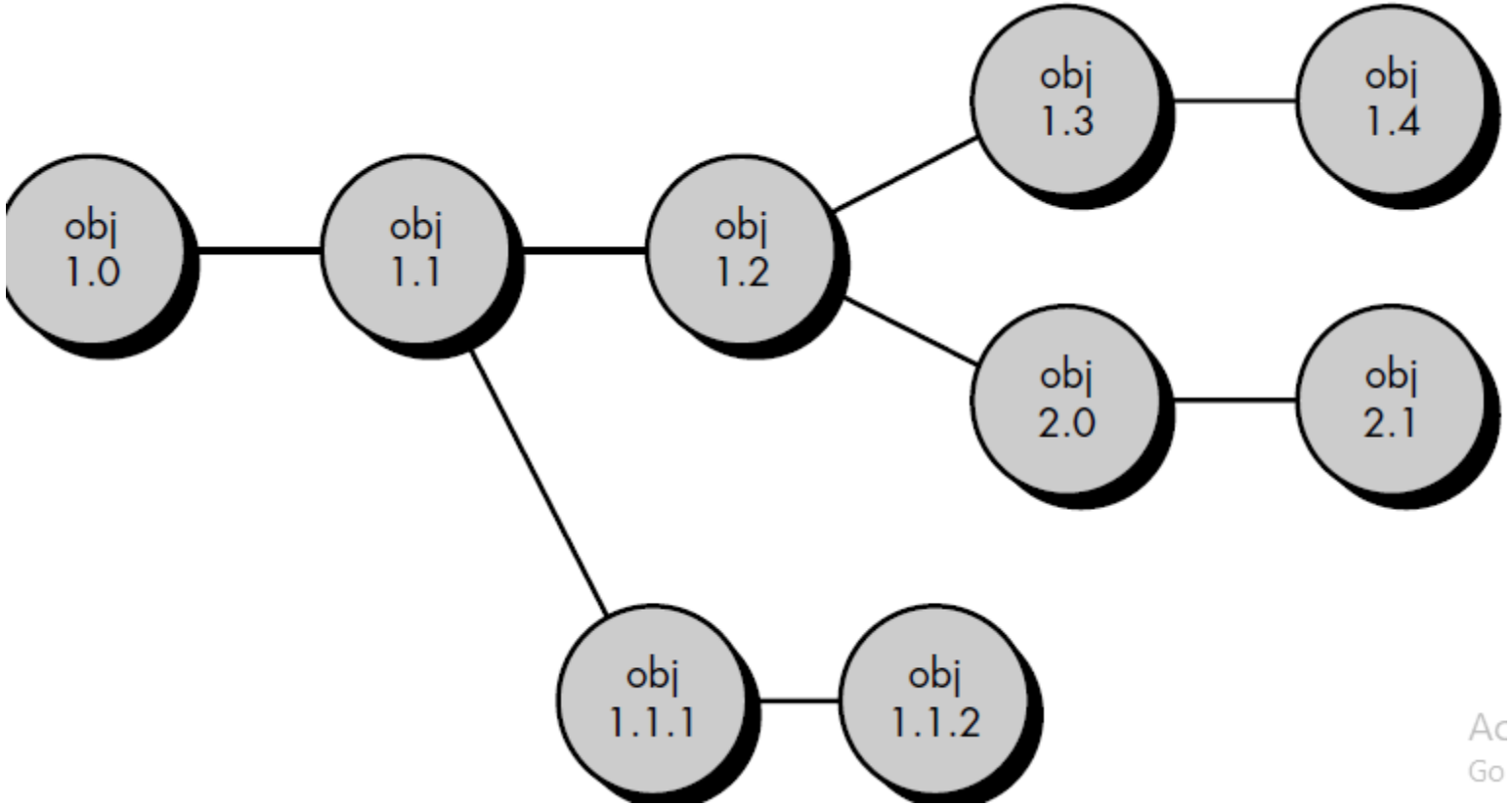
SCM Process: IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

- An aggregate object is a collection of basic objects and other aggregate objects. E.g. **Design Specification** is an aggregate object. Conceptually, it can be viewed as a named (identified) list of pointers that specify basic objects such as **data model** and **component N**.

SCM Process: IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

- The identification scheme for software objects must recognize that objects evolve throughout the software process. Before an object is baselined, it may change many times, and even after a baseline has been established, changes may be quite frequent.
- It is possible to create an evolution graph for any object.

Evolution Graph



Version Control

- *Version control* combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes.
- Each node on the graph is an aggregate object, that is, a complete version of the software.

Version Control

- Each version of the software is a collection of SCIs (source code, documents, data), and each version may be composed of different variants. E.g. one entity is applicable for monochrome display and another entity for color display.
- One or more attributes is assigned for each variant.
- E.g. Git

CHANGE CONTROL

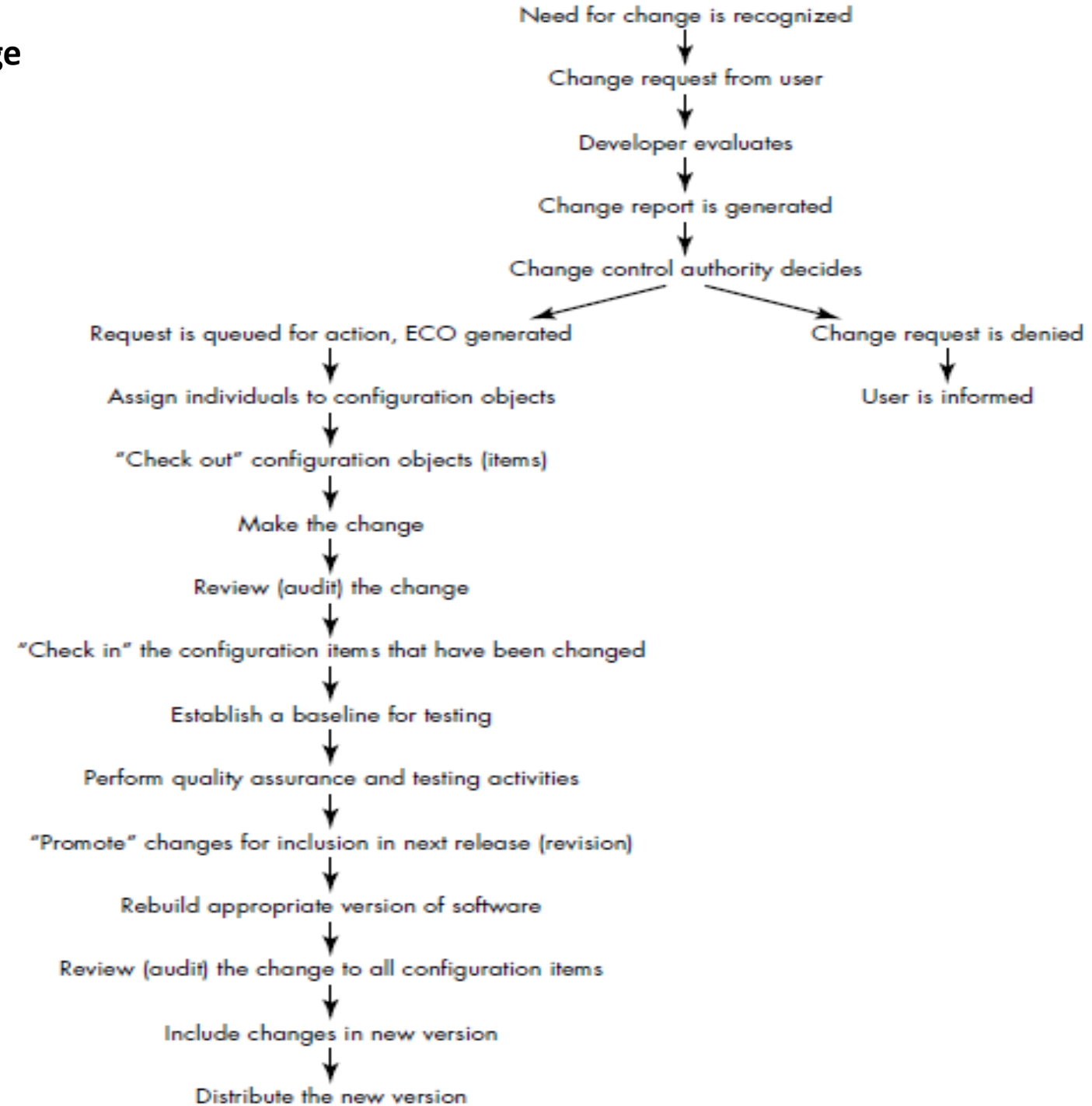
- For a large software engineering project, uncontrolled change rapidly leads to chaos.

