JAIN DEEMED-TO-BE UNIVERSITY SCHOOL OF COMPUTER SCIENCE AND IT

SOFTWARE ENGINEERING

UNIT IV
SOFTWARE IMPLEMENTATION AND MAINTENANCE

Prof. Rahul Pawar
Assistant Professor,
School of CS&IT,
MCA Department,
Knowledge Campus Jayanagar

Structured Coding Techniques

- Structured programming (sometimes known as modular programming) is a subset of procedural programming that enforces a logical structure on the program being written to make it more efficient and easier to understand and modify.
- Structured programming frequently employs a top-down design model, in which developers map out the overall program structure into separate subsections.
- A defined function or set of similar functions is coded in a separate module or submodule, which means that code can be loaded into memory more efficiently and that modules can be reused in other programs.

- After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Coding Styles-Standards and Guidelines

- Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards.
- Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards rigorously.
- The purpose of requiring all engineers of an organization to adhere to a standard style of coding is the following:
  - A coding standard gives a uniform appearance to the codes written by different engineers.
  - It enhances code understanding.
  - It encourages good programming practices.

Coding standards and guidelines

- **Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot.
- **Contents of the headers preceding codes for different modules:** The information contained in the headers of different modules should be standard for an organization.
- The following are some standard header data:
  - Name of the module.
  - Date on which the module was created.
  - Author's name.
  - Modification history.
  - Synopsis of the module.
  - Different functions supported, along with their input/output parameters.
  - Global variables accessed/modified by the module.

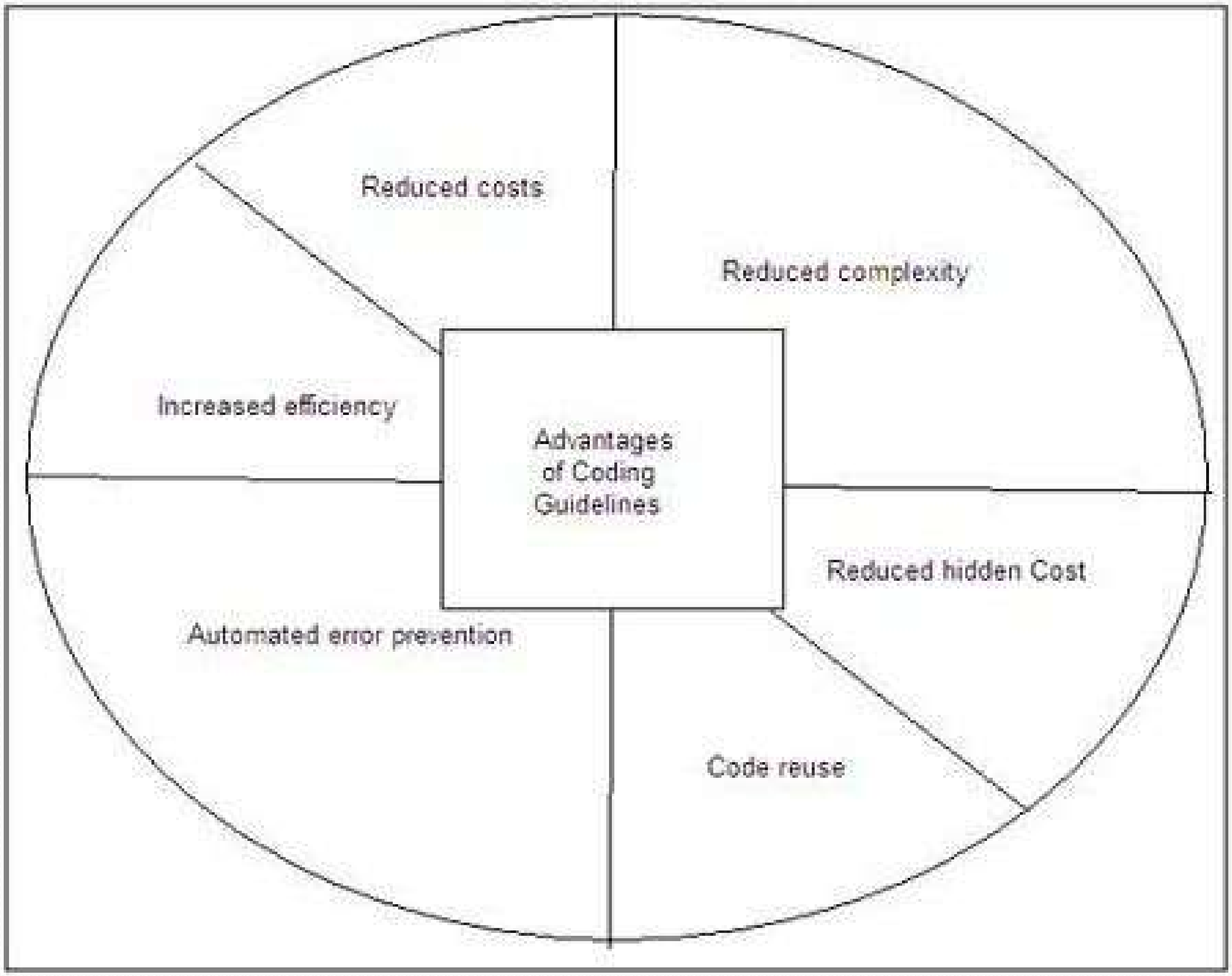
- **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
- **Error return conventions and exception handling mechanisms:** The way error conditions are reported by different functions in a program are handled should be standard within an organization. For example, different functions while encountering an error condition should either return a 0 or 1 consistently.

