UNIT - 4

Chapter: 1 - Software Configuration Management, SCM Process, Product M
Requirements Model, Design Model, Source Code, Testing and Maintenance.

Chapter: 2 - Managing Software Projects: The Project Management Spectru
Principle, Metrics in the Process and Project Domains, Software Measurement, M
Software Quality, Integrating Metrics within the Software Process, Softwa
Estimation, Decomposition Techniques.

Chapter:3 - Project Scheduling – basics, scheduling, Software Risks, Risk M
Monitoring, and Management, Software Maintenance, Software Reengineering
Engineering, Forward Engineering.

Software Configuration Manageme
Chapter -1

SOFTWARE CONFIGURATION MANAGEMENT

- AN SCM SCENARIO
- ELEMENTS OF CONFIGURATION MANAGEMENT SYSTEM
- BASELINES
- SOFTWARE CONFIGURATION ITEMS

SOFTWARE CONFIGURATION MANAGEMENT(SCM)

- The art of coordinating software development to minimize . . . confusion is called configuration management.
- Umbrella activity that is applied throughout the software process.
- SCM activities are developed to
  - (1) identify change
  - (2) control change
  - (3) ensure that change is being properly implemented, and
  - (4) report changes to others who may have an interest.
- It is important to make a clear distinction between software support and software configuration management.

SOFTWARE CONFIGURATION MANAGEMENT

The output of the software process is information that may be divided into:

1. **Computer programs** (both source level and executable forms).
2. **Work products** that describe the computer programs (targeted at various stakeholders.
3. **Data or Content** (contained within the program or external to it).

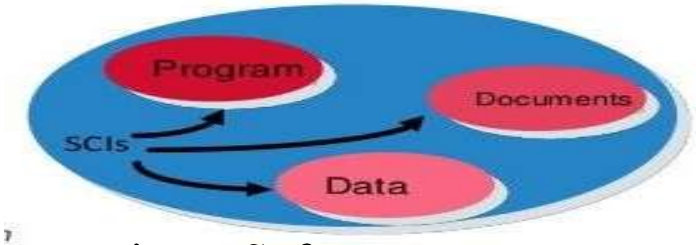


Figure:SoftwareProcess

❖ The items that comprise all information produced as part of the software process are called a software configuration.

SOFTWARE CONFIGURATION MANAGEMENT

- The First Law of System Engineering [Ber80] states:

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

- What is the origin of these changes? The answer to this question is as varied as the changes themselves. However, there are four fundamental sources of change:

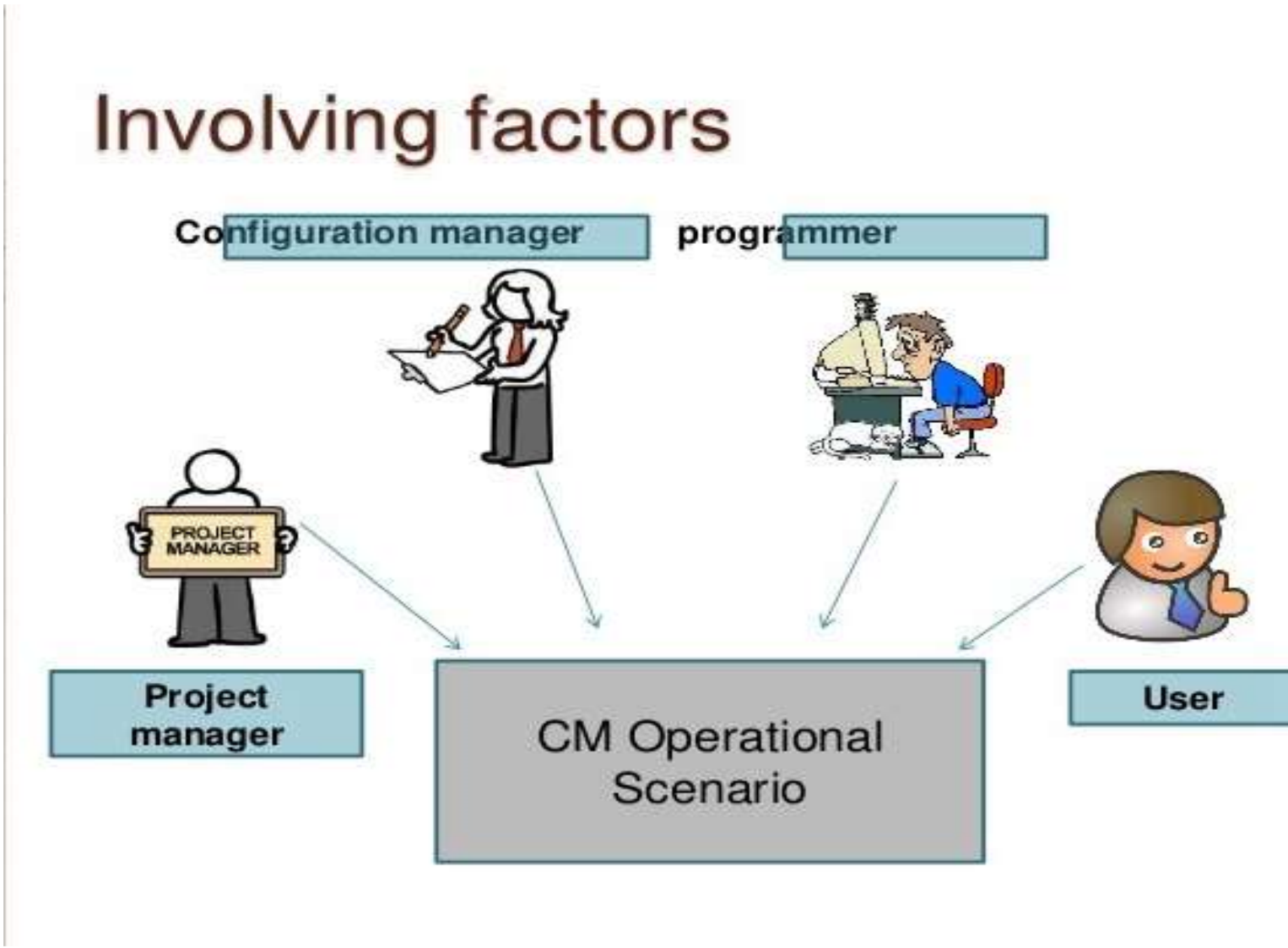
  (1)New business or market conditions dictate changes in product requirements or business rules.

  (2)New stakeholder needs demand modification of data produced by information systems, functionality delivered by products or services delivered by computer based system.

  (3)Reorganization or business growth/downsizing causes changes in project priorities or software engineering team structure.

  (4)Budgetary or scheduling constraints cause a redefinition of the system or product.

AN SCM SCENARIO



AN SCM SCENARIO

He is in charge of a computer program.

He is in charge of CM procedures and policies.

Software engineers who are responsible for developing and maintaining the software product.

He uses the product.

AN SCM SCENARIO

❖ **Role and tasks of project manager:**

(1)Ensure that the product is framed within certain time frame.

(2)Monitors the progress of development and recognizes and reacts to problems.

(3)Done by generating and analyzing reports about the status of the software system and by performing reviews on the system.

AN SCM SCENARIO

❖ **Role and tasks of Configuration manager:**

I. Ensures that procedures and policies for creating, changing and testing of code are followed.

II. Introduces mechanisms for making official requests for changes.

III. Creates and disseminates task lists for engineers and basically creates the project context.

IV. Collects statistics about the components in the software system, such as information determining which components in the system are problematic .

AN SCM SCENARIO

- **Role and tasks of Software engineer:**

1)The goal is to work effectively.

2)They communicate and coordinate by notifying one another about tasks required and tasks completed.

3)Engineers use tools that helps build a consistent software product.

4)The engineers have their own workspace for creating , changing , testing , and integrating code.

5) At certain time the code is made into a baseline from which further development.

AN SCM SCENARIO

❖ **Role of User:**

The customer uses the product . Since the product is under CM control, the customer follows formal procedures for requesting changes and indicating bugs in the product.

USER

ELEMENTS OF CONFIGURATION MANAGEMENT SYSTEM

- Susan Dart [Dar01] identifies four important elements that shoul exist when a CM system is developed:

- **Component elements—** A set of tools coupled within a file management system (e.g., a database) that enables access to an management of each software configuration item.
- **Process elements**—A collection of procedures and tasks that defin an effective approach to change management (and related activities)
- **Construction elements**—A set of tools that automate the construction of software by ensuring that the proper set of validate components (i.e., the correct version) have been assembled.
- **Human elements—** A set of tools and process features used by th software team to implement effective SCM.

BASELINES

- A baseline is a software configuration management concept that helps you to control change without seriously impeding justifiable change. The IEEE 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.
- The following figure consists of:

  Software engineering tasks

  Technical reviews

  Approved

  Stored

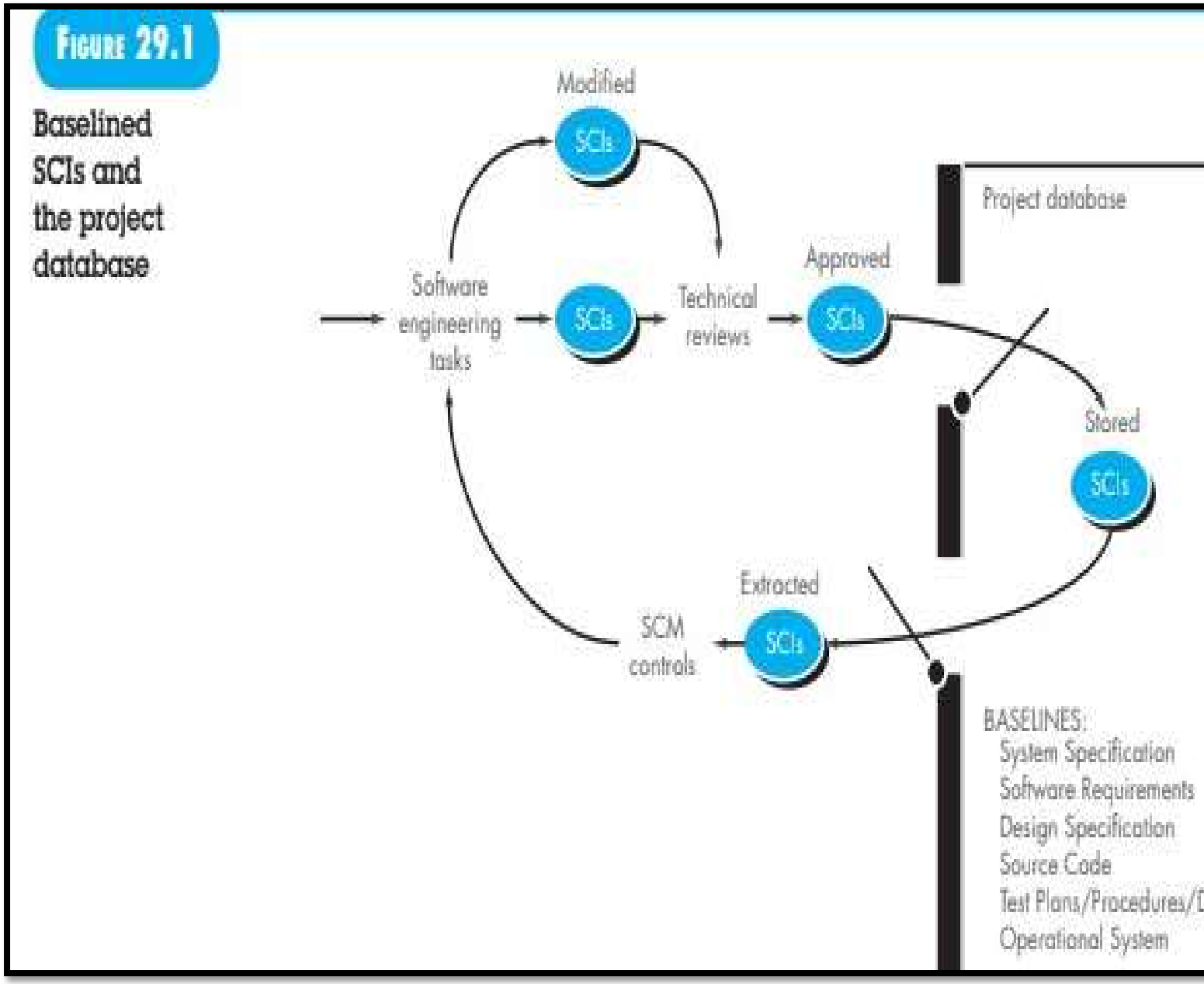
  Extracted

  SCI controls

  Modified

FIGURE 29.1

Baselined SCIs and the project database



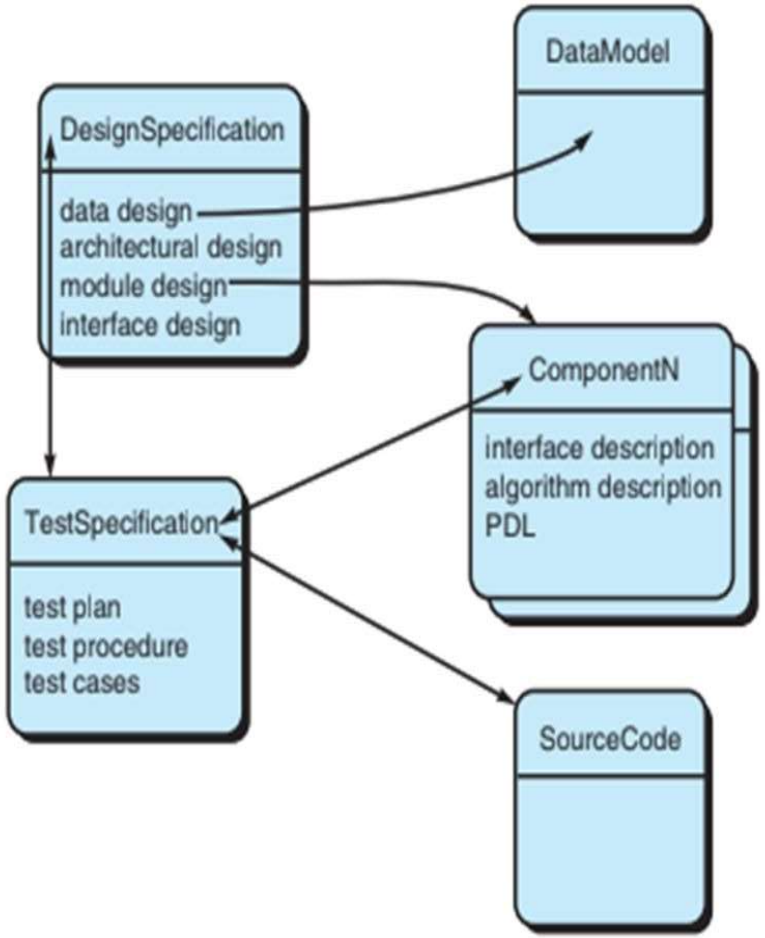
SOFTWARE CONFIGURATION ITEMS

- In reality, SCIs are organized to form configuration objects that may be cataloged 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 29.2 , the configuration objects, **Design Specification**, **Data Model**, **Component N**, **Source Code**, and **Test Specification** are each defined separately.
- In the fig 29.2 shows the management of dependencies & changes.