CHANGE CONTROL

- For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change.

- A *change request is submitted and evaluated* to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.

Engineering Change Order (ECO)



CHANGE CONTROL

- The results of the evaluation are presented as a *change report, which is used by a change control authority (CCA)—a person or group who makes a final decision on the status and priority* of the change.
- An *engineering change order (ECO) is generated for each approved* change.

CHANGE CONTROL

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

CHANGE CONTROL

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

CHANGE CONTROL

- The "check-in" and "check-out" process implements two important elements of change control—access control and synchronization control.

CHANGE CONTROL

- *Access control governs* which software engineer have the authority to access and modify a particular configuration object.

- *Synchronization control helps to ensure that parallel changes, performed* by two different people, don't overwrite one another

CHANGE CONTROL

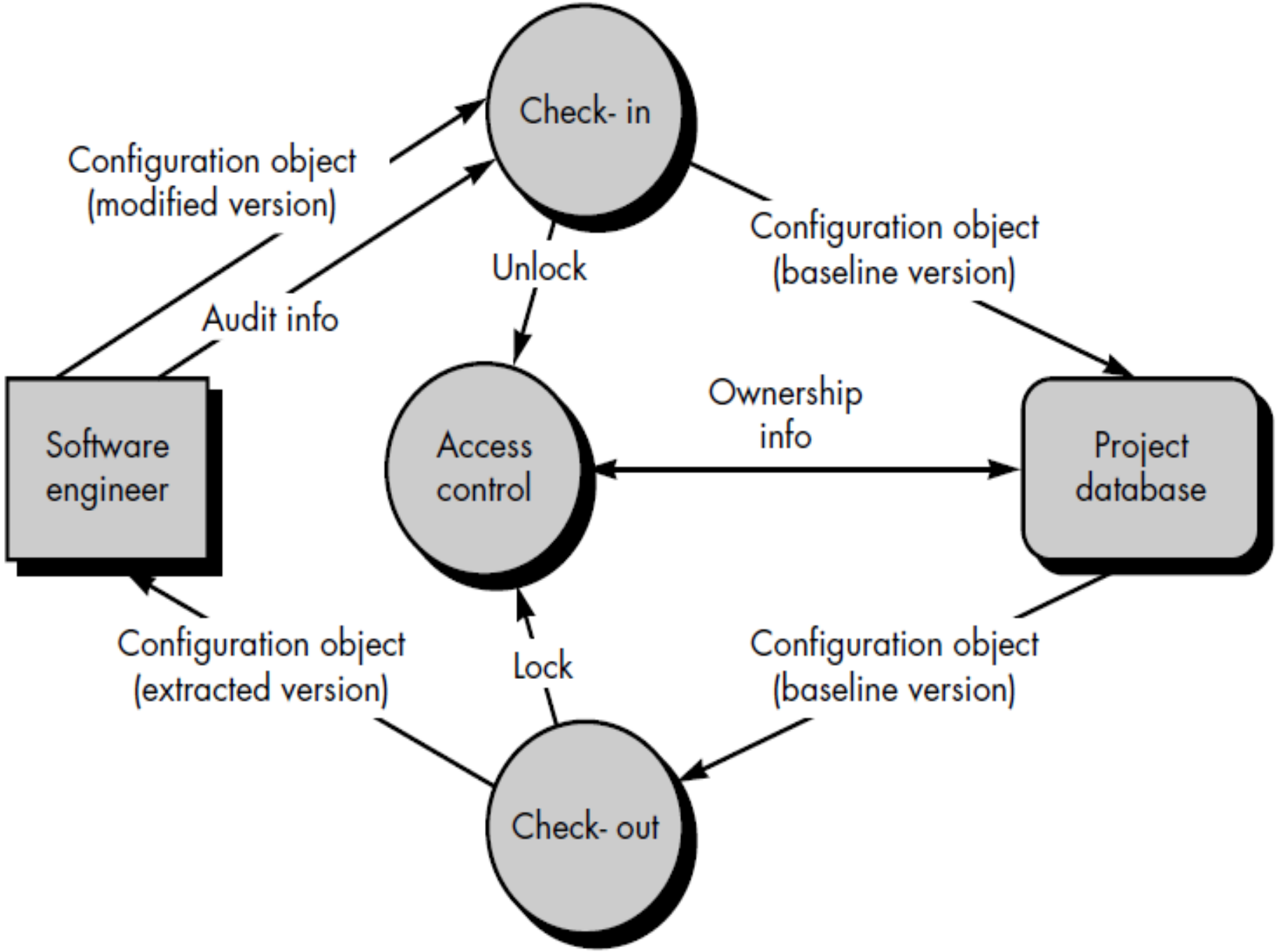
- Access and synchronization control flow are illustrated schematically in Figure (on next slide). Based on an approved change request and ECO, a software engineer *checks out* a configuration object.

CHANGE CONTROL

- An access control function ensures that the software engineer has authority to check out the object, and synchronization control *locks the object in* the project database so that no updates can be made to it until the currently checked out version has been replaced.

CHANGE CONTROL

- Other copies can be checked-out, but other updates cannot be made.
- A copy of the baselined object, called the *extracted version,* is modified by the software engineer.
- After appropriate SQA and testing, the modified version of the object is *checked in and the new baseline object is unlocked.*


Check- in
Configuration object
(modified version)
Audit info
Unlock
Configuration object
(baseline version)
Software
engineer
Access
control
Ownership
info
Project
database
Configuration object
(extracted version)
Lock
Configuration object
(baseline version)
Check- out

CHANGE CONTROL

- Prior to an SCI becoming a baseline, only *informal change control need be applied.*
- The developer of the configuration object (SCI) in question may make whatever changes are justified by project and technical requirements (as long as changes do not affect broader system requirements that lie outside the developer's scope of work).
- Once the object has undergone formal technical review and has been approved, a baseline is created.
- Once an SCI becomes a baseline, *project level change control is* implemented.
- Now, to make a change, the developer must gain approval from the project manager (if the change is "local") or from the CCA if the change affects other SCIs.
- In some cases, formal generation of change requests, change reports, and ECOs is dispensed with.
- However, assessment of each change is conducted and all changes are tracked and reviewed.