Coding Guidelines

- All the codes should be properly commented before being submitted to the review team.
- All curly braces should start from a new line.
- All class names should start with the abbreviation of each group. For example, AA and CM can be used instead of academic administration and course management, respectively.
- Errors should be mentioned in the following format: [error code]: [explanation]. For example, 0102: null pointer exception, where 0102 indicates the error code and null pointer exception is the name of the error.
- Every 'if statement should be followed by a curly braces even if there exists only a single statement.
- Every file should contain information about the author of the file, modification date, and version information.

- Similarly, some of the commonly used coding guidelines in a database (organized collection of information that is systematically organized for easy access and analysis) are listed below.
- Table names should start with TBL. For example, TBL_STUDENT.
- If table names contain one word, field names should start with the first three characters of the name of the table. For example, STU_FIRSTNAME.
- Every table should have a primary key.
- Long data type (or database equivalent) should be used for the primary key.

Advantages of Coding Guidelines



Documentation Guidelines

- When various kinds of software products are developed then not only the executable files and the source code are developed but also various kinds of documents such as users' manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process.
- Different types of software documents can broadly be classified into the following:
  - Internal documentation
  - External documentation

- Internal documentation is the code comprehension features provided as part of the source code itself.
- Internal documentation is provided through appropriate module headers and comments embedded in the source code.
- Internal documentation is also provided through the useful variable names, module and function headers, code indentation, code structuring, use of enumerated types and constant identifiers, use of user-defined data types, etc.
- External documentation is provided through various types of supporting documents such as users' manual, software requirements specification document, design document, test documents, etc.
- A systematic software development style ensures that all these documents are produced in an orderly fashion.

A well-maintained documentation should involve the following documents:

- **Requirement documentation:** works as key tool for software designer, developer and the test team to carry out their respective tasks. It includes all the functional, non-functional and behavioral description of the intended software.

  Source of this document can be previously stored data about the software, already running software at the client's end, client's interview, questionnaires and research. Generally it is stored in the form of spreadsheet or word processing document with the high-end software management team.

- **Software Design documentation** - which are needed to build the software. It contains:

  (a) High-level software architecture,

  (b) Software design details,

  (c) Data flow diagrams,

  (d) Database design

  These documents work as repository for developers to implement the software.

- **Technical documentation** - These documentations are maintained by the developers and actual coders. These documents, as a whole, represent information about the code.

- While writing the code, the programmers also mention objective of the code, who wrote it, where will it be required, what it does and how it does, what other resources the code uses, etc.
- **User documentation** - explains how the software product should work and how it should be used to get the desired results.

These documentations may include, software installation procedures, how-to guides, user-guides, uninstallation method and special references to get more information like license updation etc.

MODERN PROGRAMMING LANGUAGE FEATURES

Type checking

- **Type checking** is the process of verifying and enforcing the constraints of types, and it can occur either at compile time (i.e. statically) or at runtime (i.e. dynamically).
- Type checking is all about ensuring that the program is **type-safe**, meaning that the possibility of type errors is kept to a minimum.
- A type error is an erroneous program behavior in which an operation occurs (or tries to occur) on a particular data type that it's not meant to occur on.
- This could be a situation where an operation is performed on an integer with the intent that it is a float, or even something such as adding a string and an integer together:

- Methods of type checking: **static type checking** and **dynamic type checking**.
- A language is **statically-typed** if the type of a variable is known at **compile time** instead of at runtime.
- Common examples of statically-typed languages include Ada, C, C++, C#, JADE, Java, Fortran, Haskell, ML, Pascal, and Scala.
- The big benefit of static type checking is that it allows many type errors to be caught early in the development cycle.
- **Static typing** usually results in compiled code that executes more quickly because when the compiler knows the exact data types that are in use, it can produce optimized machine code (i.e. faster and/or using less memory).

- **Dynamic type checking** is the process of verifying the type safety of a program at **runtime**.
- Common dynamically-typed languages include Groovy, JavaScript, Lisp, Lua, Objective-C, PHP, Prolog, Python, Ruby, Smalltalk and Tcl.
- When using these languages you need not specify or declare the type of variable instead compiler itself figures out what type a variable is when you first assign it a value.
  Now consider some statements in python:
  str1="Python"
  str2=10
  Here you need not declare the data type. The compiler itself will know which type the variable belongs to when you first assign it a value (str1 is of "String" data type and str2 is of type “int").