MANAGEMENT OF DEPENDENCIES AND CHANGES

FIGURE 29.2

Configuration objects



MANAGEMENT OF DEPENDENCIES AND CHANGES

- However, each of the objects is related to the others as shown by the arrows.
- 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 you to determine what other objects (and SCIs) might be affected.

SOFTWARE CONFIGARATION MANAGEMENT PRO

(SCM process)

- SCM process
- Task of SCM Process

✔ Identification of Objects in the Software Configuration

✔ Version Control

✔ Change Control

✔ Configuration Audit

✔ Status Reporting

- Participants of SCM process

The software configuration management proc
defines a series of tasks that have four prima
objectives:

1) To identify all items that collectively define
software configuration.

2) To manage changes to one or more of the
items.

3) To facilitate the construction of different ve
of an application.

4) To ensure that software quality is maintain
the configuration evolves over time

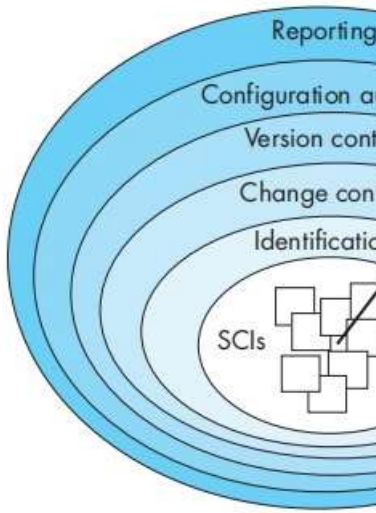
The SCM process must be developed in such a wa
the software team must answer the following set of
questions:

1. How does the software team identify the softwar
   configuration items?
2. How does the software team control the change
   software before and after delivering it to the cust
3. How does the software team manage the versio
   program in the software package?
4. How does the team get ensured that the change
   properly?
5. Who is responsible for approving the changes in
   software?

The answer to this questions lead the defin
of Five tasks of SCM Process:

1) Identification of Objects in the Software Configuration
2) Version Control
3) Change Control
4) Configuration Audit
5) Status Reporting

• SCM tasks can be viewed as conce
• SCIs flow outward through these la
their useful life, ultimately becoming p
software configuration of one or more
an application or system.



•As an SCI moves through a layer, the actiotask
may or may not be applicablens implied by eac
SCM

- For example, when a new SCI is create must be identified.
- However, if no changes are requested fo SCI, the change control layer does not a The SCI is assigned to a specific versio the software.
- A record of the SCI (its name, creation d version designation, etc.) is maintained f configuration auditing purposes and repo to those with a need to know .

- To control and manage software configuratio items, each should be separately named and then organized using an object-oriented approach.
- Two types of objects can be identified :

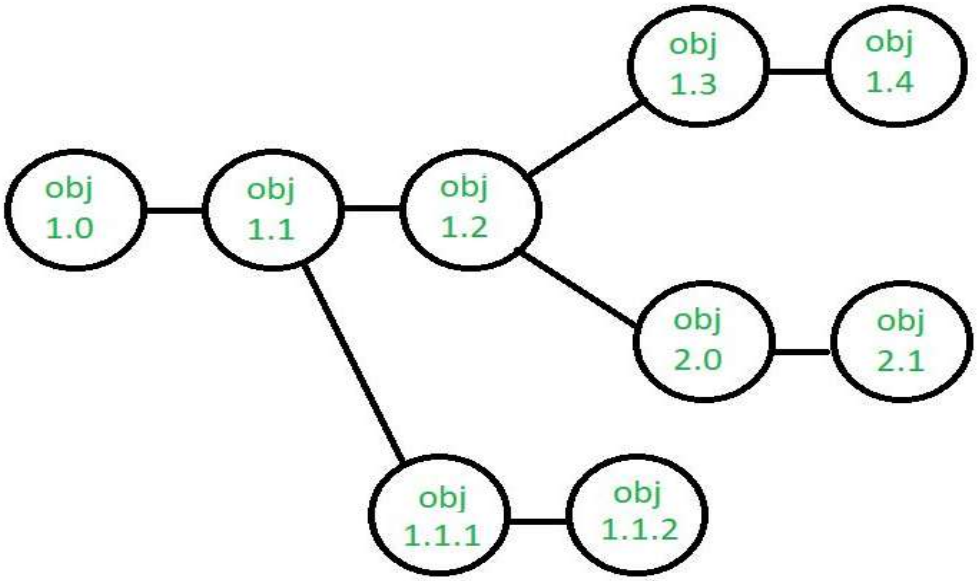
1) Basic objects,2)Aggregate objects

Basic objects: A basic object is a unit of information that you create during analysis, d code, or test.

- For example, a basic object might be a secti a requirements specifi- cation, part of a desig

- Aggregate object:An aggregate object is a collection of basic objects and other aggregate objects.
- Each object has a set of distinct features that identify it uniquely(a name, a description, a lis resources & a realization)
- The object name is a character string that iden the object uniquley.
- The object description is a list of data items tha identify the SCI type represented by the objec project identifier, and change and/or version information.
- For example, using the simple notation:

- Creating versions/specifications of the existi products to build new products form the help system.
- A new version is defined when major change been made to



- A version control system implements or is direct integrated with four major capabilities:

- a project database (repository) that stores all rel configuration objects.
- a version management capability that stores all versions of a configuration object (or enables an version to be constructed using differences from versions)
- a make facility that enables you to collect all rel configuration objects and construct a specific ve the software.
- an issues tracking (also called bug tracking) ca that enables the team to record and track the sta

- For a large software project, uncontrolled change rapidly leads to chaos.
- change control combines human procedures and automated tools.

□ ECO(engineering change order)

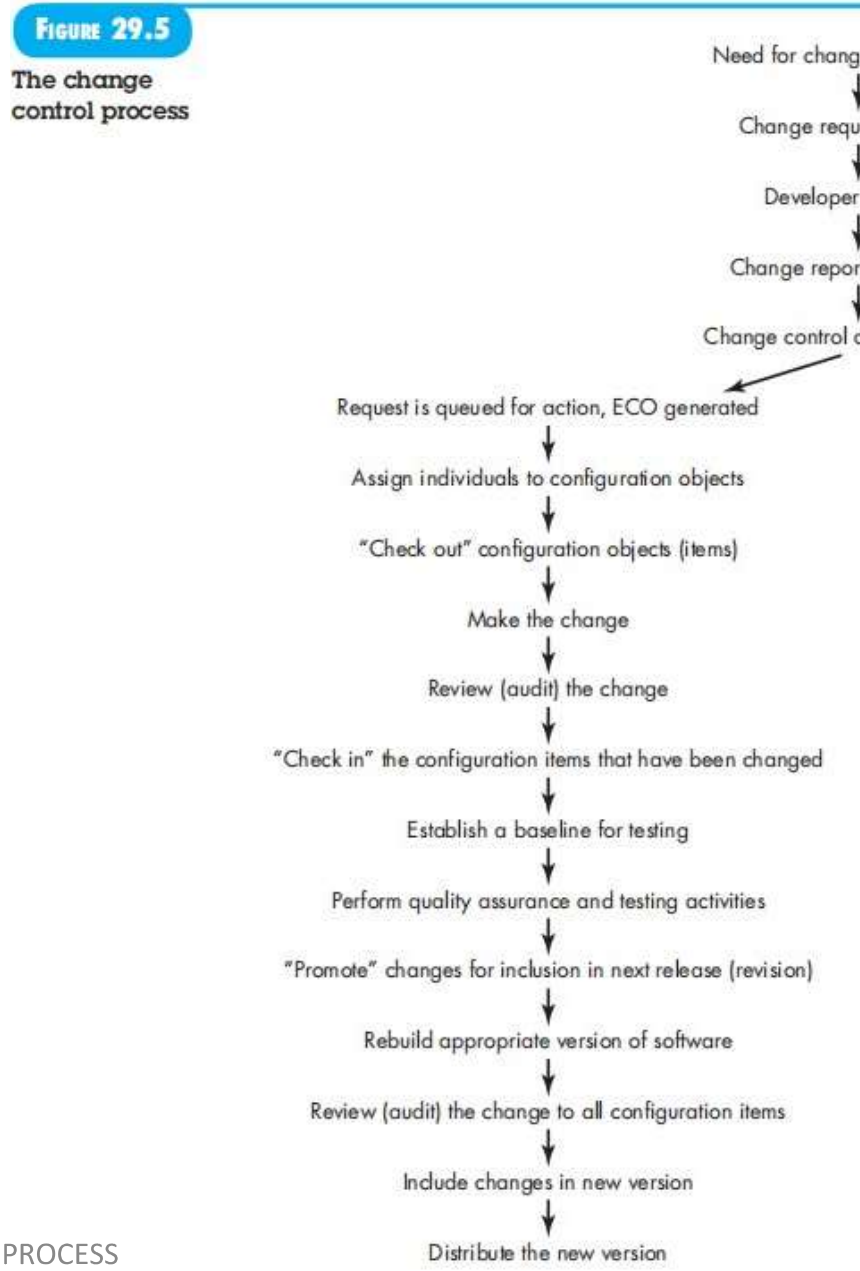


FIGURE 29.5 The change control process

- Software configuration audit verify that all the software pr satisfies the baseline needs.it ensure that what is built? , delivered?
- TO ensure that changes has been properly implemented:

i. Formal Technical Reviews

ii. Software Configuration Audit

 Formal Technical Reviews

- software quality assurance (SQA) activity performed by software engineers(and others)
- FTR serves as attraining ground,enabling junior engineer observe different approaches to software analysis,design implementation

 Software Configuration Audit

- The audit asks and answers the following questi

1. Has the change specified in the ECO been mad
Have any additional modifications been incorpo

2. Has a technical review been conducted to asse
technical correctness?

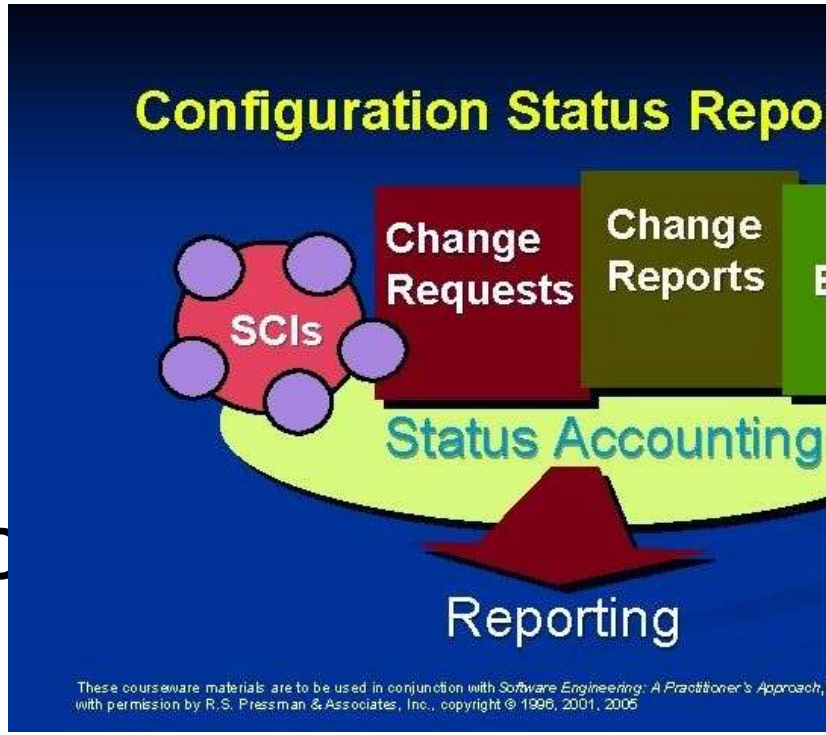
3. Has the software process been followed and ha
software engineering standards been properly app

4. Has the change been “highlighted” in the SCI?
the change date and change author been specifie
the attributes of the configuration object reflect the
change?

5. Have SCM procedures for noting the change,
recording it, and reporting it been followed?

- Configuration status reporting (sometimes ca status accounting) is an SCM task that answ following questions:

1. What happened?
2. Who did it?
3. When did it happen?
4. What else will be affec



- The status Reporting entry is made in the following conditions:
  - SCI is assigned new or updated identificatio change is approved by the CCA(change con authority)
  - Whenever the configuration audit is conduc the results are reported.
  - Output from CSR(configuration status report may be placed in an online database or web so that software developers or support staff access change information.

product METRICS FOR REQUIREMENT MODEL

- Function Based Metrics
- Questions
- Flow Model for(SafeHome) user interaction function
- Computing Function points
- Metrics for Specification Quality

METRICS: “A system or a standard measurement”.

- Technical work in software engineering begins with the creation of the requi model. It is a stage where requirements are derived and a foundation for desi established.

- The product metrics that provide insight into the quality of the analysis mode desirable.

- It is possible to adapt metrics that are often used for project estimation and a them in this context.

- These metrics examine the requirements model with the intent of predicting “size” of the resultant system.

- Size is sometimes (but not always) an indicator of design complexity and is a always an indicator of increased coding, integration, and testing effort.

Function Based Metrics:

- The function point (FP) metric can be used effectively as a means for measu the functionality delivered by a system.

- Using historical data, the FP metric can be used to

(1) Estimate the cost or effort required to design, code, and test the software;
(2) Predict the number of errors that will be encountered during testing; and
(3) Forecast the number of components and/or the number of projected source implemented system.

- Function points are derived using an empirical relationship based on counta measures of software's information domain and qualitative assessments of s complexity. Information domain values are defined in the following manner

- ***Number of external inputs (EIs):*** Each external input originates from a use transmitted from another application and provides distinct application-orien control information. Inputs are often used to update internal logical files (IL should be distinguished from inquiries, which are counted separately.