CHANGE CONTROL

- The change control authority plays an active role in the second and third layers of control.
- Depending on the size and character of a software project, the CCA may be composed of one person—the project manager—or a number of people (e.g., representatives from software, hardware, database engineering, support, marketing).
- The role of the CCA is to take a global view, that is, to assess the impact of change beyond the SCI in question.
- How will the change affect hardware? How will the change affect performance?
- How will the change modify customer's perception of the product?
- How will the change affect product quality and reliability? These and many other questions are addressed by the CCA.

Configuration Audit

- To track the proper implementation of changes, the audit is done in two folds:
  1. Formal Technical Review
  2. Software Configuration Audit
- The formal technical review focuses on Technical correctness of the configuration object that has been modified. The reviewer assess the SCI and check the consistency with other SCIs.

Configuration Audit

- A *software configuration audit* complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review. The audit is done on following points:

1. It is checked that changes specified in the ECO is done or not and also any additional modifications has been incorporated.

2. The formal technical review is done or not.

Configuration Audit

3. It is checked that the software process has been followed and software engineering standards have been properly applied.

4. It is also necessary to specify the change date and author in the configuration object.

5. It is necessary to follow the SCM procedures for noting the change, recording it, and reporting.

6. The proper updating in all related SCIs have been done or not.

Configuration Status Report

- *Configuration status reporting* (sometimes called *status accounting*) is an SCM task that answers the following questions:

(1)What happened?

(2)Who did it?

(3)When did it happen?

(4)What else will be affected?

- Configuration status reporting is important when multiple developers are modifying SCIs. It also helps to recognize the side effects of the change.

Risk Management

- Risk management is an attempt to minimize the chances of failure caused by unplanned events.
- The aim of risk management is not to avoid getting into projects that have risks but to minimize the impact of risks in the projects that are undertaken.

Risk Management

- A risk is a probabilistic event—it may or may not occur.
- For this reason, we frequently have an optimistic tendency to simply not see risks or to wish that they will not occur.

Risk Management Concepts

- *Risk is defined as an exposure to the chance of injury or loss.*
- *That is, risk* implies that there is a possibility that something negative may happen.
- In the context of software projects, negative implies that there is an adverse effect on cost, quality, or schedule.

Risk Management Concepts

- *Risk management is the area that tries* to ensure that the impact of risks on cost, quality, and schedule is minimal.

Risk Management Concepts

- Risk management can be considered as dealing with the possibility and actual occurrence of those events that are not "regular" or commonly expected, that is, they are probabilistic.
- The commonly expected events, such as people going on leave or some requirements changing, are handled by normal project management.

Risk Management

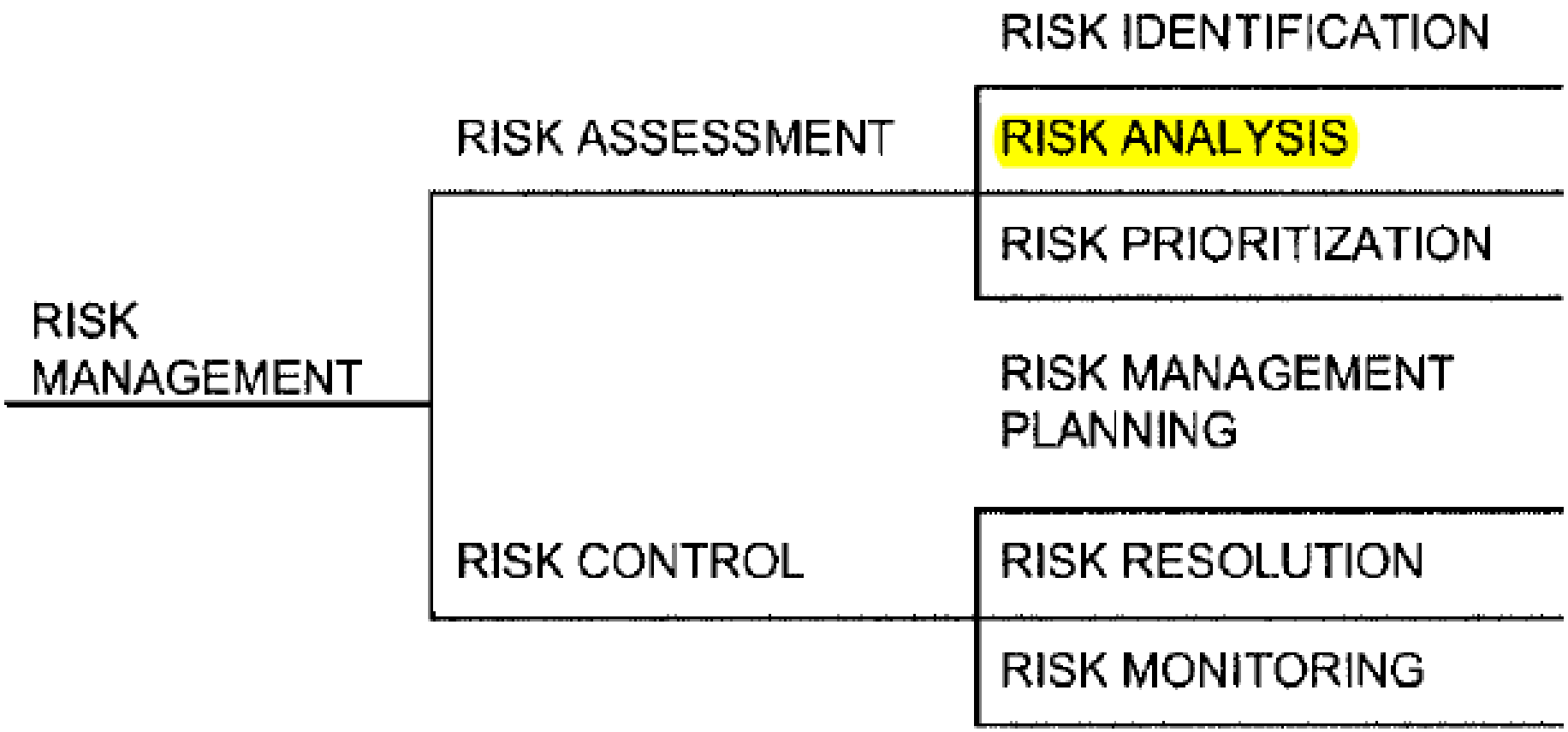
- The idea of risk management is to minimize the possibility of risks materializing, if possible, or to minimize the effects if risks actually materialize.
- It should be clear that risk management has to deal with identifying the undesirable events that can occur, the probability of their occurring, and the loss if an undesirable event does occur.

Risk Management

- Once this is known, strategies can be formulated for either reducing the probability of the risk materializing or reducing the effect of risk materializing.

- So the risk management revolves around *risk assessment and risk control.*

Risk Management Activities



Risk Assessment

- Risk assessment is an activity that must be undertaken during project planning.
- This involves identifying the risks, analysing them, and prioritizing them on the basis of the analysis.
- Due to the nature of a software project, uncertainties are highest near the beginning of the project

Risk Assessment

- **The goal of risk assessment is to prioritize the risks so that attention and resources can be focused on the more risky items.**
- *Risk identification* is the first step in risk assessment, which identifies all the different risks for a particular project.