DATA ABSTRACTION

- Abstraction is a technique for arranging complexity of computer systems.
- An abstract data type is a specification of the behavior of a data type (often a class) without reference to its underlying implementation.
- Data abstraction is one of the fundamental principles of software engineering, since it facilitates the decomposition of a problem into independent chunks and allows the implementation of the data type to change without breaking the client code.

Typically includes:

1. Data structures: constants, types, and variables accessible to user
2. Declarations of functions and procedures accessible to user .

As an example, an algebraic specification of behavior of a stack might look like pop(push(S,x)) = S, if not empty(S) then push(pop(S), top(S)) = S

- Formal specification of ADTs uses universal algebras Data + Operations + Equations = Algebra.

Goals

- That one need not understand how the data type is implemented in order to use it.
- It gives more flexibility for changing the implementation, so long as it obeys the specification.
- The user can replace one implementation with another if it proves to be more effective.

Exception Handling

- Exception – an indication of a problem that occurs during a program’s execution.
- Exception handling – resolving exceptions that may occur so program can continue or terminate gracefully.
- Exception handling enables programmers to create programs that are more robust and fault-tolerant.
- `try` **block** – encloses code that might throw an exception and the code that should not execute if an exception occurs.
- Consists of keyword `try` followed by a block of code enclosed in curly braces.

`catch` **block** – catches (i.e., receives) and handles an exception, contains:

- Begins with keyword `catch`
- Exception parameter in parentheses – exception parameter identifies the exception type and enables catch block to interact with caught exception object
- Block of code in curly braces that executes when exception of proper type occurs

- **Matching `catch` block** – the type of the exception parameter matches the thrown exception type exactly or is a superclass of it
- **Uncaught exception** – an exception that occurs for which there are no matching `catch` blocks
  - Cause program to terminate if program has only one thread; Otherwise only current thread is terminated and there may be adverse effects to the rest of the program.

Concurrency Mechanism

- Collection of techniques and mechanisms that enable a computer program to perform several different tasks simultaneously, or apparently simultaneously.
- **Mechanisms:**
  - **Managing Threads of Control**
    - To support concurrency, a system must provide for multiple threads of control. The abstraction of a thread of control can be implemented in a number of ways by **hardware and software.**
      - **Multiprocessing** — multiple CPUs executing concurrently
      - **Multitasking** — the operating systems simulates concurrency on a single CPU by interleaving the execution of different tasks
      - **Application-based solutions** — the application software takes responsibility for switching between different branches of code at appropriate times.

Multithreading

- Many operating systems, particularly those used for real-time applications, offer a "lighter weight" alternative to processes, called **"threads"** or "lightweight threads."
- Threads are a way of achieving a slightly finer granularity of concurrency within a process. Each thread belongs to a single process, and all the threads in a process share the single memory space and **other resources controlled by that process.**
- Usually **each thread is assigned a procedure to execute**.

Multitasking

- When the operating system provides multitasking, a common unit of concurrency is the process. A process is an entity provided, supported and managed by the operating system whose sole purpose is to provide an environment in which to execute a program. The process provides a memory space for the exclusive use of its application program, a thread of execution for executing it, and perhaps some means for sending messages to and receiving them from other processes.

Each process has three possible states:

- **blocked** — waiting to receive some input or gain control of some resource;
- **ready** — waiting for the operating system to give it a turn to execute;
- **running** — actually using the CPU.
- Processes are also often assigned relative priorities. The operating system kernel determines which process to run at any given time based upon their states, their priorities, and some scheduling policy. Multitasking operating systems actually share a single thread of control among all of their processes.

Multiprocessing

- Of course, multiple processors offer the opportunity for truly concurrent execution. Most commonly, each task is permanently assigned to a process in a particular processor, but under some circumstances tasks can be dynamically assigned to the next available processor.
- Perhaps the most accessible way of doing this is by using a "symmetric multiprocessor."
- In such a hardware configuration, multiple CPUs can access memory through a common bus.

Software Maintenance

- Software maintenance is widely accepted part of SDLC now a days.
- It stands for all the modifications and updations done after the delivery of software product. There are number of **reasons, why modifications are required**, some of them are briefly mentioned below:
- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.

- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of maintenance

Types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

- **Adaptive Maintenance** - This includes modifications and updations applied to **keep the software product up-to date** and tuned to the ever changing world of technology and business environment.

Types of maintenance

- **Perfective Maintenance** - This includes modifications and updates done **in order to keep the software usable over long period of time**. It includes new features, new user requirements for refining the software and improve its reliability and performance.

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

Software Supportability

It is the capability of supporting a software system over its whole product life.