- ***Number of external outputs (EOs):*** Each external output is derived data wi application that provides information to the user. In this context external out reports, screens, error messages, and the like. Individual data items within a counted separately.

- ***Number of external inquiries (EQs*)**: An external inquiry is defined as an o that results in the generation of some immediate software response in the fo online output (often retrieved from an ILF).

- ***Number of internal logical files (ILFs)***: Each internal logical file is a log grouping of data that resides within the application's boundary and is mai via external inputs.
- ***Number of external interface files (EIFs):*** Each external interface file is grouping of data that resides external to the application but provides data be of use to the application.

Once these data have been collected, the table is completed and a complexi is associated with each count. Organizations that use function point methods criteria for determining whether a particular entry is simple, average, or com Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:

$$\mathbf{FP = count\ total \times [0.65 + 0.01 \times \sum(F_i)]}$$

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations
14. Is the application designed to facilitate change and ease of use by the use

- Each of these questions is answered using an ordinal scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). The constant values weighting factors that are applied to information domain counts are determined

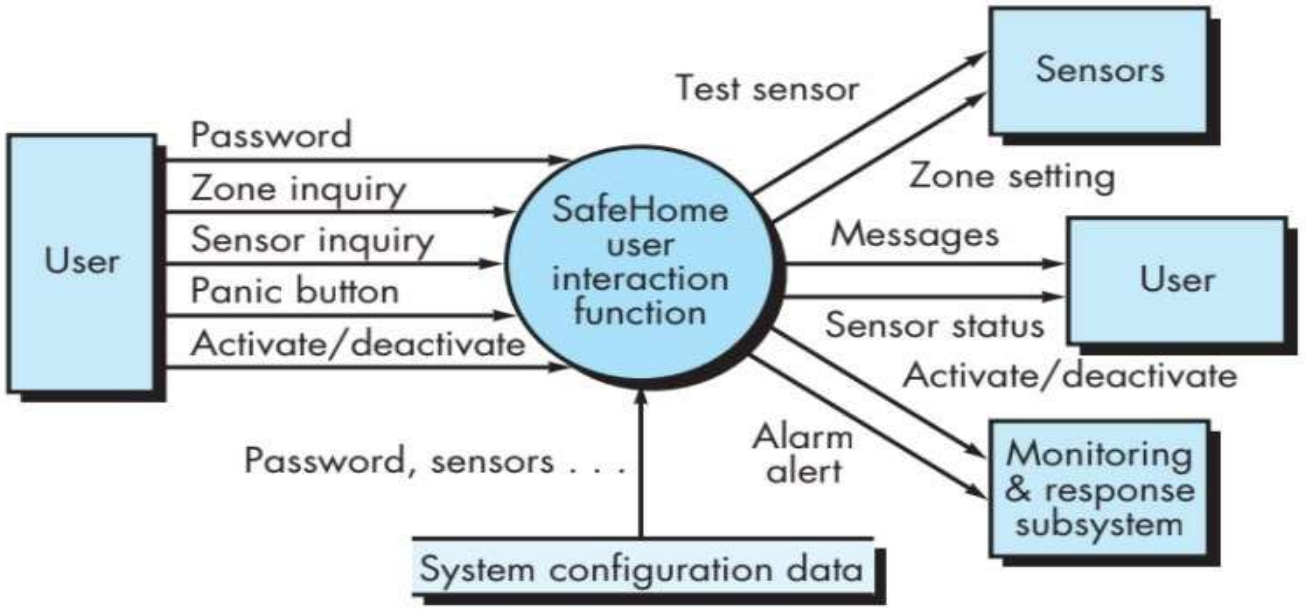
- To illustrate the use of the FP metric in this context , let us consider a simple diagram for a user interaction function within *SafeHome* software, represente in Figure 30.2 .

- The function manages user interaction, accepting a user password to activate the system, and allows inquiries on the status of security zones and various se

- The function displays a series of prompting messages and sends appropriate c to various components of the security system.

FIGURE 30.2

A flow model for *SafeHome* user interaction function



- The flow diagram is evaluated to determine a set of key information domain m required for computation of the function point metric.
- Three *external inputs*— password, panic button, and activate/deactivate along *external inquiries*— zone inquiry and sensor inquiry. One ILF(system configu shown. Two *external output* — messages and sensor status and four EIFs(test setting, activate/deactivate, and alarm alert) are also present.

FIGURE 30.3 Computing function points

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	3	3	4	6	=	9
External Outputs (EOs)	2	3	4	5	7	=	8
External Inquiries (EQs)	2	3	3	4	6	=	6
Internal Logical Files (ILFs)	1	3	7	10	15	=	7
External Interface Files (EIFs)	4	3	5	7	10	=	20
Count total							50

Where count total is sum of all FP entries obtained.

The count total shown in Figure 30.3 is

let us assume that $\sum(F_i)$ is 46 (a moderately complex product). Therefore,

$$\mathbf{FP=count\ total \times[0.65+0.01\times\sum(F_i)]}$$

$$\mathbf{FP=50\times[0.65+(0.01\times46)]=56}$$

- Based on the projected FP value derived from the requirements model, the p team can estimate the overall implemented size of the *SafeHome* user intera function.

- Assume that past data indicates that one FP translates into 60 lines of code (a object-oriented language is to be used) and that 12 FPs are produced for each person-month of effort.

- These historical data provide the project manager with important planning in that is based on the requirements model rather than preliminary estimates.

- Assume further that past projects have found an average of three errors per f during requirements and design reviews and four errors per function point du and integration testing.

- These data can ultimately help us to assess the completeness of our review a activities.

- Uemura and his colleagues [Uem99] suggest that function points can also be from UML class and sequence diagrams.

Metrics for Specification Quality

- Davis and his colleagues [Dav93] propose a list of characteristics that can be assess the quality of the requirements model and the corresponding requirem specification:

"*specificity(lack of ambiguity), completeness, correctness, understandability, v internal and external consistency, achievability, concision, traceability, modifi precision, and reusability".*

- Although many of these characteristics appear to be qualitative in nature , ea represented using one or more metrics .For example, if we assume that there requirements in a specification, such that

$$n_r = n_f + n_{nf}$$

- Where n_f is the number of functional requirements and n_{nf} is the number of n (e.g., performance) requirements.

- To determine the specificity (lack of ambiguity) of requirements, Davis and suggest a metric that is based on the consistency of the reviewers' interpretatio requirement:

$$Q_1 = \frac{n_{ui}}{n_r}$$

- Where n_{ui} is the number of requirements for which all reviewers had identic interpretations. The closer the value of Q to 1, the lower is the ambiguity of specification.

- The completeness of functional requirements can be determined by computi

$$Q_2 = \frac{n_u}{n_i \times n_s}$$

- Where n_u is the number of unique function requirements, n_i is the number o (stimuli)defined or implied by the specification, and n_s is the number of stat specified. The Q_2 ratio measures the percentage of necessary function that h specified for a system.

- However, it does not address nonfunctional requirements. To incorporate t overall metric for completeness, you must consider the degree to which req have been validated:

$$Q_3 = \frac{n_c}{n_c \times n_{nv}}$$

where n_c is the number of requirements that have been validated as correct an number of requirements that have not yet been validated.

METRICS FOR DESIGN MODEL

- Architectural Design Metrics
- Metrics for Object-Oriented Design
- Class-Oriented Design – The CK Metrics Suite
- Class-Oriented Design – The Mood Metrics Suite
- Component-level design metrics

ARCHITECTURAL DESIGN MO

- It focus on characteristics of the program Archit
  effectiveness of modules Or components within
  architecture.
- Card and Glass defines three software design co
  measures:

1. Structural complexity
2. Data complexity
3. System complexity

STRUCTAL COMPLEXITY:

A Structural complexity of module is defined in manner as

S(i) = f 2 out(i)

Where f out is the fan-out module of i

DATA COMPLEXITY:

It provides an indication of the complexity in the internal i for a module i and is defined as:

D(i) = V(i)/fout(i)+1

SYSTEM COMPLEXITY:

It is defined as the sum of the structural and data complex specified as:

C(i) = S(i) + D(i)

FENTON

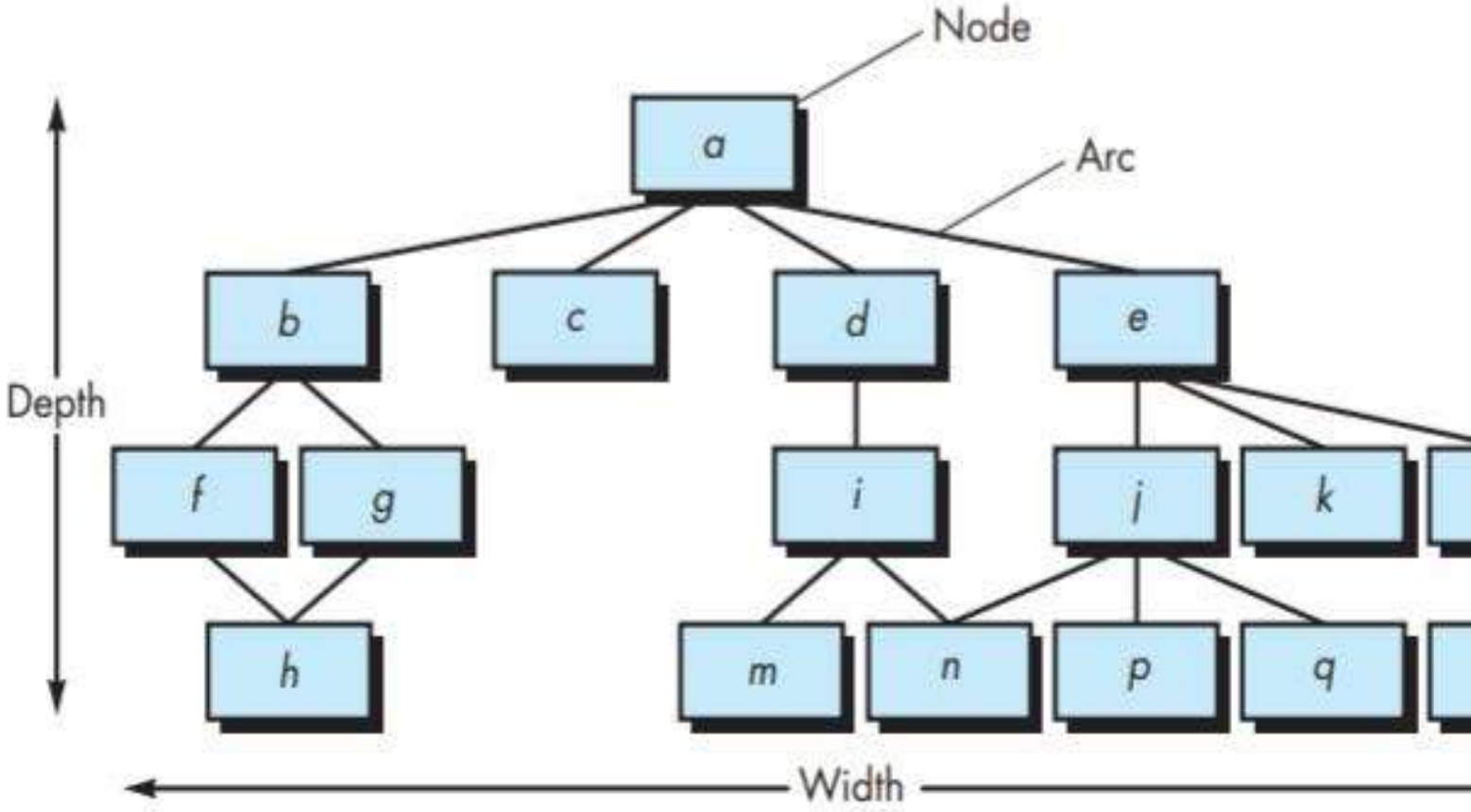
It suggest a number of simple morphology (shape) metri
enable different program architecture to be compared us
set of straight forward dimensions

Size = n+a

n= number of nodes

a= number of arcs

MORPHOLOGY METRICS



METRICS FOR OBJECT-ORIENTED DE

In a detailed treatment of software metrics for OO systems, Whitmire [Wh describes nine distinct and measurable characteristics of an OO design.

Size is defined by taking a static count of OO entities such as classes or ope coupled with the depth of an inheritance tree.

Complexity is defi ned in terms of structural characteristics by examining h of an OO design are inter-related to one another.

Coupling is measured by counting physical connections between element design (e.g., the number of collaborations between classes or the number messages passed between objects).

METRICS FOR OBJECT-ORIENTED DESIGN

- Sufficiency is “the degree to which an abstraction [class] possesses th required of it . . .”
- Completeness determines whether a class delivers the set of properti reflect the needs of the problem domain.
- Cohesion is determined by examining whether all operations work to achieve a single, well-defined purpose.
- Primitiveness is the degree to which an operation is atomic—that is, t operation cannot be constructed out of a sequence of other operatio contained within a class.
- Similarity determines the degree to which two or more classes are sir terms of their structure, function, behavior, or purpose. Volatility me likelihood that a change will occur.