Risk Assessment

- These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong.
- Methods that can aid risk identification include **checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products**

Risk Assessment

- Checklists of frequently occurring risks are probably the most common tool for risk identification—most organizations prepare a list of commonly occurring risks for projects, prepared from a survey of previous projects.
- Such a list can form the starting point for identifying risks for the current project.

Risk Assessment

- **Boehm has produced a list of the top 10 risk items** likely to compromise the success of a software project.
- Though risks in a project are specific to the project, this list forms a good starting point for identifying such risks.

	RISK ITEM	RISK MANAGEMENT TECHNIQUES
1	Personnel Shortfalls	Staffing with top talent; Job matching; Team building; Key personnel agreements; training; Prescheduling key people
2	Unrealistic Schedules and Budgets	Detailed multi source cost and schedule estimation; Design to cost; Incremental Development; Software reuse; Requirements scrubbing
3	Developing the Wrong Software Functions	Organization analysis; Machine analysis; Ops concept forumlation; User surveys; Prototyping; Early user's manuals
4	Developing the Wrong User Interface	Prototyping; Scenarios; Task analysis; User characterization
5	Gold Plating	Requirements scrubbing; Prototyping; Cost benefit analysis; Design to cost
6	Continuing Stream of Requirement Changes	High change threshold; Information hiding; Incremental development
7	Shortfalls in Externally Furnished Components	Benchmarking inspections; Reference checking; Compatibility analysis
8	Shortfalls in Externally Performed Tasks	Reference checking; Preaward audits; Award free contracts; Competetive design or prototyping; Team building
9	Real Time Performance Shortfalls	Simulation; Benchmarking; Modeling; Prototyping; Instrumentation; Tuning
10	Straining Computer Science Capabilities	Technical analysis; Cost benefit analysis; Prototyping; Reference checking

Risk Assessment

- The top-ranked risk item is personnel shortfalls. This involves just having fewer people than necessary or not having people with specific skills that a project might require.
- Some of the ways to manage this risk is to get the top talent possible and to match the needs of the project with the skills of the available personnel.
- Adequate training, along with having some key personnel for critical areas of the project, will also reduce this risk.

Risk Assessment

- The second item, unrealistic schedules and budgets, happens very frequently due to business and other reasons.
- It is very common that high-level management imposes a schedule for a software project that is not based on the characteristics of the project and is unrealistic.
- Underestimation may also happen due to inexperience or optimism.

Risk Assessment

- The next few items are related to requirements. Projects run the risk of developing the wrong software if the requirements analysis is not done properly and if development begins too early.
- Similarly, often improper user interface may be developed.
- This requires extensive rework of the user interface later or the software benefits are not obtained because users are reluctant to use it.

Risk Assessment

- Some requirement changes are to be expected in any project, but sometimes frequent changes are requested, which is often a reflection of the fact that the client has not yet understood or settled on its own requirements.

Risk Assessment

- The effect of requirement changes is substantial in terms of cost, especially if the changes occur when the project has progressed to later phases.
- Performance shortfalls are critical in real-time systems and poor performance can mean the failure of the project.

Risk Assessment

- If a project depends on externally available components—either to be provided by the client or to be procured as an off-the-shelf component— the project runs some risks.
- The project might be delayed if the external component is not available on time.

Risk Assessment

- The project would also suffer if the quality of the external component is poor or if the component turns out to be incompatible with the other project components or with the environment in which the software is developed or is to operate.
- If a project relies on technology that is not well developed, it may fail.
- This is a risk due to straining the computer science capabilities.

Risk Identification

- Risk identification merely identifies the undesirable events that might take place during the project, i.e., enumerates the "unforeseen" events that might occur.
- It does not specify the probabilities of these risks materializing nor the impact on the project if the risks indeed materialize.

Risk Analysis

- In risk analysis, **the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize.**
- This is often done through discussion, using experience and understanding of the situation.

Risk Analysis

- However, if cost models are used for cost and schedule estimation, then the same models can be used to assess the cost and schedule risk.

Risk Analysis

- One possible source of cost risk is underestimating these cost drivers. The other is underestimating the size.
- Risk analysis can be done by estimating the worst-case value of size and all the cost drivers and then estimating the project cost from these values.

Risk Analysis

- This will **give us the worst-case analysis**. Using the worst-case effort estimate, the worst-case schedule can easily be obtained.
- A more detailed analysis can be done by considering different cases or a distribution of these drivers.

Risk Analysis

- The other approaches for risk analysis **include studying the probability and the outcome of possible decisions (decision analysis), understanding the task dependencies to decide critical activities and the probability and cost of their not being completed on time (network analysis), risks on the various quality factors like reliability and usability (quality factor analysis), and evaluating the performance early through simulation**, etc., if there are strong performance constraints on the system (performance analysis).

Risk Analysis

Risk Analysis

- Once the probabilities of risks materializing and losses due to materialization of different risks have been analyzed, they can be prioritized.
- One approach for prioritization is through the concept of *risk exposure (RE),* which is sometimes called *risk impact. RE is defined by the relationship*

$$RE = Prob(UO) * Loss(UO)$$

•where *Prob{UO) is the probability of the risk materializing (i.e., undesirable* outcome) and *Loss{UO) is the total loss incurred due to the unsatisfactory* outcome.
•The loss is not only the direct financial loss that might be incurred but also any loss in terms of credibility, future business, and loss of property or life.
•The RE is the expected value of the loss due to a particular risk.
•For risk prioritization using RE, the higher the RE, the higher the priority of the risk item.

Risk Analysis

- A subjective assessment can be done by the estimate of one person or by using a group consensus technique like the Delphi approach.
- In the Delphi method, a group of people discusses the problem of estimation and finally converges on a consensus estimate.

Risk Control

- The main objective of risk management is to identify the top few risk items and then focus on them.
- Once a project manager has identified and prioritized the risks, the top risks can be easily identified.
- Knowing the risks is of value only if you can prepare a plan so that their consequences are minimal—that is the basic goal of risk management.
- One obvious strategy is risk avoidance, which entails taking actions that will avoid the risk altogether

Risk Control

- For most risks, the strategy is to perform the actions that will either reduce the probability of the risk materializing or reduce the loss due to the risk materializing.
- These are called **risk mitigation steps.**
- To decide what mitigation steps to take, a list of commonly used risk mitigation steps for various risks is very useful here.

Risk Control

- Selecting a risk mitigation step is not just an intellectual exercise.
- The risk mitigation step must be executed (and monitored).
- To ensure that the needed actions are executed properly, they must be incorporated into the detailed project schedule.

Risk Control

- **Risk prioritization and consequent planning are based on the risk perception at the time the risk analysis is performed**.
- Because risks are probabilistic events that frequently depend on external factors, the threat due to risks may change with time as factors change.
- Clearly, then, the risk perception may also change with time. Furthermore, the risk mitigation steps undertaken may affect the risk perception.

Risk Control

- This dynamism implies that risks in a project should not be treated as static and must be monitored and re-evaluated periodically.
- Hence, in addition to monitoring the progress of the planned risk mitigation steps, a project must periodically revisit the risk perception and modify the risk mitigation plans, if needed.

Risk Control

- *Risk monitoring is the activity of monitoring* the status of various risks and their control activities.

- One simple approach for risk monitoring is to analyze the risks afresh at each major milestone, and change the plans as needed.

What is CASE ??

CASE – Stands for Computer Aided Software Engineering.