This implies satisfying any necessary needs or requirements, but also the provision of equipment, support infrastructure, additional software facilities, manpower, or any other resource required to maintain the software operational and capable of satisfying its function.

Software Supportability

- **What does Software Support encompass?**
  - **covers the whole software life-cycle** once it enters into service.
  - In particular, it covers the following key aspects associated to the software:
    - **Operation**
    - **Logistics Management**
    - **Modification**

- **Operation** covers all aspects associated to the actual use of the software, including the installation, loading (or unloading), configuration, error recovery and execution of the software.
- **Logistics Management** covers all aspects related to the handling of the software once a new baseline has been produced, until its delivery to the end user.
- **Modification** covers all aspects related to the evolution of the software due to the need of fixing bugs, or adding/changing functionality due to changing user needs.

Software Re-engineering

- Reorganising and modifying existing software systems to make them more maintainable
- Restructuring or rewriting part or all of a system **without changing its functionality**
- **Applicable** when some (but not all) subsystems of a larger system require frequent maintenance
- The reengineered system may also be restructured and should be redocumented .
- For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

When to re-engineer

- When system changes are mostly confined to part of the system then re-engineer that part
- When hardware or software support becomes obsolete
- When tools to support re-structuring are available.

Re-engineering advantages:

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software

Business Process Reengineering

- The search for, and the implementation of, radical change in business process to achieve breakthrough results
- **Business Processes**
  - “a set of logically related tasks performed to achieve a defined business outcome”

Examples of business processes include

- designing a new product
- purchasing services and supplies,
- hiring a new employee, and paying suppliers

Every business process has a defined customer—a person or group that receives the outcome (idea, a report, a design, a service, a product)

Business Process Reengineering

- **Business Hierarchy**

  The business -> business systems -> business processes

  -> business sub-processes

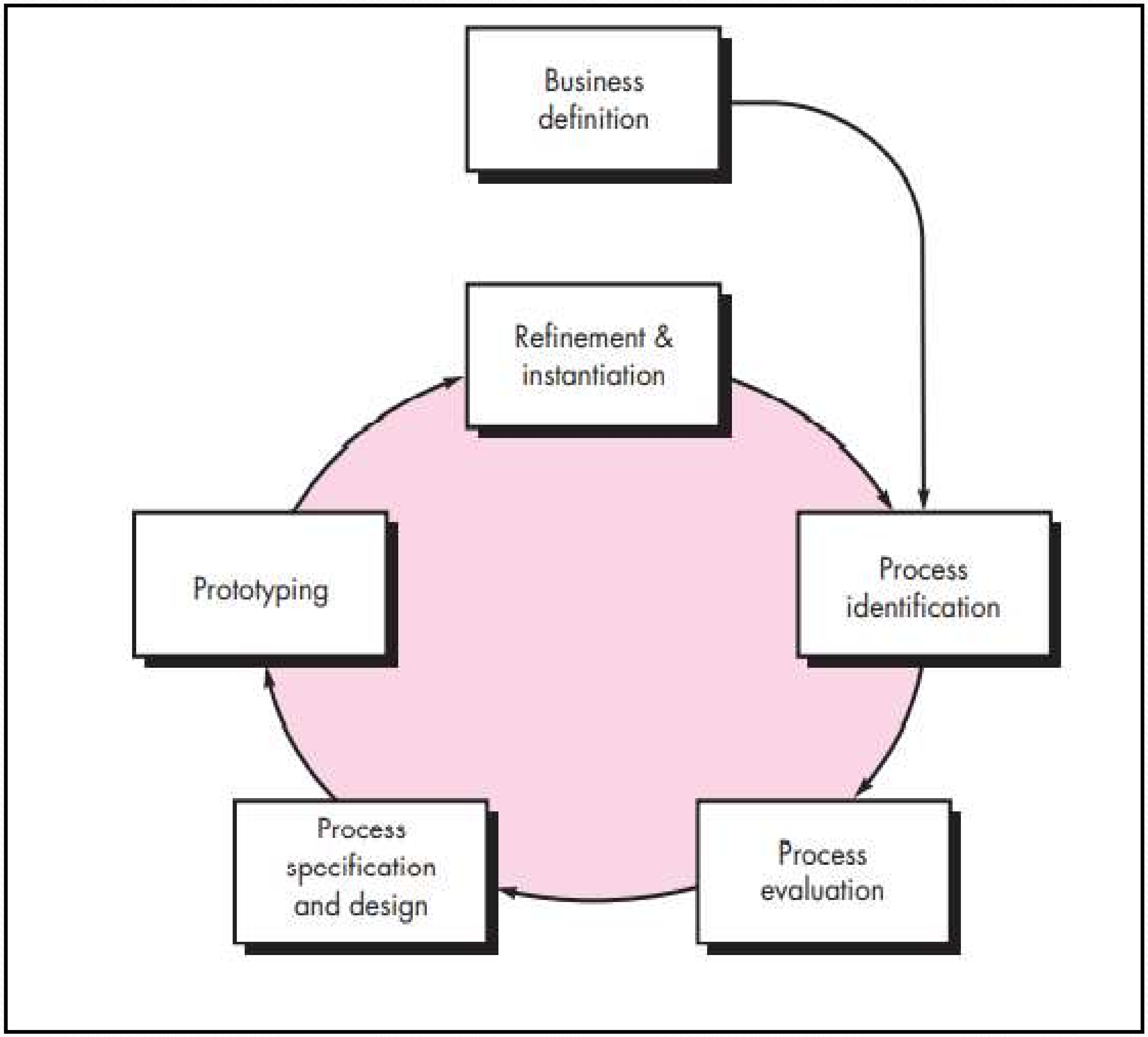
Each business system (also called business function)