Class-Oriented Metrics—The CK Metrics

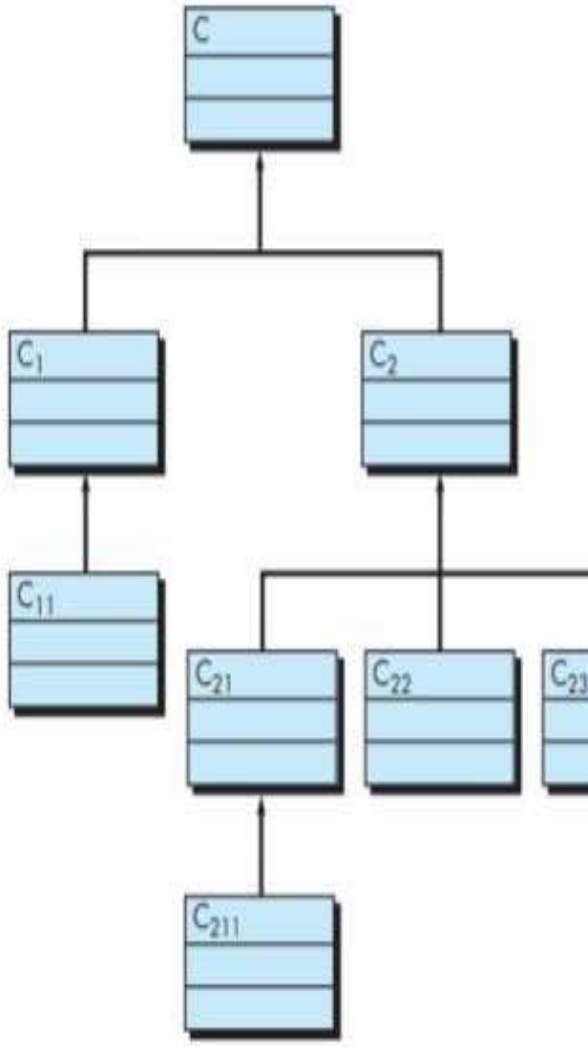
- Chidamber and Kemerer (CK) have proposed one of the most referenced sets of OO software metrics. Often referred to as metrics suite, the authors have proposed six class-based desi metrics for OO systems.
- Weighted methods per class (WMC). Assume that n methods complexity c1, c2, . . . , cn are defined for a class C. The specif complexity metric that is chosen (e.g., cyclomatic complexity be normalized so that nominal complexity for a method take value of 1.0.
- WMC = S Ci

Class-Oriented Metrics—The CK Metrics S

- **Depth of the inheritance tree (DIT). This metric is "the maximum le from the node to the root of the tree". Referring to Figure, the valu for the class hierarchy shown is 4.**
- **As DIT grows, it is likely that lower-level classes will inherit many**
- **This leads to potential diffi culties when attempting to predict the of a class. A deep class hierarchy (DIT is large) also leads to greater complexity. On the positive side, large DIT values imply that many may be reused.**

Class-Oriented Metrics—The CK Met Suite

- **Number of children (NOC).** The subclasses that are immediately subordinate to a class in the class hierarchy are termed its children. Referring to Figure, class C2 has three children—subclasses C21, C22, and C23 As NOC increases, the amount of testin (required to exercise each child in its operational context) will also increase.



Class-Oriented Metrics—The CK M Suite

Coupling between object classes (CBO). The CRC model may be used to determine the value for CBO. In essence, CBO is the number of collaborations listed for a class on its CRC index card. As CBO increases, it i likely that the reusability of a class will decrease.

Response for a class (RFC). The response set of a class is "a set of methods that can potentially be executed in response to a message receiv by an object of that class". RFC is the number of methods in the response set.

Lack of cohesion in methods (LCOM). Each method within a class accesses one or more attributes (also called instance variables). LCOM is th number of methods that access one or more of the same attributes. If no methods access the same attributes, then LCOM = 0.

Class-Oriented Metrics—The MOOD Me Suite

- Method inheritance factor (MIF). The degree to which the class architecture of an OO system makes use of inheritance for bo methods (operations) and attributes is defined as
- MIF = SMi(Ci) /Sma(Ci)
- where the summation occurs over i = 1 to TC.
- And also Ma(Ci) = Md(Ci) + Mi(Ci)
- Where Ma(Ci) = number of methods that can be invoked in association with Ci
- Md(Ci) = number of methods declared in the class Ci.
- Mi(Ci) = number of methods inherited (and not overridden) i

Class-Oriented Metrics—T MOOD Metrics Suite

- **Coupling factor (CF).** The MOOD metrics suite defi nes coupli following way:
- CF = Si Sj is_client (Ci, Cj)/TC 2 – TC
- where the summations occur over i = 1 to TC and j = 1 to TC. The client
- = 1, if and only if a relationship exists between the client class C server class Cs, and Cc != Cs
- = 0, otherwise

Component-level design metr

- Component-level design metrics for conventional softwar components focus on internal characteristics of a softwar component and include measures of the “three Cs”—mod cohesion , coupling, and complexity.
- These measures can help you judge the quality of a component-level design.
- Component-level design metrics may be applied once a pr design has been developed and are “glass box” in the sens they require knowledge of the inner workings of the modu consideration.

METRICS FOR SOURCE CODE,TESTING AND MAINTENANCE

Metrics for source code

Metrics for Testing

Metrics for Maintenance

Metrics

- **Metric is a quantitative measure of the degree to which a component or a process possesses a given attributes.**
- **Metrics provide the insight necessary to create effective requirements and design models.**
- **It must be simple and computable, consistent.**

Halstead's Theory

- **Halstead's theory of "software science" [Hal77] proposed the fi analytical "Quantitative laws" for computer software**
- **Derived once the design phase is complete and code is generated**
- **The measures are:**
- **n1= number of distinct operators that appear in a program**
- **n2= number of distinct operands that appear in a program**
- **N1= total number of operator occurrences**
- **N2= total number of operand occurrences**
- **By using these measures, Halstead developed an expression for c program length, program volume, program difficulty, developm**

Contd….

- **Halstead shows Program length (N)can be calculated by using equation:**
- **$N = n1 \log_2 n1 + n2 \log_2 n2$**
- **Program volume(V) can be calculated by using equation:**
- **$V = N \log_2 (n1 + n2)$**
- **It should be noted that V will vary with programming language and represents the volume of information required to specify a program**
- **Volume ratio (L) can be calculated by using equation:**
- **$L=(2/n1)*(n2/N2)$**

Metrics For Testing

- **In general, testers must rely on analysis, design, and code m to guide them in the design and execution of test cases.**
- **Architectural design metrics provide information on the ea difficulty associated with integration testing and the need fo specialized testing software.**
- **Cyclomatic complexity (a component-level design metric) li the core of basis path testing, a test-case design method**

Cont..,

- **In addition, cyclomatic complexity can be used to target modules as candidates for extensive unit testing. Modules with high cyclomatic complexity are more likely to be error prone than modules whose cyclomatic complexity is lower**

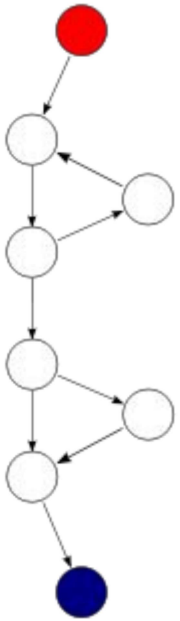


Fig:01
Cyclomatic complexity

Halstead Metrics Applied to Testing

- **Testing effort can be estimated using metrics derived from Hals measures.**
- **Program volume(V) and program level (PL), Halstead effort e computed as**

$$PL = 1/ [(n1 /2) *(N2 /n2)]$$

$$e = V/ PL$$

- **The percentage of overall testing effort to be allocated to a mod can be estimated using the following relationship:**
- **Percentage of testing effort (k) = e (k)/ Σe(i)**

where e (k) is computed for module k using Equations and the summation in the denominator of Equation is the sum of Halstead across all modules of the system.

Metrics for Object-Oriented Testing

- **The OO design metrics provide an indication of design quality. They provide a general indication of the amount of testing effort required t exercise an OO system.**
- **Binder suggests a broad array of design metrics that have a direct inf on the "testability" of an OO system. The metrics consider aspects of encapsulation and inheritance.**
- **Lack of cohesion in methods (LCOM):The higher the value of LCOM more states must be tested to ensure that methods do not generate sid**
- **Percent public and protected (PAP): Public attributes are inherited f other classes and therefore are visible to those classes. Protected attri accessible to methods in subclasses. High values for PAP increase the likelihood of side effects among classes.**

Cont..,

- **Public access to data members (PAD):This metric indicates the number of classes (or methods) that can access another class's attributes. High values for PAD lead to the potential for side effects among classes.**
- **Number of root classes (NOR): Hierarchies that are described in the design model. NOR increases, testing effort also increases.**
- **Fan-in (FIN):Indication of multiple inheritance.**
- **Number of children (NOC) and depth of the inheritanc tree (DIT):superclass methods will have to be retested each subclass.**

Metrics for Maintenance

- **All the software metrics used for the development of new software an maintenance of existing software.**
- **IEEE suggests a software maturity index (SMI) that provides an indi the stability of a software product.**
- **The following information is determined:**

  MT = number of modules in the current release

  Fc = number of modules in the current release that have been change

  Fa = number of modules in the current release that have been added

Fd = number of modules from the preceding release that were deleted i current release

- **The software maturity index is computed in the following manner:**

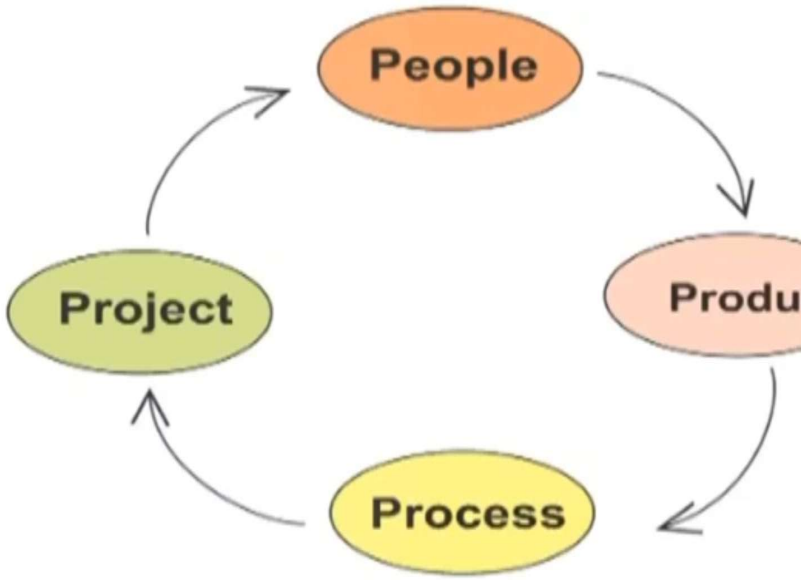
$$SMI = [MT - (Fa + Fc + Fd)]/ MT$$

Chapter - 2
MANAGING SOFTWARE PROJECTS

THE PROJECT MANAGEMENT SPECTRUM

Management spectrum

- ❑ People
- ❑ Product
- ❑ Process
- ❑ project



PROJECT MANAGEMENT:

- Project management involves the planning, monitoring, and control of the people, process and eve

THE MANAGEMENT SPECTRUM:

- In software engineering, the management spectrum describes the management of software projec
- The management of a software project starts from requirement analysis and finishes based on the product ,it may or may not end because almost all software products faces changes and requires su

Effective software project management focuses mainly on the four ps:

- People
- Product
- Process
- Project

- Here ,the manager of the project has to control all of these p's to have a smooth flow in the project reach the goal.

The four p's of management spectrum are:-

The People:-

- People of a project includes from manager to devolper, from customer to end user.

❖ The People:-

- People of a project includes from manager to developer, from cust end user. But mainly people of a project highlight the developers.
- In fact, the “people factor” is so important that the software engine institute has developed a people capability maturity model(People-C
- The people capability maturity model defines the following key prac for software people: staffing communication and coordination.

work-environment, performance management, training , compensation and development, career development, workgroup development and team/culture development and others.

- Organizations that achieve high levels of people-CMM maturity have a higher likelihood of implementing effective software project management practices.

❖ The product::-

- Product is any software that has to be developed.
- Before a project can be planned, product objectives and scope should be establishe alternative solutions should be considered ,technical and management constraints s identified .
- Without this information , it is impossible to define reasonable and accurate estima cost ,an effective assessment of a risk ,a realistic breakdown of project tasks or a ma project schedule that provides a meaningful indication of progress.

❖ The process:-

- A software process provides the framework from which a comprehensive plan for software development can be established .
- A small number of framework activities are applicable to all software projects , regardless of their size and complexity.
- A number of different task sets- tasks, milestones, work products, and quality assurance points – enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team
- Finally ,Umbrella activities overlay the process model.

- Umbrella activities are independent of any one framework activity and occur throughou process.

❖ The Project:-

- The project includes all and everything of the total development process and to avoid p failure the manager has to take some steps, has to be concerned about some common warnings.

PROCESS & PROJECT

INTRODUCTION:

- Suggested by Barry Boehm in one of his papers
- Excellent planning outline for project managers and software team
- Applicable to all sizes of software projects
- It is an approach to address
  - Project objectives
  - Milestones&schedule
  - Management&technical approach
  - Required resources

- **THE PROCESS:**
- The framework activities that characterize the software process are applicable software projects.
- The problem is to select the process model that is appropriate for the softwar engineering by your project team.
- Your team must decide which process model is most appropriate for the custo who have requested the product and the people who will do the work.
- The characteristics of the product itself, the project environment in which the software team works.
- When a process model has been selected , the team them defines a prelimina project plan based on the set of process decomposition begins.

THE PROJECT:

- In order to manage a successful software project, you have to understand wh go wrong so that problems can be avoided.
- In an excellent paper on software projects.
- John Reel defines signs that indicate that an information systems project .
- This leads to a project with a poorly defined scope .
- In other project .
- Changes are managed poorly.
- Sometimes , the chosen technology changes or business needs shift and management sponsorship is lost.
- And finally, there are developers who never seem to term from their mista
- The seeds that leads to the 90-90 rules are contained in the signs noted in th preceding list.

W5HH Principle:

- Principle is applicable regardless of the size or complexity of a software pro
- The questions noted provide you can and you team with an excellent plann

❖**Why** is the system being developed?

The business purpose justify the expenditure of people, time and money.

❖**What** will be done?

The fast set required for the project is defined.

❖**When** will it be done?

The team establishes a project schedule by identifying when project tasks ar
connected and when milestones are to be reached.

❖**Who** is responsible for a function?

The role and responsibility of each member of the software team is defined.

❖**Where** are they located organizationally?

Not all roles& responsiblities reside within software practitioners

The customers users and other stakeholders also have responsiblities.

W5HH Principle:

- **How** will the job be done technically and managerically?
  once product scope is established, a management and technical stralegy for the project must be defined.

- **How** much of each resource is needed?
  the answer to this question is derived by developing estimates based on answers to earlier questions.

PROCESS AND PROJECT METRIC

❖ AGENDA

- ✔ *Metrics in the process domain*
- ✔ *Metrics in the project domain*
- ✔ *Process metrics and Software Process Improvement*
- ✔ *Statistical software process improvement*
- ✔ *Project Metrics*

❖ Why do we measure….???

- In their guidebook on software measurement, Park, Goethert, and Florac note the reasons that we measure:
  - To characterize
  - To evaluate
  - To predict
  - To
  - improve
  - Measurement is a management tool. If conducted properly, it provides a project manager with insight.

  it assists the project manager and the software team in making decisions that will lead to a successful project.

❖ Metrics in the Project Domains

- Project Metrics enable a software project manager t

1) Assess the status of an ongoing project
2) Track potential risks
3) Uncover problem areas before they go “critical”
4) Adjust work flow or tasks, and
5) Evaluate the project team’s ability to control quality of software work products

Measures that are collected by project team and converted into metrics for use during a project can also be transmitted to those with responsibility for software process improvement.