Definition –

What is CASE ??

It is the scientific application of a set of tools and methods to a software system which is meant to result in high-quality, defect-free, and maintainable software products

It also refers to methods for the development of information systems together with automated tools that can be used in the software development process.

What is CASE ??

Two key ideas of Computer-aided Software System Engineering (CASE) are

- To increase productivity
- To help produce better quality software at lower cost

CASE & It's Scope

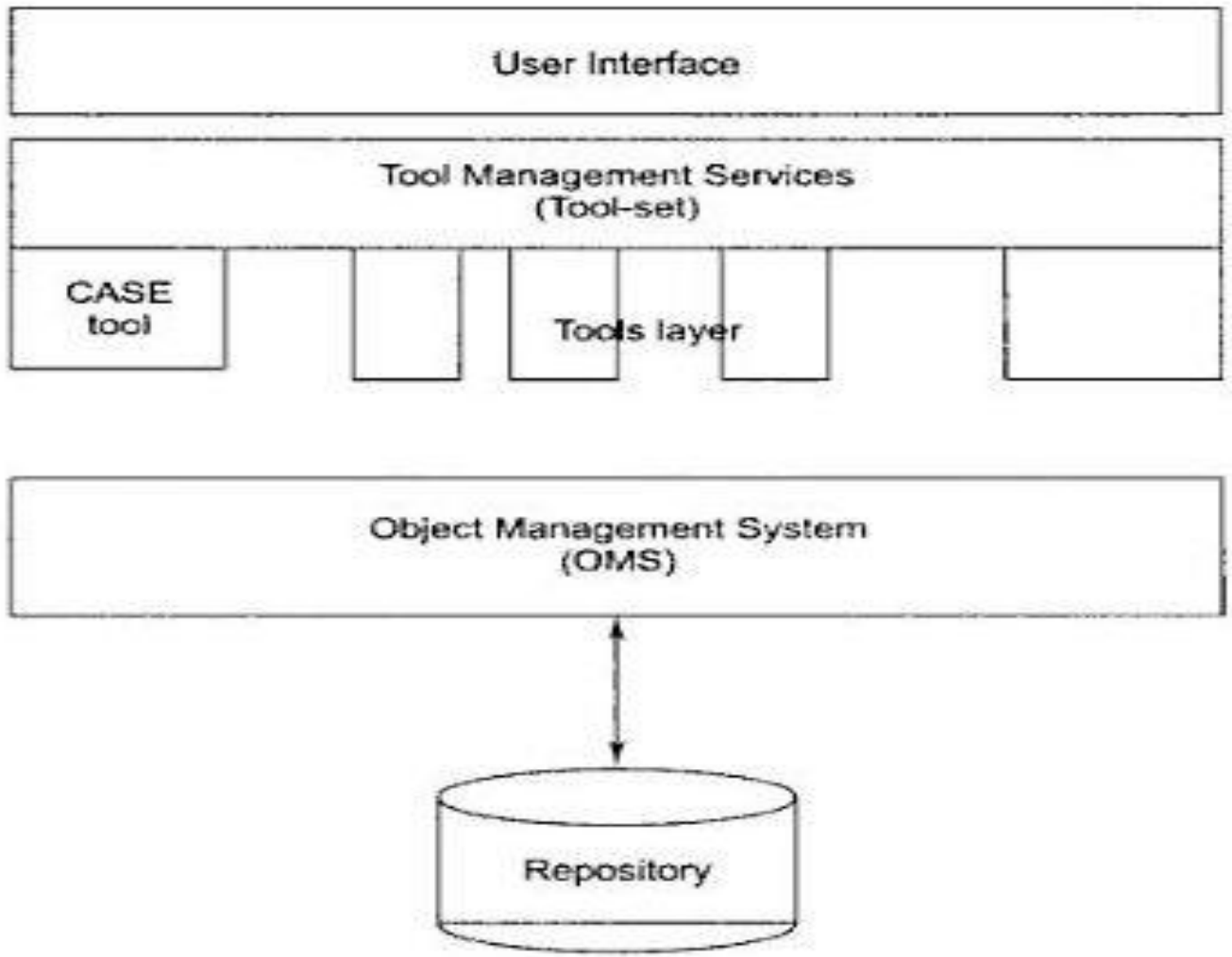
CASE technology provides software-process support by automating some process activities and by providing information about the software being developed. Examples of activities which can be automated by using CASE tools are

- The development of graphical models as part of the requirement specification or the software design
- Understanding a design using a data dictionary, which holds information about the entities and relations in a design.
- The generation of user interfaces
- Program debugging
- The automated translation of programs from an older version of a programming language to a recent version.

Levels Of CASE

- There are three different levels of CASE Technology
- **Production support Technology** – This includes support for process activities such as specification, design, implementation , testing and so on.
- **Process Management Technology** – This includes tools to support process modeling and process management.
- **Meta Case Tools** – are generators, which are used to create production process-management tools

Architecture of CASE Environment



Important Components of a modern CASE environment

1. **User Interface** – it provides a consistent framework for accessing different tools, thus making it easier for the user to interact with different tools and reduces learning time of how the different tools are used.

Important Components of a modern CASE environment

Important Components of a modern CASE environment

Important Components of a modern CASE environment

2. **Tools Management System (Tool set)** – The tools set section holds different types of improved quality tools. The tools layer incorporates a set of tools-management services with the CASE tool themselves. The Tools Management Services (TMS) controls the behavior of tools within the environment. If multitasking is used during the execution of the one or more tools, TMS performs multitask synchronization and communication, co-ordinates the flow of information from the repository and object-management system into the tools, accomplishes security and auditing functions and collects metrics on tool use.

Important Components of a modern CASE environment

3. **Object Management System (OMS)** – maps these (specification design, text data project plan, etc.) logical entities into the underlying storage-management system i.e. the repository.

Important Components of a modern CASE environment

4. **Repository** – is the CASE database and the access-control functions that enable the OMS to interact with the database. The CASE repository generally referred as Project Database, Data Dictionary, CASE Database.

Characteristics of CASE Tools

1. A graphical interface to draw diagrams, chart models, (upper case, middle case, lower case)
2. An information repository, a data dictionary for efficient information management selection, usage, application and storage.
3. Common user interface for integration of multiple tools used in various phases.
4. Automatic code generators
5. Automating testing tools

Types of CASE Tools

- Upper CASE Tools
- Lower CASE Tools
- Integrated Case Tools (I –CASE) Tools

CASE Tool Types

Upper CASE Tools –

Designed to support the Analysis and Design phase of SDLC.

These are also known as front-end tools.

They support traditional diagrammatic languages such as ER diagrams, Data flow diagram, Structure charts, Decision Trees, Decision tables, etc.

The example of CASE tool in this group are ERwin and Visual UML.

CASE Tool Types

Lower Case Tools –

- Designed to support the implementation, testing and maintenance phase of SDLC
- Example – Code Generators.
- These are called back-end tools.
- The example of CASE tool in the group are Ecore Diagram Editor and dzine.

CASE Tool Types

Integrated Tools –

- These tools support the entire life cycle, such as project management and estimation of costs
- These are also known as I- CASE Tools.
- The example of CASE tool in this group is Rational Rose from IBM.