- is composed of one or more business processes, and each business process is defined by a set of sub-processes.

- BPR can be applied at any level of the hierarchy
- BPR Model
  - business process <u>reengineering is iterative</u>
  - A model for business process reengineering <u>defines six activities</u>

Business Process Reengineering

- BPR MODEL



Business Process Reengineering

BPR MODEL

- **Business definition:** Business goals are identified within the context of four key drivers

  - cost reduction, time reduction, quality improvement, and personnel development and empowerment.

  Goals may be defined at the business level or for a specific component of the business.

- **Process identification:** Processes that are critical to achieving the goals defined in the business definition are identified.

  They may then be ranked by importance, by need for change, or in any other way that is appropriate

Business Process Reengineering

- **Process evaluation:**

  Existing process is thoroughly analyzed and measured.

  - Process tasks are identified

  -consumed costs and time are noted and

  - quality/performance problems are isolated.

- **Process specification and design**:

  - use cases are prepared for each process that is to be redesigned

  - a new set of tasks are designed for the process.

- **Prototyping:** A redesigned business process must be prototyped before it is fully integrated into the business.

  This activity "tests" the process so that refinements can be made

Business Process Reengineering

- **Refinement and instantiation:**

  Based on feedback from the prototype, the business process is refined and then instantiated within a business system.

Software Reengineering

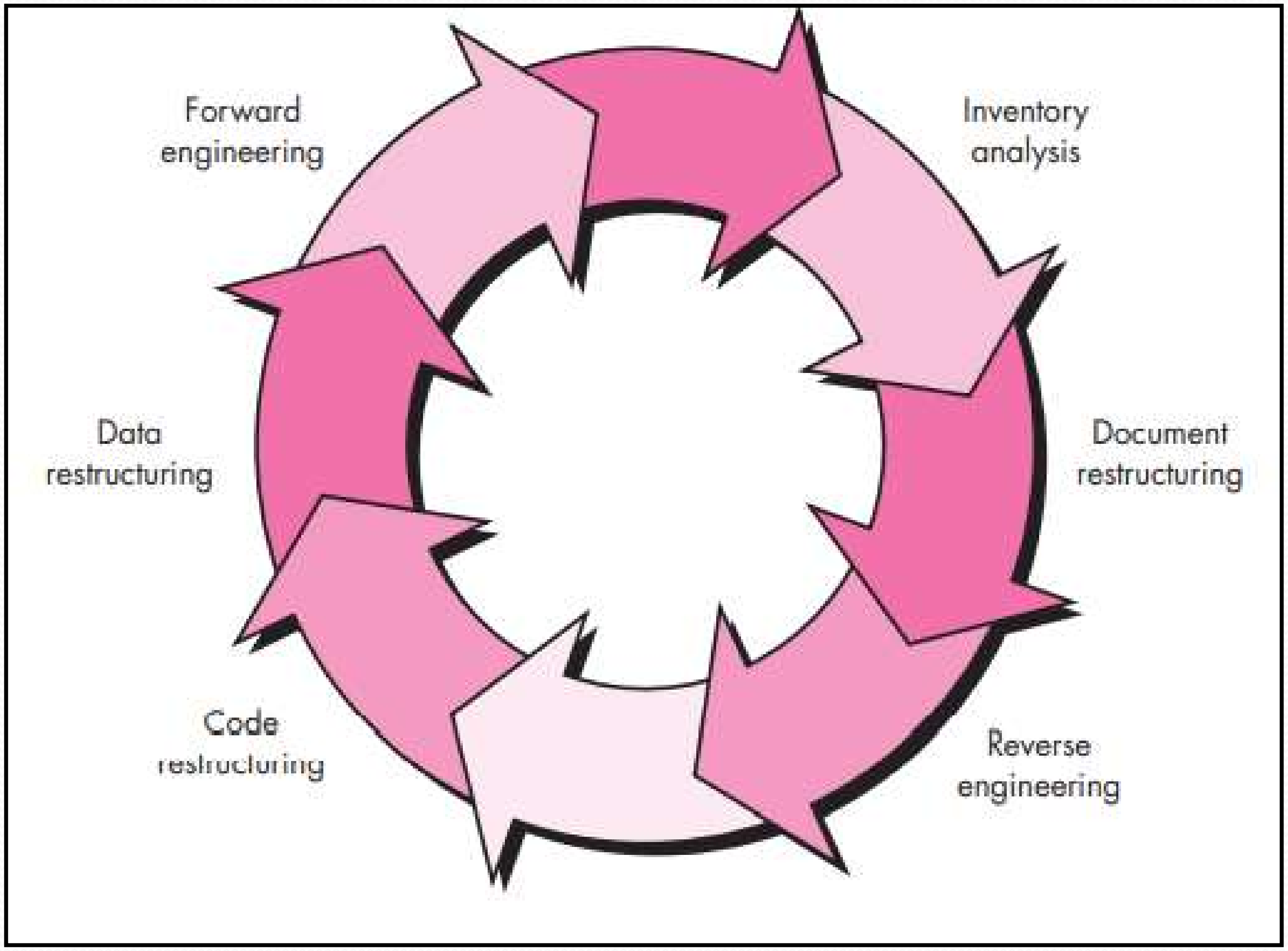
- Software Reengineering has been spawned by software maintenance problems
- Reengineering of information systems is an activity that will absorb information technology resources for <u>many years</u>
- Reengineering is a rebuilding activity:

  - Ex: the rebuilding of a house
- Software Reengineering Process Model defines 6 activities

  - Inventory Analysis, Doc Restructuring, Reverse Engg

  - Code Restructuring, Data Restructuring, Forward Engg

Software Reengineering



Software Reengineering

SCHOOL OF COMPUTER SCIENCE AND IT

•**Inventory analysis:**

- **inventory is a spreadsheet model** containing information that provides a detailed description of every active application.
- Reengineering may appear based on sorted information according to business criticality, longevity, current maintainability and supportability, and other locally important criteria, candidates. Resources can then be allocated to candidate applications for reengineering work.

•**Document restructuring**: options are

- program is relatively static, is coming to the end of its useful life then document it
- portions of the system that are currently undergoing change are fully documented
- The system is business critical and must be fully redocumented
- software organization must choose the one that is most appropriate for each case

Software Reengineering

- **Reverse engineering:**
  - the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction
  - a process of design recovery
  - tools extract data, architectural, and procedural design information from an existing program.

- **Code restructuring:**
  - when modules in existing are difficult to understand, test and maintain
  - code is then restructured or even rewritten in a more modern programming language
  - restructured code is reviewed and tested to ensure that no anomalies have been introduced

Software Reengineering

- **Data Structuring:**
  - if data structure is weak then restructure will be done
  - Data objects and attributes are identified, and existing data structures are reviewed for quality.

- **Forward engineering:**
  - Forward engineering recovers design information from existing software
  - uses this information to alter or reconstitute the existing system in an effort to improve its overall quality

<u>Reengineered software reimplements</u>

- **the function of the existing system** and
- also **adds new functions and/or improves** overall performance

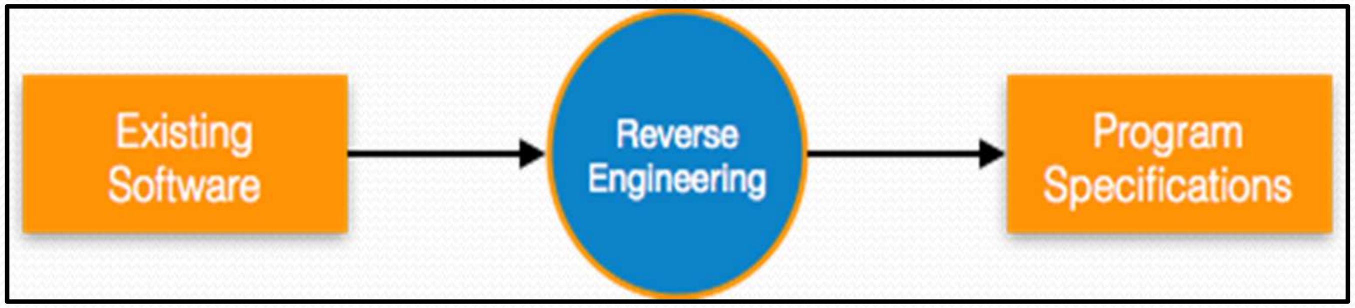
Reverse Engineering

- The process of

  - analyzing a subject system to identify the systems' components and their interrelationships

  - create representations of the system in another form or at a higher level of abstraction.