For this reason, many of the same metrics are used in both the process and project domains.

- We measure the effectiveness of a software process indirectly.
- That is, we derive a set of metrics based on the outcomes that can b derived from the of the process.

OUTCOMES INCLUDE

- Errors uncovered before release of the software
- Defects delivered to and reported by the end
- users Work products delivered(productivity)
- Human effort expended
- Calendar time expended
- Conformance to the
- schedule

Time and effort to complete each generic activity

We also derive process metrics by measuring the characteristics of specific software engineering tasks

Metrics in the Process Domain

Process metrics are collected across all projects and over long periods of time

- They are used for making strategic decisions
- The intent is to provide a set of process indicators that lead to long-term software process improvement.
- The only way to know how/where improve any process is to measure specific attributes of the process.
- Develop a set of meaningful metrics based on these attributes

Use the metrics to provide indicators that will lead to a strategy for improvement.

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
  - Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and m that will be used to achieve them.
  - Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered.

"negative." These data are merely an indicator for process improvement.

- Don't obsess on a single metric to the exclusion of other

❖ *Process metrics and Software Process Improvement*

- Process at the center of a triangle connecting 3 factors that have a profound influence on software quality and organizational performanc
- The skill and motivation of people has been shown to be the most influential factor in quality and performance.
- The complexity of the product can have a substantial impact on quality and team performance.
- The technology(i.e., software engineering

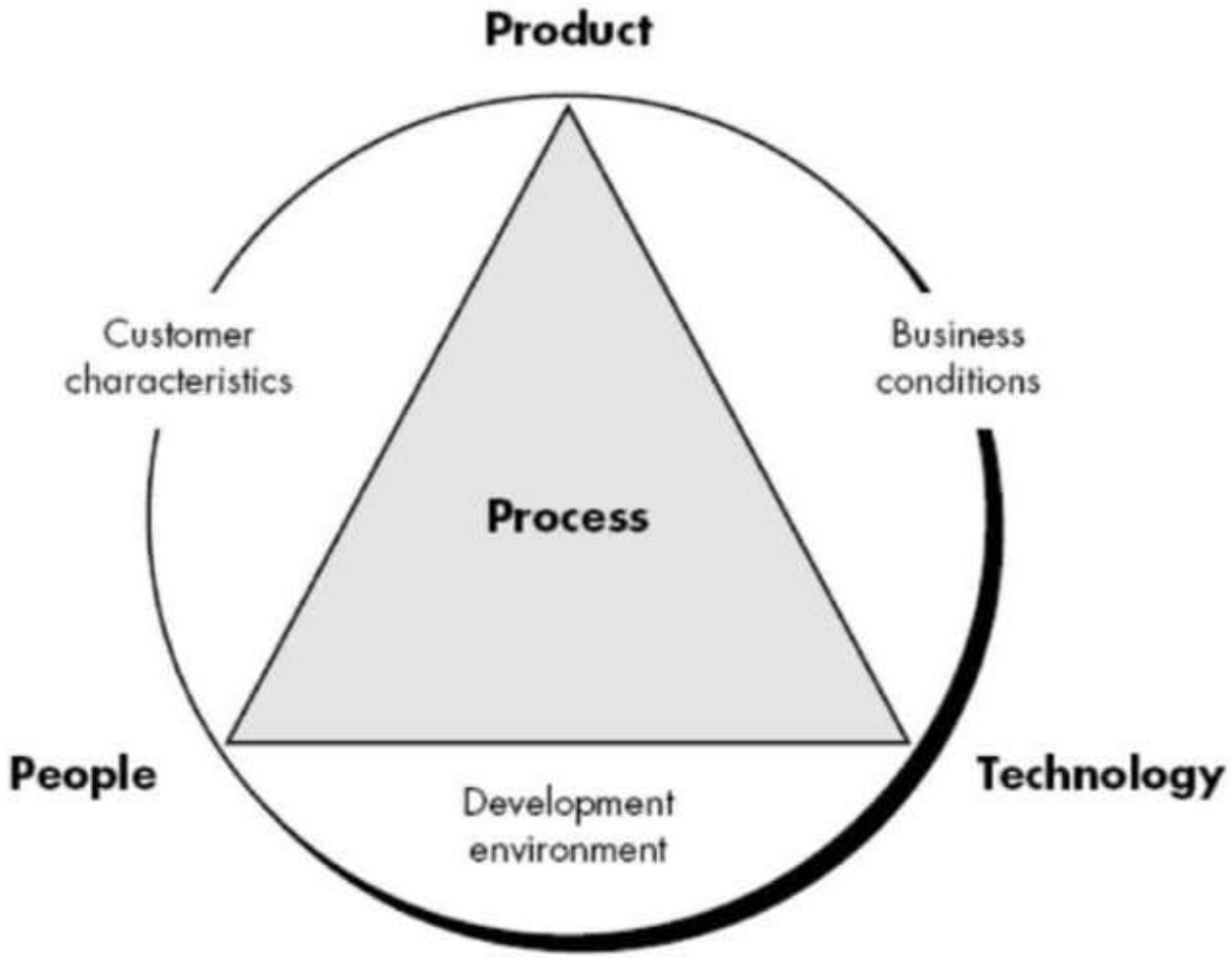
❖ Process Metrics and Software Process Improvement

Improvement

- Process triangle exists within a circle of environmental conditions that include the development environment (e.g., integrated software tools), business conditions(e.g., deadlines , busines rules) and customer characteristics(e.g., easy of communication and collaboration).
- To measure the efficacy of a software process indirectly. Derive metrics based on the outcomes th can be derived from the process.
  - Outcomes include measures of
  - errors uncovered before release of the software,
  - defects delivered to and reported by end users,
  - work products delivered (productivity)

- Determinants for software quality and organizational effectiveness

FIGURE 4.1
Determinants for software quality and organizational effectiveness (adapted from [PAU94])



▪ *Statistical Software Process Improvement*

- As an organization becomes more comfortable with the collection and use of process metrics, th derivation of simple indicators gives way to a mor rigorous approach called Statistical software process improvement (SSPI).
- In essence, SSPI uses software failure analysis to collect information about all errors and defects.

  - Error –Some flaw in a s/w engineering work product that is uncovered before the s/w is delivered to the end-us

  - Defect –A flaw that is uncovered after delivery the end-user.

Project Metrics

- The intent of project metrics is twofold.

-First, these metrics are used to minimize the development schedule

by making the adjustments necessary to avoid delays and mitigate potential problems and risks.

-Second, project metrics are used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.

As quality improves, defects are minimized, and as the defect

count goes down, the amount of rework required during the project is also reduced. This leads to a reduction in overall project cost.

SOFTWARE MEASUREMENT

Tyeps of software measurements

- **Direct measurements**
- Indirect measurements

Direct Measures are the one are measured directly from t software project itself.

- **Easy to collect**

E.g. Cost, Effort, Lines of cod (LOC), Execution Speed, Mem size, Defects etc.

Tyeps of software measurements

- **Direct measurements**
  Indirect measurements

The Indirect measures
not measured directly
complexity, reliability
maintainability etc.

- **Indirect measures of**
  product:
  Functionality
  Quality
  Complexity
  Reliability
  Maintainability

Example

2 different project teams are working to record errors in a s process:

Team A – Finds 342 errors during software process before r

Team B-Finds 184 errors.

Which team do you think is more effective in finding errors???????????

To answer this we need to know the size & complexity of the projects. But if normalize the measures,

it is possible to create software metrics that enable comparison to broader organizational averages.

We have various types of metrics:

***Size - Oriented Metrics.**

***Function - Oriented Metrics.**

***Object Oriented Metrics.**

***Use case – Oriented Metrics.**

*** Webapp Project Metrics.**

SIZE – ORIENTED METRICS

*** Derived by normalizing quality and/or productivity measures considering the size of the software produced.**
*** In Size-oriented metrics, Line of Code(LOC) is considered as th normalized value.**
*** In Size-oriented metrics, Line of Code have been used as a productivity metrics in many organizations.**

*** Productivity Comparison across organization are impossible u each organization uses same method for counting lines of**
*** These Size oriented metrics are not Universally accepted.**

*** Size-oriented metrics depend on the programming language u**

Size oriented Metrics example

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Size oriented Metrics example

Referring to the table entry for project alpha:

LOC = lines ot code = 12,100

Effort = 24

Pp.Doc = Pages documentation = 365

Erorrs = 134 (Which are always recorded before the delivery of the product)

Defects = 29 (Which we find out after the released of the product to customer)

People =3 (Who are working in the software development Project - alpha)

Size-Oriented Metrics

many other metrics can be computed:

. • **Errors per KLOC (thousand lines of code)**
- **Defects per KLOC**
- **$ per KLOC**
- **Pages of documentation per KLOC**

In addition, other interesting metrics can be computed:

- **Errors per person-month**
- **KLOC per person-month**
- **$ per page of documentation**

Function-Oriented Metrics

- Function-oriented software metrics use a measure of the functionality del the application as a normalization value.
- The most widely used function-oriented metric is the function point (FP).
- Computation of the function point is based on characteristics of the softw information domain and complexity.
- The Size of the system can be measure directly but the functionality canno measured directly.
  Function point is not a single characteristics, But it's a combination of the characteristic.
  Function-oriented metrics are independent of implementation language.
  Function-oriented metrics are used to evaluate Cost(S) and Productivity v
- Function-orientedmetrics are used for data processing system,

Reconciling LOC and FP Metrics

The relationship between lines of code and function points depends upon the pro-gramming language that is used to implement the software and the quality of the design. A number of studies have attempted to relate FP and LOC measures. The following table4 [QSM02] provides rough estimates of the average number of lines of code required t build one function point in various programming languages

Programming Language	LOC per Function Point			
	Avg.	Median	Low	High
Ada	154	—	104	205
ASP	56	50	32	106
Assembler	337	315	91	694
C	148	107	22	704
C++	59	53	20	178
C#	58	59	51	704
COBOL	80	78	8	400
ColdFusion	68	56	52	105
DBase IV	52	—	—	—
Easytrieve+	33	34	25	41
Focus	43	42	32	56
FORTRAN	90	118	35	—
FoxPro	32	35	25	35
HTML	43	42	35	53
Informix	42	31	24	57
J2EE	57	50	50	67
Java	55	53	9	214
JavaScript	54	55	45	63
JSP	59	—	—	—
Lotus Notes	23	21	15	46

Programming Language	LOC per Function Point		
	Avg.	Median	Low
Mantis	71	27	22
Natural	51	53	34
.NET	60	60	60
Oracle	42	29	12
OracleDev2K	35	30	23
PeopleSoft	37	32	34
Perl	57	57	45
PL/1	58	57	27
Powerbuilder	28	22	8
RPG II/III	61	49	24
SAS	50	35	33
Smalltalk	26	19	10
SQL	31	37	13
VBScript	38	37	29
Visual Basic	50	52	14

- **LOC and FP measures are often used to derive productivity me**
  Function points and LOC-based metrics have been found to be relati
  ac-curate predictors of software development effort and cost. Howe
  order to use LOC and FP for estimation an historical baseline of info
  must be established.
  Within the context of process and project metrics, you should be co
  primarily with productivity and quality—measures of software dev
  "output" as a function of effort and time applied and measures of th
  for use" of the work products that are produced. For process improv
  and project planning purposes, your interest is historical. What was
  development productivity on past projects? What was the quality of
  software that was produced? How can past productivity and quality
  extrapolated to
  the present? How can it help us improve the process and plan new p
  more accurately?

Object-Oriented Metrics

Conventional software project metrics (LOC or FP) can be used to estimate object software projects. However, these metrics do not provide enough granularity for schedule and effort adjustments that are required as you iterate through an evol incremental process.

Lorenz and Kidd [Lor94] suggest the following set of metrics for OO projects:

Number of scenario scripts; Scenario scripts are a sequence of steps, which depic interaction between the user and the application. A number of scenarios is direct to application size and number of test cases that are developed to test the softwa is developed.

Number of key classes; Key classes are independent components, which are defin object-oriented analysis. As key classes form the core of the problem domain, the the effort required to develop software and the amount of 'reuse' feature to be a development process.

Number of support classes; Classes, which are required to implement the system indirectly related to the problem domain, are known as support classes. For exam interface classes and computation class are support classes. It is possible to devel support class for each key class. Like key classes, support classes indicate the effo to develop software and the amount of 'reuse' feature to be applied during the development process.

Object oriented Metrics

Average number of support classes per key class; Key classes are de early in the software project while support classes are defined thro the project. The estimation process is simplified if the average num support classes per key class is already known.

Number of subsystems; A collection of classes that supports a funct visible to the user is known as a subsystem. Identifying subsystems it easier to prepare a reasonable schedule in which work on subsys divided among project members.

The afore-mentioned metrics are collected along with other pr metrics like effort used, errors and defects detected, and so on. Aft organization completes a number of projects, a database is develop which shows the relationship between object-oriented measure an project measure. This relationship provides metrics that help in pro estimation.

Use Case-Oriented Metrics

- Use-cases describe user visible functions and features .
- They are defined early in software process and can be used as normalization measure before significant activities are initiate
- They are independent of programming language.
  No. of use cases directly proportional to
  size of application in LOC & no. of test cases that will be desig
  There is no standard size of a use case as they are created at different levels of abstraction
  For this reason, it is a suspect as a normalization measure.

WebApp Project Metrics

All WebApp projects is to deliver a combination of content and functionality to t user

- **The measures that can be collected are the following:.**

Number of static Web pages;These pages represent low relative com-plexity and require less effort to construct than dynamic pages. This measure provides an in of the overall size of the application and the effort required to develop it.

Number of dynamic Web pages; These pages represent higher relative complex require more effort to construct than static pages. This measure provides an ind the overall size of the application and the effort required to develop it .

Number of internal page links; This measure provides an indication of the degre architectural coupling within the WebApp. As the number of page links increase effort expended on navigational design and con-struction also increases.

WebApp Project Metrics

Number of persistent data objects; As the number of persistent data objec
a database or data file) grows, the complexity of the WebApp also grows a
effort to implement it increases proportionally.

Number of external systems interfaced; As the requirement for interfac-ng
system complexity and development effort also increase.

Number of static content objects; These objects represent low relative
com-plexity and generally require less effort to construct than dynamic pa

Number of dynamic content objects; These objects represent higher rela-t
complexity and require more effort to construct than static pages.