Advantages of CASE Tools

1. Improved Productivity
2. Better Documentation
3. Improved Accuracy
4. Intangible benefits
5. Improved Quality
6. Reduced Lifetime Maintenance
7. Reduced cost of Software
8. Produce high quality and consistent document
9. Easy to program software
10. Easy project management
11. An increase in project control through better planning, monitoring and communication.
12. Help standardization of notations and diagrams
13. Reduction of time and effort

Disadvantages of CASE

- Purchasing is not an easy task – Cost is very high.
- Learning curve – Initial productivity may fall. Users may require extra training to use CASE tools.
- Tool Mix – Proper selection of CASE tools should there to get maximum benefit.
- May lead to restriction to the tool’s capabilities

Resource Allocation Model

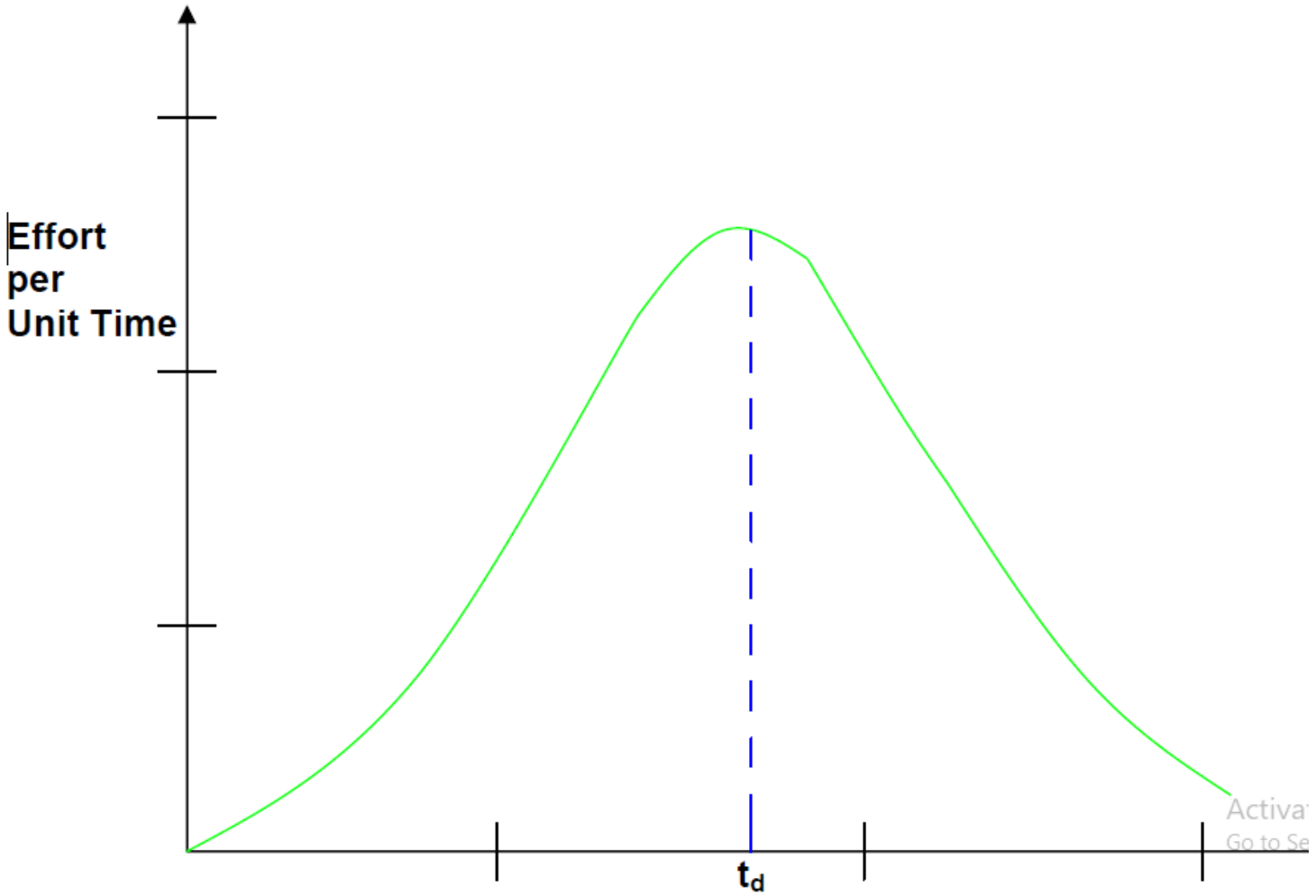
- Putnam first studied the problem of what should be a proper staffing pattern for software projects. He extended the work of Norden who had earlier investigated the staffing pattern of research and development (R&D) type of projects.
- In order to appreciate the staffing pattern of software projects, Norden’s and Putnam’s results must be understood.

Resource Allocation Model: Norden's Work

- Staffing pattern can be approximated by the Rayleigh distribution curve. Norden represented the Rayleigh curve by the following equation:

$$E = K/t_d^2 * t * e^{-t^2 / 2 t_d^2}$$

- Where E is the effort required at time t.
- E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project,
- K is the area under the curve, and td is the time at which the curve attains its maximum value. It must be remembered that the results of Norden are applicable to general R & D projects and were not meant to model the staffing pattern of software development projects.



Resource Allocation Model: Nordens Work

Resource Allocation Model : Putnam's Work

- Putnam studied the problem of staffing of software projects and found that the software development has characteristics very similar to other R & D projects studied by Norden and that the Rayleigh-Norden curve can be used to relate the number of delivered lines of code to the effort and the time required to develop the project.
- By analyzing a large number of army projects, Putnam derived the following expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

Resource Allocation Model : Putnam's Work

- The various terms of this expression are as follows:
- K is the total effort expended (in PM) in the product development and L is the product size in KLOC.
- td corresponds to the time of system and integration testing. Therefore, td can be approximately considered as the time required to develop the software.

$$L = C_k K^{1/3} t_d^{4/3}$$

Resource Allocation Model : Putnam's Work

- Ck is the state of technology constant and reflects constraints that impede the progress of the programmer.
- Typical values of C_k = 2 for poor development environment (no methodology, poor documentation, and review, etc.), Ck = 8 for good software development environment (software engineering principles are adhered to), Ck = 11 for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used).
- The exact value of Ck for a specific project can be computed from the historical data of the organization developing it.

Resource Allocation Model: Putnam's Work

- Putnam suggested that optimal staff build-up on a project should follow the Rayleigh curve.

Resource Allocation Model: Putnam's Work

- Only a small number of engineers are needed at the beginning of a project to carry out planning and specification tasks.
- As the project progresses and more detailed work is required, the number of engineers reaches a peak.

Resource Allocation Model: Putnam's Work

- After implementation and unit testing, the number of project staff falls.
- However, the staff build-up should not be carried out in large installments.

Resource Allocation Model: Putnam's Work

- The team size should either be increased or decreased slowly whenever required to match the Rayleigh-Norden curve.
- Experience shows that a very rapid build up of project staff any time during the project development correlates with schedule slippage.

Resource Allocation Model : Putnam's Work

- It should be clear that a constant level of manpower through out the project duration would lead to wastage of effort and increase the time and effort required to develop the product.
- If a constant number of engineers are used over all the phases of a project, some phases would be overstaffed and the other phases would be understaffed causing inefficient use of manpower, leading to schedule slippage and increase in cost.