- Process to achieve system specification by thoroughly analyzing, understanding the existing system.
- This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

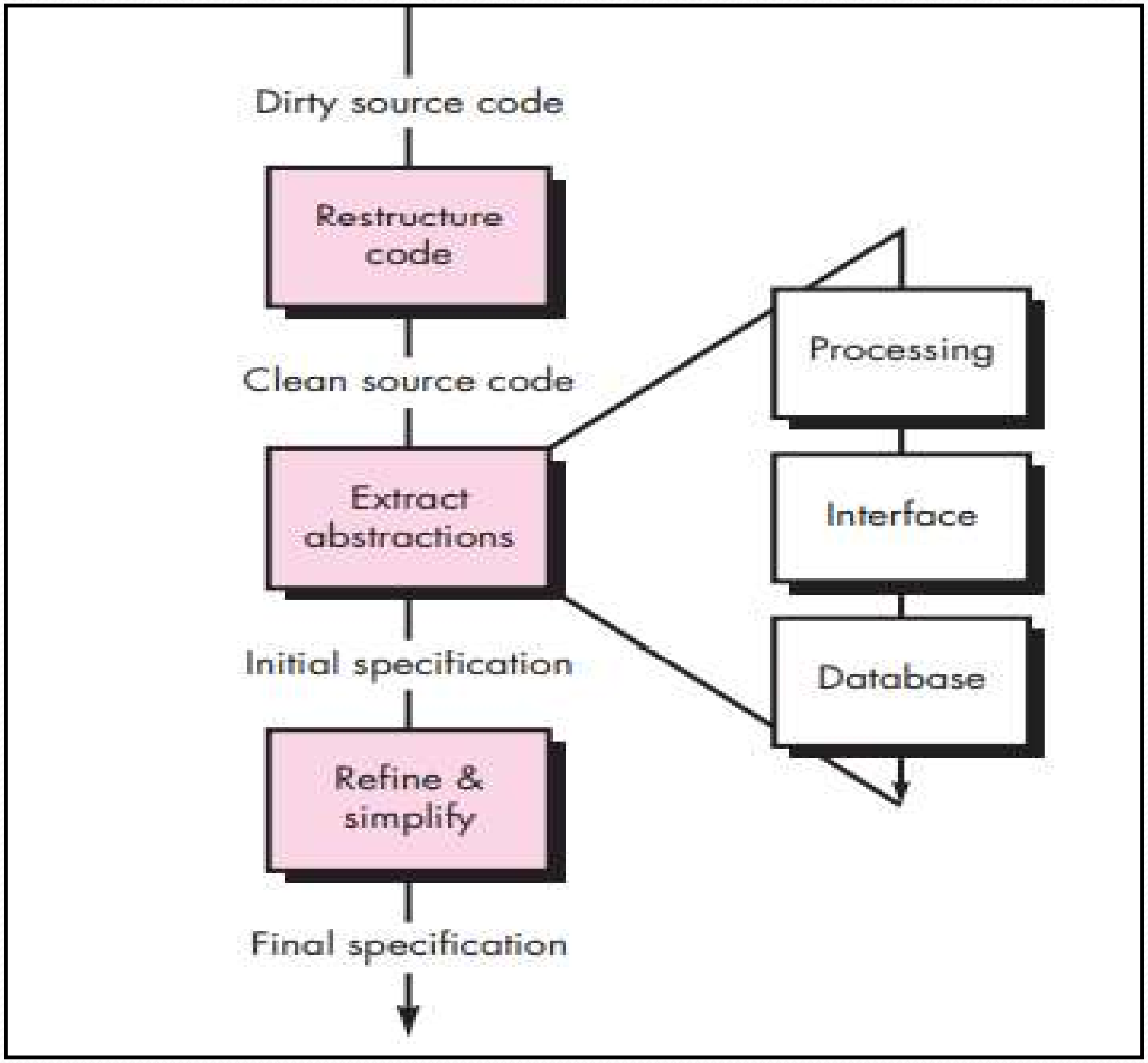


Reverse Engineering Concepts

- Abstraction level
  - ideally want to be able to derive design information at the highest level possible
- Completeness
  - level of detail provided at a given abstraction level
- Interactivity
  - degree to which humans are integrated with automated reverse engineering tools

- Directionality
  - one-way means the software engineer doing the maintenance activity is given all information extracted from source code
  - two-way means the information is fed to a reengineering tool that attempts to regenerate the old program
- Extract abstractions
  - meaningful specification of processing performed is derived from old source code

Reverse Engineering Process



Reverse Engineering Activities

- **RE to understand Data**
  - occurs at different levels of abstraction

    At the program level, internal program data structures must often be reverse engineered

    At the system level, global data structures (e.g., files, databases) are often reengineered to accommodate new DBMS paradigms
- **RE to understand Process**
  - begins with an attempt to understand and then extract procedural abstractions represented by the source code.
  - the code is analyzed at varying levels of abstraction: system, program, component, pattern, and statement
  - Functional Abstraction is found and Block diagram will be made to create Higher Level of Abstraction
  - Automated tools can be used to help enggrs understand the semantics of existing code

- **User interfaces**
  - what are the basic actions (e.g. key strokes or mouse operations) processed by the interface?
  - what is a compact description of the system's behavioral response to these actions?
  - what concept of equivalence of interfaces is relevant?