Number of executable functions; As the number of executable functions (
script or applet) increases, modeling and construction effort also increase.

Metrics for software quality

- Measuring quality
- Defect Removal Efficiency

Integrating metrics within the software process

- Arguments For Software quality
- Establishing a metrics Baseline
- Metrics collection, computation and evaluation

Metrics for software quality

- The goal of software engineering is to produce a high-quality system, application, or product within a time frame that satisfies a market need.
- To achieve this, you must apply effective methods coupled with modern tools within the context of a software process.
- The quality of a system, application, or product is only as good as
  - i) The requirements that describe the problem,
  - ii) The design that models the solution,
  - iii) The code that leads to an executable program, and
  - iv) The tests that exercise the software to uncover errors.
- You can use measurement to assess the quality based on above four parts.
- To accomplish this real-time assessment, you apply product metrics to evaluate the quality of software engineering work products in objective, rather than subjective ways.

- A project manager must also evaluate quality as the project progresses. Metrics collected by individual software engineers are combined to prov project level results.

- Although many quality measures can be collected, the primary thrust at project level is to measure errors and defects.

- Error data can also be used to compute the defect removal efficiency (D for each process framework activity.

Measuring quality

- Although there are many measures of software quality, correctness, maintainability , int usability provide useful indicators for the project team.

1.**CORRECTNESS:** Correctness is the degree to which the software performs its required fun

- It is measured in terms of defects per KLOC.
- Defects are those problems reported by a user of the program after the program has be general use.
- For quality assessment purposes, defects are counted over a standard period of time, ty

2.Maintainability: Maintainability is all about to correct an error if encountered, adapt any c environment, or enhancement of product if the customer, desires a change in requirements

- There is no way to measure maintainability directly.
- It can be measured by a simple time oriented metric ie, mean -time –to-change (MTTC) the project as follows :-
  a) the time it takes to analyze the change request,
  b) design an appropriate modification,
  c) implement the change,
  d) test it, and
  e) distribute the change to all users.

3.Integrity: This attribute measures a system's ability to withstand attacks
security.

- Attacks can be made on all three components of software : programs, da
  and documentation.
- It can be measured in terms of threat and security.

a) Threat is the probability that an attack of a specific type will occur withi
given time.

b) Security is the probability that the attack of a specific type will be repel

The integrity of a system can be defined as:

Integrity=€[1-(threat x(1-security))]

For example, if threat probability is 0.25 and security probability is 0.95 th
integrity of the system is 0.99.

4.Usability: If a program is not easy to use, it is often doomed to failure, ev
the functions that if performs are valuable.

- Usability is an attempt to qualify ease of use.

Defect removal efficiency

- A quality metric that provides benefit at both the project and process level is defect re efficiency (DRE). When considered for a project as a whole, DRE is defined in the followi

$$DRE=E/(E+D)$$

- Where E is the number of errors found before delivery of the software to the end user. D is the number of defects found after delivery.

The ideal value for DRE is 1 .That is no defects are found in the software.

- Realistically D Will be greater than 0,but the value of DRE can still approach 1 as E incre given value of D.
- As E increases, it is likely that the final value of D will decrease (errors are filtered out be become defects).
- DRE also assess a team ability to find errors before they are passed to the next framewo
- For example, requirements analysis produces a requirement model that can be reviewe correct errors.
- Those errors that are not found during the review of the requirements model are passe design. When used in this context, we redefine DRE as

$$DRE_i=E_i/(E_i+E_{i+1})$$

- ✔ Where E_i is the number of errors found during softw engineering action i

  E_{i+1} is the number of errors found during software engineering action i + 1 that are traceable to errors tha not discovered in software engineering action i.

- ✔ A quality objective for a software team (or an ind software engineer) is to achieve DREi that approach is, errors should be filtered out before they are pass the next activity or action.

INTEGRATING Metrics within the softwa process

- Many software developers do not collect measures.
- Without measurement it is impossible to tell whether a improving or not.
- Baseline metrics data should be collected from a large, representative sampling of past software projects.
- Getting this historic project data is very difficult, if the pr developers did not collect data in an on going manner.

Arguments for software quality

- Most Software developers measure, and most have little de
  begin.
- Establishing a successful company wide software metrics pr
  be a multi year effort.
- But if we do not measure, there is no real way of determinin
  whether we are improving.
- Measurement is used to establish a process baseline from w
  improvements can be assessed.
- Software metrics help people to develop better project esti
  produce higher –quality systems, and get products out the d
  time.

Establishing a metrics baseline

- By establishing a metrics baseline, benefits can be obtained at the soft
  process, product and project levels.
- The same metrics can serve many masters.
- The baseline consists of data collected from past projects.
- Baseline data must have the following attributes .
  -Data must be reasonably accurate (guesses should be avoided)
  -Data should be collected for as many projects as possible.
  -Measure must be consistent.
  - Past applications should be similar to the work that is to be estimat
- After data is collected and metrics are computed, the metrics should b
  evaluated and applied during estimation, technical work, project contr
  process improvement.

Integrating Metrics within the Software Process

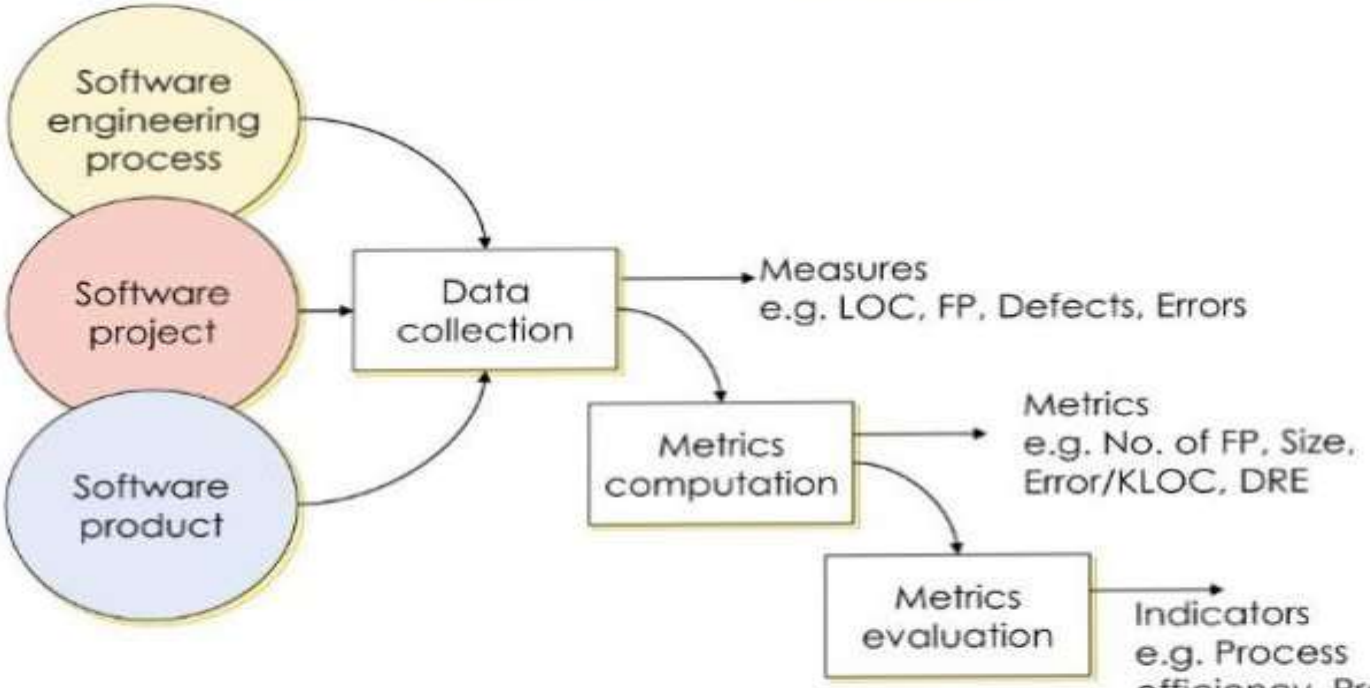


Fig: - Software metrics collection process

Metrics collection, computation and evaluation

- Ideally, data needed to establish a baseline has been collected in an ongoi
  Therefore, data collection requires a historical investigation of past projec
  reconstruct required data.
- Once measures have been collected metrics computation is possible. Depe
  breadth of measures collected, metrics can span a broad range of LOC or
  well as other quality- and project-oriented metrics.
- Finally, metrics must be evaluated and applied during estimation, technica
  project control, and process improvement. Metrics evaluation focuses on
  underlying reasons for the results obtained and produces a set of indicator
  the project or process.

SOFTWARE PROJECT ESTIMATION & DECOMPOSITION TECHNIQUES

Content :

- Software project estimation.
- Decomposition techniques.
- Software sizing
- Problem based estimation.
  - An example of LOC based estimation.
  - An example of FP based estimation.
- Process based estimation.
  - An example of process based estimation.
- Estimation with use cases.
- Reconciling estimates.

SOFTWARE PROJECT ESTIMATION

- Software is the most expensive elements of virtually all computer-based system.
- For complex, custom system, a large cost estimation error car make difference between profit and loss, cost over run can be disastrous.
- Software cost and effort estimation will never be an exact scie Because to many variable factors can effected the ultimate co the software or effort applied to develop it.

SOFTWARE PROJECT ESTIMATION

Variable project factors:

1.Human factors

2.Techinical variable factors

3.Environmental factor

4. Political factors

OPTIONS FOR RELIABLE ESTIMATION

1. Delay estimation until late in the project (obviously, we can achieve 100 percent accurate estimates after the project is complete!).

2. Base estimates on similar projects that have already been completed.

3. Use relatively simple decomposition techniques to generate project cost and effort estimates.

4. Use one or more empirical models for software cost and effo estimation

DECOMPOSITION TECHNIQUES:

- Decomposition techniques works on “Divide and Conquer” approach in software project estimation. By decomposition a projec is divided into different components and related software engineering activities. Cost and effort estimation can be performed step by step on each component.

  Software cost estimation is a form to solve the problems. Most of th times problem to be solved is too complex to be solve in a single st

DECOMPOSITION TECHNIQUES:

Than the problem is decomposed in to number of components in order to achieve an accurate cost estimate.

There are two approaches in decomposition technique:

1. Problem based estimation
2. Process based estimation

SOFTWARE SIZING

Accuracy of a softeware project estimation is predicted on a number of things:

I. (1) the degree to which you have properly estimated the size the product to be built.

2. the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of re software metrics frompast projects);

SOFTWARE SIZING:

- the degree to which the project plan reflects the abilities of the software team
- the stability of product requirements and the environment that supports the software engineering effort.
- There are four different sizing approaches:
- 1.Fuzzy logic sizing
- 2.Function poin sizing
- 3.Statement component sizing
- 4.change sizing

SOFTWARE SIZING

the result of each pricing approaches must be combined statistical e to create expected value estimate this accomplished by determining the following values

Combine the value size using the following equation to determine the expected value estimation

$$S=[sopt+(4\times sm)+Spess]/6$$

PROBLEM BASED ESTIMATION

- LOC and FP data are used into two ways during software Estimation and estimation.
- As an Estimation variable to size each element of software.
- LOC and FP estimations are different estimation technique both have number of characteristics in common.
- LOC or FP is then estimation for each Function.

PROBLEM BASED ESTIMATION

1) Start with a bounded statement of scope.
2) Decompose the software into problem functions that can each be estimated individually.
3) Compute an LOC or FP value for each function.
4) Derive cost or effort estimates by applying the LOC or FP values t baseline productivity metrics(e.g., LOC/person-month or FP/person-m
5) Combine function estimates to produce an overall estimate for the project.

PROBLEM BASED ESTIMATION

For both approaches, the planner uses lessons learned to estim
an optimistic, most likely, and pessimistic size value for each fu
or count (for each information domain value) • Then the expecte
value S is computed as follows :

$$S = [(Sopt + 4Sm + Spess)]/6$$

• Historical LOC or FP data is then compared to S in order to cross-check it.

EXAMPLE OF LOC-BASED ESTIMATION

As an example of LOC & FP problem-based estimation techniques, we consider a software package to be developed for a computer-aided (CAD) design application for mechanical components.

Function	Es
User interface and control facilities (UICF)	
Two-dimensional geometric analysis (2DGA)	
Three-dimensional geometric analysis (3DGA)	
Database management (DBM)	
Computer graphics display facilities (CGDF)	
Peripheral control function (PCF)	
Design analysis modules (DAM)	
Estimated lines of code	

AN EXAMPLE OF LOC-BASED ESTIMATION

=> Total Effort = Total LOC/Productivity = 33200/620=53.54 ≈ 54 person-months.

=> 6 developers Effort = Total Effort/6 = 54/6 = 9 months

=> Total Cost = Total Effort * Labor Rate = 54 * 800 ≈ $43,200 +W

=> Cost per LOC = LaborRate/Productivity=800/620=$1.29 ≈ $1.3

=> Total Cost = Total LOC * Cost per LOC = 33,200* 1.3=$43160 $431,200

=> Total Effort = Total Cost / Labor Rate = 43,200/800 = 54 person-months.

EXAMPLE OF FP- BASED ESTIMATION

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Factor
Backup and recovery
Data communications
Distributed processing
Performance critical
Existing operating environment
Online data entry
Input transaction over multiple screens
Master files updated online
Information domain values complex
Internal processing complex
Code designed for reuse
Conversion/installation in design
Multiple installations
Application designed for change
Value adjustment factor

FPestimated = count total * [0.65 +(0.01*ΣFi)]

FPestimated = 320* [0.65 +(0.01*52] = 375

EXAMPLE OF FP- BASED ESTIMATION

- The organizational average productivity for systems o
  type is 6.5 FP/pm.Based on a burdened labor rate of
  $8,000 per month, the cost per FP is approximately
  $1,230. Based on the FP estimate and the historical
  productivity data, the total estimated project cost is
  $461,000 and the estimated effort is 58 person-month

PROCESS BASED ESTIMATION

1) Identify the set of functions that the software needs to perform
obtained from the project scope.
2) Identify the series of framework activities that need to be
performed for each function.
3) Estimate the effort (in person months) that will be required to
accomplish each software process for each function.

4)Compare the resulting values to those obtained by way of the
and FP estimates .
If both sets of estimates agree, then your numbers are highly re
.

AN EXAMPLE OF PROCESSPROCESS-BASED ESTIMATION

FIGURE 33.4

Process-based estimation table

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Based on an average bu
rate of $8,000 per mont
estimated project cost is
and the estimated effort
person-months.