Resource Allocation Model : Putnam's Work

$$L = CkK^{1/3}td^{4/3}$$

- Effect of schedule change on cost:
  - By analyzing a large number of army projects, Putnam derived the following expression:
  - Where, K is the total effort expended (in PM) in the product development and L is the product size in KLOC, td corresponds to the time of system and integration testing and Ck is the state of technology constant and reflects constraints that impede the progress of the programmer Now by using the above expression it is obtained that,

Resource Allocation Model : Putnam's Work

- Effect of schedule change on cost:

$$K = L^3/C_k^3 t_d^4$$

Or,

$$K = C/t_d^4$$

For the same product size, $C = L^3 / C_k^3$ is a constant.

or, $K_1/K_2 = t_{d2}^4/t_{d1}^4$

or, $K \propto 1/t_d^4$

or, $cost \propto 1/t_d$

Resource Allocation Model : Putnam's Work

- From the above expression, it can be easily observed that when the schedule of a project is compressed, the required development effort as well as project development cost increases in proportion to the fourth power of the degree of compression.

Resource Allocation Model : Putnam's Work

- It means that a relatively small compression in delivery schedule can result in substantial penalty of human effort as well as development cost.
- For example, if the estimated development time is 1 year, then in order to develop the product in 6 months, the total effort required to develop the product (and hence the project cost) increases 16 times.

COCOMO Model

- COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e. **number of Lines of Code**.

COCOMO Model

- It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.
- It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

COCOMO Model

- The key parameters which define the quality of any software products, which are also an outcome of the COCOMO are primarily Effort & Schedule.

COCOMO Model

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

COCOMO Model

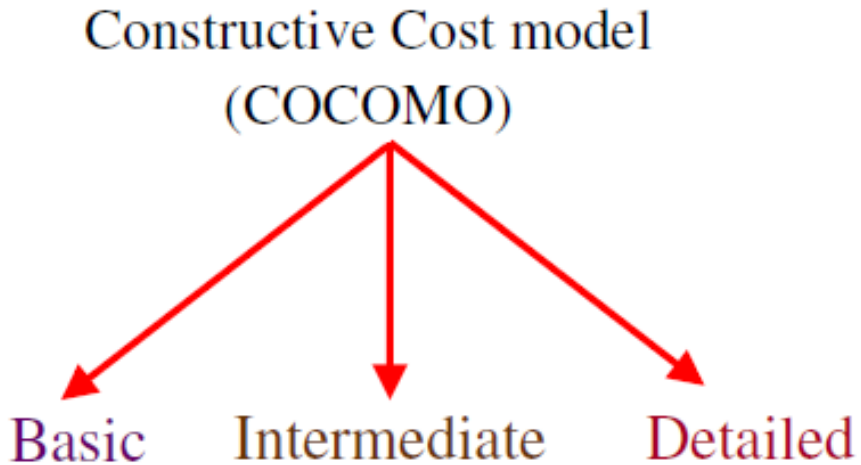
- Different models of COCOMO have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required.

COCOMO Model

- All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

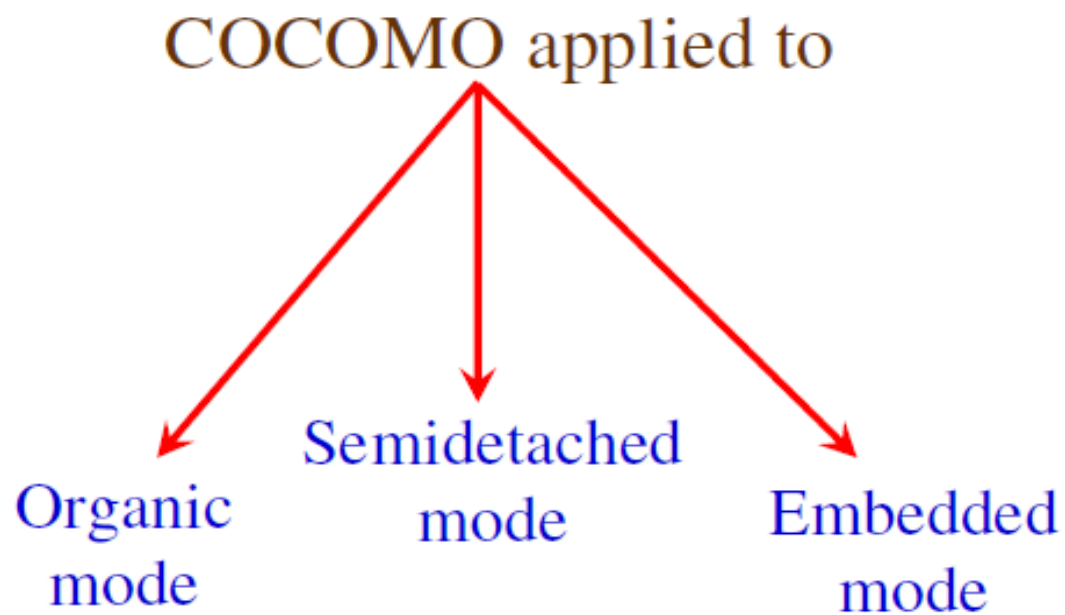
Software Project Planning

The Constructive Cost Model (COCOMO)



Model proposed by
B. W. Boehm's
through his book
Software Engineering Economics in 1981

Software Project Planning



Software Project Planning

Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

Table 4: The comparison of three COCOMO modes

- **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs.
- Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Basic Model

Basic COCOMO model takes the form

$$E = a_b(KLOC)^{b_b}$$

$$D = c_b(E)^{d_b}$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in table 4 (a).

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 4(a): Basic COCOMO coefficients

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size } (SS) = \frac{E}{D} \; Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} \; KLOC / PM$$

Example: 4.5

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

Solution

Estimated size of the project = 400 KLOC

(i) Organic mode

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \text{ PM}$$

(ii) Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{0.35} = 38.45 \text{ PM}$$

(iii) Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{0.32} = 38 \text{ PM}$$

Example: 4.6

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{0.35} = 29.3$ PM

$$\text{Average staff size } (SS) = \frac{E}{D} Persons$$

$$= \frac{1133.12}{29.3} = 38.67 Persons$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765\ KLOC / PM$$

$$P = 176\ LOC / PM$$

Intermediate Model –However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as **reliability, experience, Capability**. These factors are known as **Cost Drivers** and the **Intermediate Model utilizes 15 such drivers for cost estimation**.

Classification of Cost Drivers and their attributes:

Intermediate Model

Cost drivers

(*i*) Product Attributes

- Required s/w reliability
- Size of application database
- Complexity of the product

(*ii*) Hardware Attributes

- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

(*iii*) Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(*iv*) Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

Multipliers of different cost drivers

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	--
DATA	--	0.94	1.00	1.08	1.16	--
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	--	--	1.00	1.11	1.30	1.66
STOR	--	--	1.00	1.06	1.21	1.56
VIRT	--	0.87	1.00	1.15	1.30	--
TURN	--	0.87	1.00	1.07	1.15	--

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	--
AEXP	1.29	1.13	1.00	0.91	0.82	--
PCAP	1.42	1.17	1.00	0.86	0.70	--
VEXP	1.21	1.10	1.00	0.90	--	--
LEXP	1.14	1.07	1.00	0.95	--	--
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	--
TOOL	1.24	1.10	1.00	0.91	0.83	--
SCED	1.23	1.08	1.00	1.04	1.10	--

Table 5: Multiplier values for effort calculations

Intermediate COCOMO equations

$$E = a_i (KLOC)^{b_i} * EAF$$

$$D = c_i (E)^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Table 6: Coefficients for intermediate COCOMO

The product of all effort multipliers results in an *effort adjustment factor (EAF)*. Typical values for EAF range from 0.9 to 1.4.