Reverse Engineering Applicability

- Reverse engineering often precedes reengineering
- Sometimes reverse engineering is preferred
  - if the specification and design of a system needs to be defined prior using them as input to the requirements specification process for a replacement systems
  - if the design and specification for a system is needed to support program maintenance activities

RESTRUCTURING

- Software restructuring modifies source code and/or data in an effort to make it amenable to future changes.
- In general, restructuring does not modify the **overall program architecture.**
- It tends to focus on the design details of individual modules and on local data structures defined within modules,
- If the restructuring effort extends beyond module boundaries and encompasses the software architecture, restructuring becomes forward engineering

Program modularisation/Code restructuring

- The process of re-organising a program so that related program parts are collected together in a single module
- Usually a manual process that is carried out by program inspection and re-organisation
- A resource exchange diagram maps each program module and the resources that are exchanged between it and other modules
- By creating representations of resource flow, the program architecture can be restructured to achieve minimum coupling among modules

Data Restructuring

- Before data restructuring can begin, a reverse engineering activity called **Analysis Of Source Code** should be conducted.
- The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called **Data Analysis**.
- After that, ***Data Redesign*** commences. In its simplest form, a ***data record standardization*** step clarifies data definitions to achieve consistency among data item names or physical record formats within an existing data structure or file format.
- Another form of redesign, called ***Data Name Rationalization***, ensures that all data naming conventions conform to local standards and that aliases are eliminated as data flow through the system.

- When restructuring moves beyond standardization and rationalization, physical modifications to existing data structures are made to make the data design more effective.
- This may mean a translation from one file format to another, or in some cases, translation from one type of database to another.

FORWARD ENGINEERING

- The forward engineering process applies software engineering principles, concepts, and methods to re-create an existing application.
- In most cases, forward engineering does not simply create a modern equivalent of an older program. Rather, new user and technology requirements are integrated into the reengineering effort.
- The redeveloped program extends the capabilities of the older application

FORWARD ENGINEERING

Forward Engineering for Client-Server Architectures

- centralized computing resources (including software) are distributed among many client platforms.
- a variety of different distributed environments can be designed
- typical mainframe application that is reengineered into a client-server architecture has 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 server site.

  - New communications, security, archiving, and control requirements must be established at both the client and server sites

FORWARD ENGINEERING

Three Layers of Abstraction can be identified

- **Database**

  - In some cases a new data model is created

  - the client-server database is reengineered to ensure that transactions are executed in a consistent manner

- **Business Rules Layer**

  - software resident at both the client and the server

  - software performs control and coordination tasks to ensure that transactions and queries between the client and database

- **Client applications layer**

  - implements business functions that are required by specific groups of end users.

FORWARD ENGINEERING

Forward Engineering for OO Architectures

First, the existing software is reverse engineered so that appropriate data, functional, and behavioral models can be created.

If the reengineered system extends the functionality or behavior of the original application, use cases are created.

Class hierarchies, object-relationship models, object-behavior models, and subsystems are defined

OO forward engineering progresses from analysis to design, a CBSE(Component Based Software Engineering) process model can be invoked it may be possible to reuse algorithms and data structures from the existing conventional application

THE ECONOMICS OF REENGINEERING

- Reengineering drains resources that can be used for other business purposes.
- Therefore, before an organization attempts to reengineer an existing application, it **should perform a cost-benefit analysis.**
- A cost-benefit analysis model for reengineering has been proposed by Sneed [Sne95]. Nine parameters are defined

P_1 = current annual maintenance cost for an application

P_2 = current annual operations cost for an application

P_3 = current annual business value of an application

P_4 = predicted annual maintenance cost after reengineering

P_5 = predicted annual operations cost after reengineering

P_6 = predicted annual business value after reengineering

P_7 = estimated reengineering costs

P_8 = estimated reengineering calendar time

P_9 = reengineering risk factor (P_9 = 1.0 is nominal)

L = expected life of the system

The cost associated with continuing maintenance of a candidate application (i.e., reengineering is not performed) can be defined as

$$C_{maint} = [P_3 - (P_1 + P_2)] \times L \qquad (29.1)$$

The costs associated with reengineering are defined using the following relationship:

$$C_{reeng} = P_6 - (P_4 + P_5) \times (L - P_8) - (P_7 \times P_9) \qquad (29.2)$$

Using the costs presented in Equations (29.1) and (29.2), the overall benefit of reengineering can be computed as

$$\text{Cost benefit} = C_{reeng} - C_{maint} \qquad (29.3)$$

- The cost-benefit analysis presented in these equations can be performed for all high priority applications identified during inventory analysis .
- Those applications that show the highest cost-benefit can be targeted for reengineering, while work on others can be postponed until resources are available.

Thank you!