AN EXAMPLE OF PROCESS-BASED ESTIMATION

Once these values have been determined, the final UCP value is computed in the following manner:

UCP =(UUCW +UAW) * TCF * ECF

here, UCP= use case points.

UUCW= unadjustment use case weight

UAW= unadjustment actor weight

TCF= technical complex ityfactor

ECF= environmental complexity factor

AN ESTIMATION WITH USE CASES:

- The CAD software introduced in is composed of three subsyster groups:
  user interface subsystem (includes UICF), engineering subsyste group (includes the 2DGA, 3DGA, and DAM subsystems), and infrastructure subsystem group (includes CGDF and PCF subsystems). Sixteen complex use cases describe the user interface subsystem. The engineering subsystem group is descr by
  14 average use cases and 8 simple use cases. And the infrastructure subsystem is described with 10 simple use cases.

RECONCILING ESTIMATION

The results gathered from the various estimation techniques must be reconciled to produce a sing estimate of effort, project duration, and cost.

• If widely divergent estimates occur, investigate following causes.

– The scope of the project is not adequately understood or has been misinterpreted by the planner.

RECONCILING ESTIMATION

– Productivity data used for problem-based estimation techniques is inappropriate for the application, obsolete (i.e., outdated for the current organization), or has been misapplied

• The planner must determine the cause of divergence and then reconcile the estimates

Chapter - 3
Project Scheduling

Project Scheduling:-

- What is project scheduling?
- Why is it important?
- Why are software project late?
- What should we do when deadline occurs
- The basic principles
- The relation between people and effort
- The PNR curve
- Project effort distribution

WHAT IS PROJECT SCHEDULING ?

Project schedule communicates what work needs
be done

Resources who will perform the
work

Timeframes in which that work ne
to be performed

PROJECT SCHEDULING

- One of the most difficult jobs for a project manager.
- Begins with process decomposition. It involves separat total work involved in a project into separate activities and judging the time required to complete these activities.
- Managers estimate time and resources required to complete activities and organize them into coherent sequence.

- When estimating schedulers,you should NOT assume that ever
  project will be problem-free.
  -- People may fall ill or may leave
  -- Hardware may break down
  -- Essential software/hardware may be delivered late
- If the project is new and technically advanced, then certain part
  of it may turn out to be more difficult and take longer than
  originally anticipated.

Project scheduling:WHY IS IT IMPORTANT?

- Today's software systems are large, sophisticated and
  complex

  > Many software engineering tasks occur in parallel

  > Result of work performed during one task may have a profound effect
  work conducted in another task

  > Task interdependencies are very difficult to understand without a sche

  > It is virtually impossible to assess progress on a large software project
a
  detailed schedule

WHY ARE SOFTWARE PROJECTS LATE?

- *There are many reasons why software is delivered late …*

- An unrealistic deadline established by some one outside the software development group .
- Changing customer requirements that are not reflected in schedule changes .
- Risk not considered at the beginning of the project .
- Miscommunication among project staff resulting in re-work and delay.
- Honest underestimation of the job .

- Technical difficulties that could not have been foreseen in advance
- Human difficulties that could not have been foreseen in advan
- Project Management fails to track progress—
    -- Unaware of the project is in behind the schedule and in trouble
    -- No correction actions taken

WHAT SHOULD WE DO WHEN MANAGEMENT DEMANDS THAT WE MAKE A DEADLINE THAT IS IMPOSSIBLE?

- Perform a detailed estimate of effort and time using historica data
- Using an incremental process model, develop a strategy to deliver critical functionality by the deadline– document the plan
- Meet with the customer and explain why the project is late
- Offer the incremental development strategy as an alternative

SOFTWARE PROJECT SCHEDULING: THE BASIC PRINCIPLES

1. Compartmentalizing : Compartmentalize the project into a number of manageable

   activities, actions and tasks

2. Interdependency : Indicate tasks interrelationship

3. Time allocation : Each task must be allocated some no. of work units with

   startdate and completion date

4. Effort validation : Be sure resources are available

5. Defined responsibilities : Every task should be assigned to a specific team r

6. Defined outcomes : Each task must have an output (e.x
work product)

7. Defined milestones : Tasks should be associated with a
project milestone (A milestone Is accomplished when or
or more work products has to be reviewed and approved

THE RELATION BETWEEN PEOPLE AND EFFORT

■”If we fall behind schedule, we can always add more programmers and cat up later in the project!”

Why does this not work?

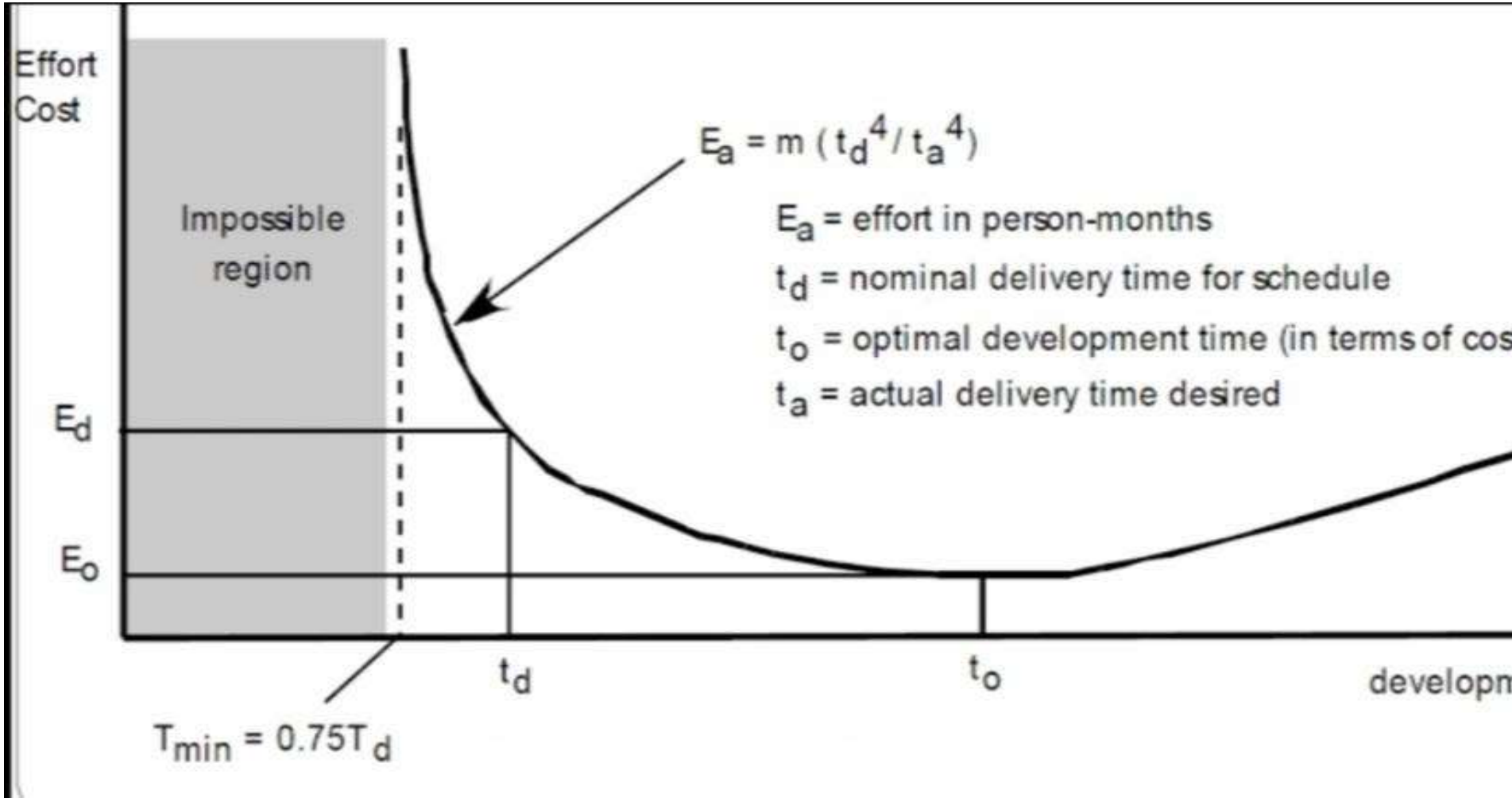
-- Added/ new people must learn the system and learning takes time

-- Teaching takes time away from productive work

-- The more people increase the number of communication paths and complexity ofcommunication increases

- Over the years, empirical data and theoretical analysis have demonstrated that project schedules are elastic
  -- It is possible to compress a desired project completion date
  adding additional resources to some extent
  -- It Is also possible to extend a completion date by reducing th
  number of resources
- The Putnam–Norden–Rayleigh (PNR) curve provides an indicati of the relationship between effort applied and delivery time for software project

RELATION BETWEEN EFFORT AND DELIVERY TIME
THE PNR CURVE



PROJECT EFFORT DISTRIBUTION

■ *How should effort be distributed across the software workflow?*

- A recommended distribution of effort across the software process is oft referred to as the ***40-20-40*** rule
  - ☆ 40% of effort ”front - end"(analysis and design)
  - ☆ 20% of effort “Coding “
  - ☆ 40% of effort “back – end” (testing)

SOFTWARE RISK, RISK MANAGEMENT & THE RMMM PLAN

- What is a RISK ?
- Definitions of risks
- Types of software risks
- Negative impact of risk
- Risk Management
- How to manage the risks
- Reactive versus proactive risk strategies
- The RMMM Plan

What is a risk ?

- A risk is a potential problem – it might hap
  and it might not, this is uncertainty.
- We don't know whether a particular even
  occur or no but if it does has a negative im
  on a project.

Definitions of risks :

- Risk is the probability of suffering loss.
- Risk provides an opportunity to develop the pr
  better.
- There Is a difference between a problem and r
- Problem is some event which has already occu
  but risk is something that is unpredictable.

TYPES OF SOFTWARE RISKS :

There are different kinds of risks. Some of them are a follows:

- Project Risk
- Technical Risk
- Business Risk
- Known Risk
- Predictable Risk
- Unpredictable Risk

NEGATIVE IMPACT OF RISK :

- Diminished quality of product
- Increased cost
- Delayed completion
- Project failure

RISK MANAGEMENT :

- Risk Management is a methodology that helps managers make best use of their available resources.
- By using the various principles we can manage the risks.
- The project should be managed in such a way that the risks don’t affect the project in a big way.

RISK MANAGEMENT :

- Risk management consists of 6 different states.
- Identify
- Analyze
- Plan
- Track
- Control
- Communication

HOW TO MANAGE THE RISKS :

- Determine risk sources and categories.
- Determine risk parameters.
- Establish a risk management strategy.
- Identify risks
- Evaluate and prioritize the risks.
- Develop and implement risk mitigation plans.

REACTIVE VERSUS PROACTIVE RISK STRATEGIES :

- Reactive Risks strategies have been called the “Indiana Jones Sc of risk Management”.
- A proactive strategy is a more intelligent strategy for Risk Management.
- A proactive strategy begins long before technical work is initiat
- In this proactive strategy potential risks are identified their pro and impact are assessed and they are ranked by importance.
- The primary objective of proactive strategy is to avoid risk.

THE RMMM PLAN :

- The meaning of RMMM Plan is risk mitigation, monitoring and man plan.
- An effective strategy must consider 3 issues risk avoidance, risk monito risk management and contingency planning.
- A Risk management strategy can be included in the software project pla risk management steps can be organized into a separate risk m monitoring and management plan.
- The RMMM plan documents all work performed as part of risk analys used by the project manager as part of the overall project plan.
- Some software teams do not develop a formal RMMM document rath risk is documented individually using a risk information sheet.

MANAGING RISKS :

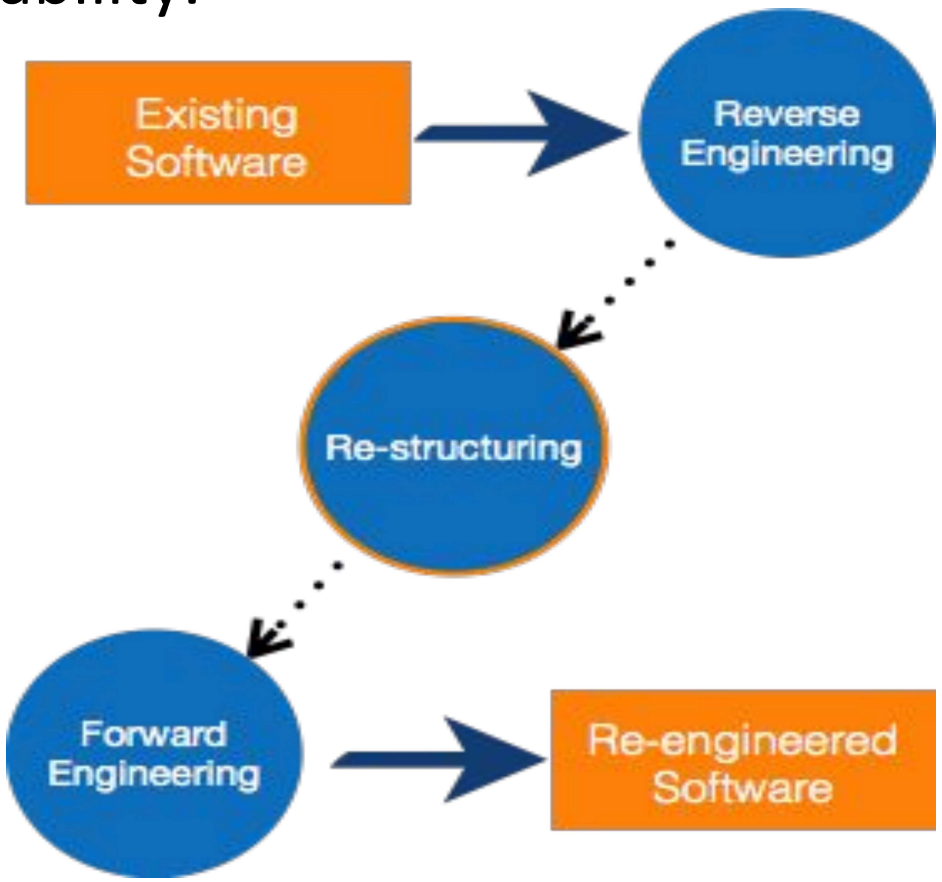
- To manage the risks we need to establish a strong bond bet
  customers and the team members.
- A strong base about risk management would help a great d
  tackling the risks.
- Software metrics and tools can be developed to manage th
- Risk necessarily need not be negative and it can be viewed
  opportunity to develop our projects in a better way.