Example: 4.7

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used

Or

Developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

Detailed COCOMO Model

- Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.
- The five phases of detailed COCOMO are:
    1. Planning and requirements
    2. System structure
    3. Complete structure
    4. Module code and test
    5. Integration and test

Detailed COCOMO = Intermediate COCOMO + assessment of Cost Drivers impact on each phase.

Detailed COCOMO Model

- Multiply all 15 Cost Drivers to get **Effort Adjustment Factor(EAF)**
- **E(Effort) = $a_b(KLOC)^{b_b}$ * EAF**(in Person-Month)
- **D(Development Time) = $c_b(E)^{d_b}$** (in month)
- **E_p (Total Effort) = μ_p * E** (in Person-Month)
- **D_p (Total Development Time) = τ_p * D** (in month)

Lifecycle Phase Values of μ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.06	0.16	0.26	0.42	0.16
Organic medium S≈32	0.06	0.16	0.24	0.38	0.22
Semidetached medium S≈32	0.07	0.17	0.25	0.33	0.25
Semidetached large S≈128	0.07	0.17	0.24	0.31	0.28
Embedded large S≈128	0.08	0.18	0.25	0.26	0.31
Embedded extra large S≈320	0.08	0.18	0.24	0.24	0.34

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

Lifecycle Phase Values of τ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.10	0.19	0.24	0.39	0.18
Organic medium S≈32	0.12	0.19	0.21	0.34	0.26
Semidetached medium S≈32	0.20	0.26	0.21	0.27	0.26
Semidetached large S≈128	0.22	0.27	0.19	0.25	0.29
Embedded large S≈128	0.36	0.36	0.18	0.18	0.28
Embedded extra large S≈320	0.40	0.38	0.16	0.16	0.30

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

Detailed COCOMO : Example

Consider a project to develop a full screen editor. The major components identified and their sizes are (i) Screen Edit – 4K (ii) Command Lang Interpreter – 2K (iii) File Input and Output – 1K (iv) Cursor movement – 2K (v) Screen Movement – 3K. Assume the Required software reliability is high, product complexity is high, analyst capability is high & programming language experience is low. Use COCOMO model to estimate cost and time for different phases.

Size of modules : 4 + 2 + 1 + 2 + 3 = 13 KLOC **[Organic]**

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	**1.15**	1.40	--
CPLX	0.70	0.85	1.00	**1.15**	1.30	1.65
ACAP	1.46	1.19	1.00	**0.86**	0.71	
LEXP	1.14	**1.07**	1.00	0.95	--	--

EA

Example (Contd.)

Initial Effort (E) = $a_b(KLOC)^{b_b}$ * EAF = $3.2*(12)^{1.05}$ * 1.2169 = 52.9 person-months

Initial Development Time = $c_b(E)^{d_b}$ =$2.5*(52.9)^{0.38}$ = 11.29 months

Phase value of μ_p and τ_p

Phase wise effort & development time distribution

	Plan & Req^r	System Design	Detail Design	Module code & test	Integration & Test
Organic Small μ_p	0.06	0.16	0.26	0.42	0.16
Organic Small τ_p	0.10	0.19	0.24	0.39	0.18

	E	D	Ep (in person-months)	Dp (in months)
Plan & Requirement	52.9	11.29	0.06*52.9 = 3.17	0.10*11.29=1.12
System Design	52.9	11.29	0.16*52.9=8.46	0.19*11.29=2.14
Detail Design	52.9	11.29	0.26*52.9=13.74	0.24*11.29=2.70
Module code & test	52.9	11.29	0.42*52.9=22.21	0.39*11.29=4.40
Integration & test	52.9	11.29	0.16*52.9=8.46	0.18*11.29=2.03

COCOMO-II

The following categories of applications / projects are identified by COCOMO-II and are shown in fig. 4 shown below:

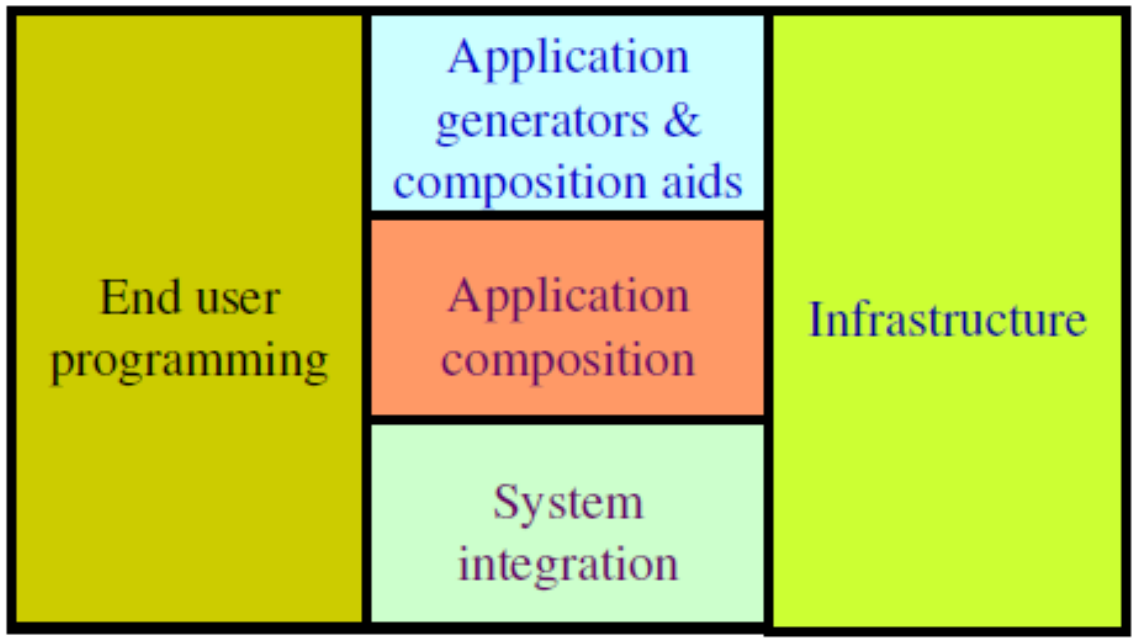


Fig. 4 : Categories of applications / projects

1. End user programming: This category is applicable to small systems, developed by end user using application generators. End user may write small programs using application generators.
2. Infrastructure sector: Software that provide infrastructure like OS , DBMS, networking systems etc. These developers generally have good knowledge of software development.
3. Intermediate sectors: Divided into three sub categories.
    1. Application generators and composition aids: Create largely pre packaged capabilities for user programming.
    2. Application composition sector: GUI’s , databases , domain specific components such as financial , medical etc.
    3. System integration: Large scale highly embedded systems

Stage No	*Model Name*	*Application for the types of projects*	*Applications*
Stage I	Application composition estimation model	Application composition	In addition to application composition type of projects, this model is also used for prototyping (if any) stage of application generators, infrastructure & system integration.
Stage II	Early design estimation model	Application generators, infrastructure & system integration	Used in early design stage of a project, when less is known about the project.
Stage III	Post architecture estimation model	Application generators, infrastructure & system integration	Used after the completion of the detailed architecture of the project.

Table 8: Stages of COCOMO-II