SOFTWARE REENGINEERING, reverse engineering and forward engineering

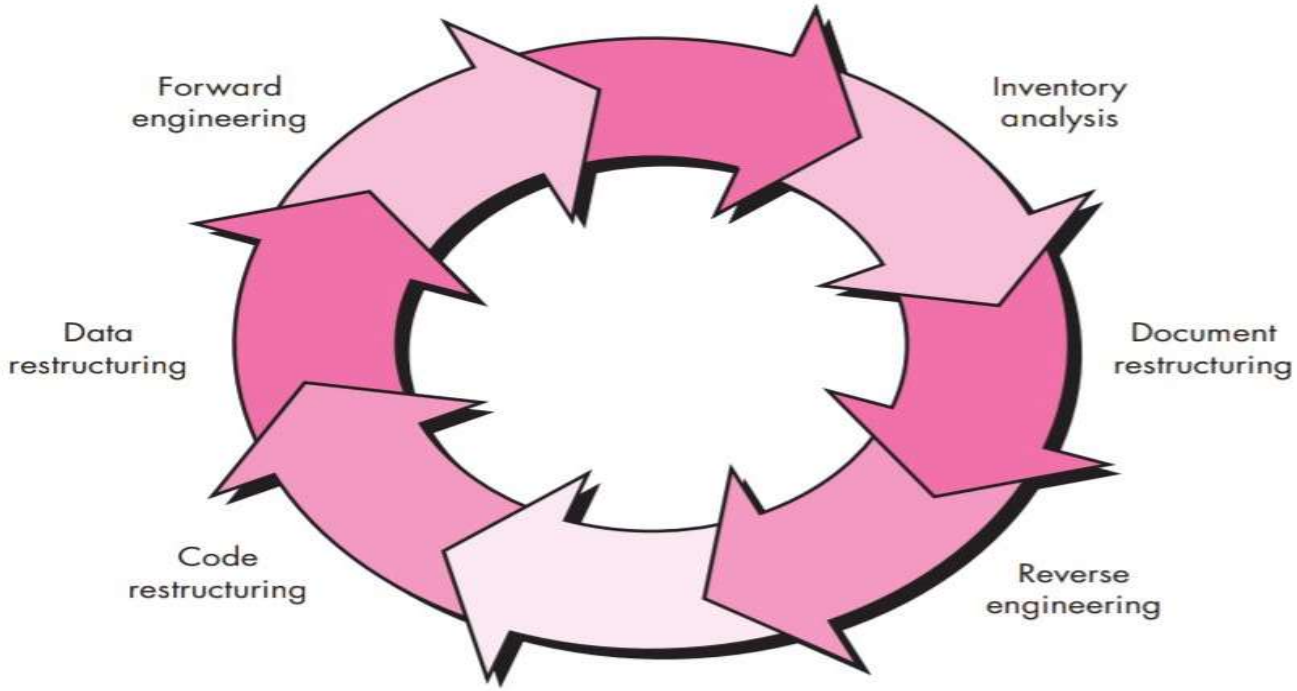
- **Software Reengineering**
- **Reverse Engineering**
- **Forward Engineering**

Software Reengineering

- It is an activity that improves ones understanding of software or p
  or improves the software itself usually for increased maintainabili
  reusability evolvability.



- Reengineering takes time, it costs significant amounts of money.
- Reengineering is a rebuilding activity



Software re engineering process model

Software Re Engineering Activities

1. Inventory Analysis:

Every software organization should have an inventory of all the applicatio

- Inventory can be nothing more than a spreadsheet model containing infor that provides a detailed description(e.g.. size , age, business criticality) of active application.

2. Document restructuring:

Documentation of a system either explains how it operates or how to use

- Documentation must be updated.
- It may not be necessary to fully document an application.

Software Reengineering Activities

3. Reverse Engineering:

Reverse engineering is a process of design recovery. Reverse enginee tools extract data, architectural and procedural design information from existing program.

4. Code Reconstructing:

- To accomplish code reconstructing, the source code is analyzed using reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed.
- The resultant restructured code is reviewed and tested to ensure that n anomalies have been introduced.

SOFTWARE REENGINEERING ACTIVITIES

5. Data Restructuring:

- Data restructuring begins with a reverse engineering activity.
- Current data architecture is dissected, and the necessary data models
- Data objects and attributes are identified, and existing data structure a

for quality.

6. Forward Engineering:

Forward Engineering also called as renovation or reclamation not only design information from existing software but uses this information to a reconstitute the existing system in an effort to improve its overall quali

REVERSE ENGINEERING

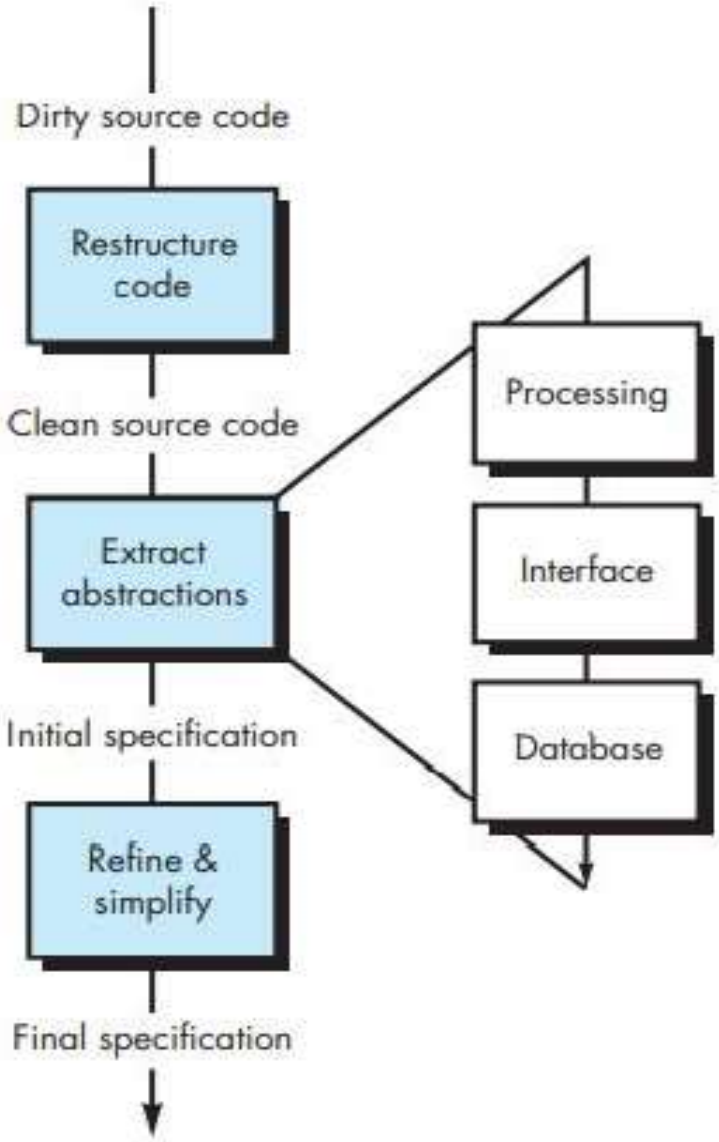
REVERSE ENGINEERING: Reverse engineering is the process by which a man made object is Deconstructed to reveal its designs , architecture , or to extract knowledge from the object, and called as Back engineering

- Reverse engineering can extract design information from source code but the abstraction lev completeness of the documentation , the degree to which tools and a human analyst work to directionality of the process are highly variable.
- **Abstraction level:** The abstraction level of a reverse engineering process and the tools used t refers to sophistication of design information that can be extracted from source code .
- The reverse engineering process should be capable of deriving
  - Procedural design representations
  - Program and data structure information
  - Object models
  - Data and/control flow models
  - Entity relationship models
- As the abstraction level increases , you are provided with information that will allow easier understanding of the program

REVERSE ENGINEERING

- The completeness of a reverse engineering process refers to the level of de provided at an abstraction .
- Interactivity refers to the degree to which the human is integrated with aut tools to create an effective reverse engineering process.
- If the directionality of the reverse engineering process is one way all inform extracted from the source code is provided to the software engineer who ca it during any maintenance activity.
- If directionality is two way the information is fed to a reengineering tool tha to restructure or regenerate the old program

REVERSE ENGINEERING PROCESS



REVERSE ENGINEERING TO UNDERSTAND DATA

- First reengineering task
- Occurs at different level of abstraction
- At the program level , internal program data structure must often be reverse enginee of an overall effect
- At the system level , global data structures(e.g.. Files, databases) are often reenginee accommodate new database management

Internal data structures:

- Reverse engineering techniques for internal program data focus on the definition of objects by examining the program code with the intent of grouping related program

Database structure:

- Regardless of its logical organization and physical structure a database allows the defi data objects and supports some method for establishing relationships among the objec
- Therefore reengineering one database schema into another requires an understanding objects and their relationships

Reverse engineering to understand processing

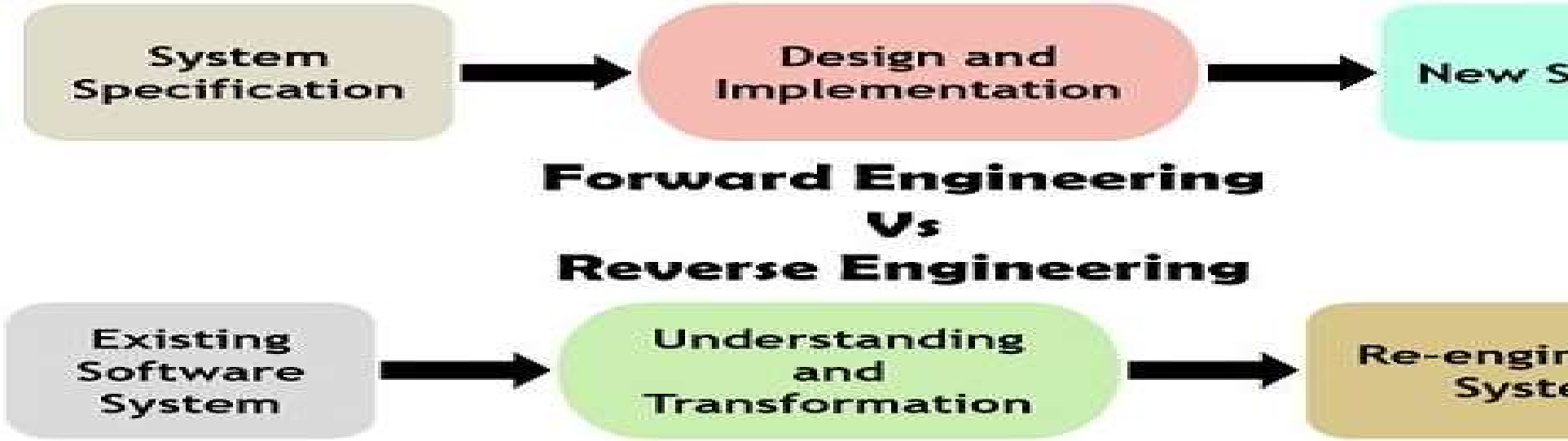
- Reverse engineering to understand processing begins with an attempt to understan and then extract procedural abstractions represented by the source code.
- To understand procedural abstractions, the code is analyzed at varying levels of abstraction: system, program, component, pattern, and statement
- Each component performs some subfunction and represents a defined procedural abstraction.
- In almost every component, a section of code prepares data for processing (within th module), a different section of code does the processing, and another section of code prepares the result of processing for export from the component
- The output of this process is then passed to restructuring and forward engineering to to complete the reengineering process.

Reverse Engineering User Interface

- The redevelopment of user interfaces has become one of the most common type reengineering activity.
- But before a user interface can be rebuilt, reverse engineering should occur.
- To fully understand an existing user interface, the structure and behavior of the i be specified. Merlo and his colleagues [Mer93] suggest three basic questions that answered as reverse engineering of the UI
- What are the basic actions (e.g., keystrokes and mouse clicks) that the interface r
- What is a compact description of the behavioral response of the system to these
- What is meant by a “replacement,” or more precisely, what concept of equivalen interfaces is relevant here?

FORWARD ENGINERRING

- Forward engineering is a method or making an application with the help of the given req
  Forward engineering is also known as Renovation and Reclamation.
- Forward Engineering process applies software engineering principles , concepts and met
  an existing application.
- In most cases Forward Engineering does not simply create a modern equivalent of an old
- Rather , new user and technology requirements are integrated into the reengineering eff
- This redeveloped program extends the capabilities of the older application.



FORWARD ENGINEERING FOR CLIENT SERVER ARCHTECTURE

- Many mainframe applications have been reengineered to accommodate client ser
  architecture
- A client server architectures is Centralized computing resources (including softwar
  distributed among many client platforms.
- A typically mainframe application that reengineered into a client – server architect
  the following features
- Application functionality migrates to each client computer,
- New GUI interfaces are implemented at the client sites,
- Database functions are allocated to the server,
- Specialized functionality (e.g., compute-intensive analysis) may remain at the serv
- New communications, security, archiving, and control requirements must be esta
  both the client and server sites.

FORWARD ENGINEERING FOR OBJECT ORIENTED ARCHITECTURE

- Older applications must be reengineered so that they can be easily integrated into large, object-oriented systems.
- First, the existing software is reverse engineered so that appropriate data, functional, and models can be created.
- If the reengineered system extends the functionality or behavior of the original applicatio are created.
- Class hierarchies, object-relationship models, object-behavior models, and subsystems are object-oriented design commences.

Project-oriented forward engineering progresses from analysis to design, a CBSE (Component software engineering) process model can be invoked.

- If the existing application resides within a domain that is already populated by many object- applications, it is likely that a robust component library exists and can be used during forwa engineering.

Difference between reverse engineering and forward engineering

Forward engineering	Reverse engineering
• In forward engineering, the application are developed with the given requirements. • Forward Engineering is high proficiency skill. • Forward Engineering takes more time to develop an application. • The nature of forward engineering is Prescriptive. • In forward engineering, production is started with given requirements • The example of forward engineering are construction of electronic kit, construction of DC MOTOR etc.	• **In reverse engineering or backward engineering, the information are collected from the given application** • **Reverse Engineering or backward engineering is low proficiency skill. ior** • **While Reverse Engineering or backwa engineering takes less time to develop an application.** • **The nature of reverse engineering or backward engineering is Adaptive.** • **. In reverse engineering, production is started by taking existing product.** • **The example of backward engineering are research on Instruments etc.**

Thank you