Introduction- Software Maintenance

Mrs. Nisha,
School of Computing,
IIIT Una.

Evolution versus maintenance

- Software Evolution:
  - Software lifecycle.
  - Growth characteristics of software.
- RG Canning compared maintenance with iceberg.
- Maintenance categories:
  - Corrective maintenance.
  - Adaptive maintenance.
  - Perfective maintenance.
  - Preventive maintenance.
- Maintenance example in physical life.

Cont..

Software Maintenance

- **Preventing s/w from failing** to deliver the intended functionalities by means of bug fixing.
- All support activities carried out **after s/w delivery**.
- **Doesn't include changes to the architecture**.
- **Keeping the installed system running**.

Software Evolution

- **Continuous changes** from lesser, simpler or worse state to better/ higher state.
- All activities are carried out to **effect changes in requirements**.
- **New** but related design from existing system.

Software evolution

- **Eight laws**
  1. Continuous change
  2. Increasing complexity
  3. Self regulations
  4. Conservation of organizational stability
  5. Conservation of familiarity
  6. Continuing growth
  7. Declining quality
  8. Feedback systems

Software maintenance

- Software which is delivered, itself not complete. So, there is always a chance of maintenance / repair/ updating.
- SM is based on the intent of developer and motivation for change.
- Types of SM:
  - Corrective Maintenance
  - Adaptive Maintenance
  - Perfective Maintenance
  - Perfective Maintenance

Cont..

- Alternative way of classifying s/w modification is to simply categorize the **modification in terms of activities** performed:
  - Activities to make corrections
  - Activities to make enhancements
    - Enhancements that modify existing requirement.
    - Enhancements that creates new requirements.
    - Enhancements that modify the implementations without changing the requirements.

Cont..

- Another author expanded the defined topologies into an **evidence-based classification** of 12 different types of software:
  - Training
  - Consultive
  - Evaluative
  - Reformative
  - Updative
  - Groomative
  - Preventive
  - Performance
  - Adaptive
  - Reductive
  - Corrective
  - Enhancive

Cont..

- Three **objectives for classifying the types of software maintenance** are as follow:
  1. Classify maintenance tasks based on objective **evidence** that can be verified with observations or comparisons of software before and after modification.
  2. The **granularity** of the proposed classification can be made accurately reflect the actual mix of activities observed in the practice of software maintenance and evolution.
  3. The **classification of the groups** are independent of hardware platform, OS, design methodology, implementation language and availability of the personal doing the original development.

Maintenance of COTS- based systems

- New components are developed by combining commercial off-the-shelf (COTS) components, custom- built (in-house) components and open source software components.
- Obtained components are maintained by different vendors, different countries.
- Component based s/w system (CBS) is different from custom built software systems:
  - Skills of the system maintenance teams.
  - Infrastructure and organization
  - COTS maintenance cost
  - Larger user community
  - Modernization
  - Complex planning
  - Split maintenance function

Software evolution models and processes

- Software maintenance is a part of software development.
- **SMLC models in literature are**:
  1. Understanding the code.
  2. Modifying the code.
  3. Revalidation the code.
- Other models view software development as an iterative process and based on the idea of change mini- cycle as explained below:
  1. Iterative model
  2. Change mini cycle projects
     - Steps: change request, analyse and plan changes, implement changes, verify & validate and documentation change.

Cont..

3. Staged model of maintenance and evolution
    1. Initial development
    2. Evolution
    3. Servicing
        1. Focused on fixing bugs only.
        2. Effects are difficult to predict.
        3. Main focus on software and ignore the hardware part.
        4. Cost and risk of making changes are very significant.
    4. Phaseout : minimal system service and system enters to its final stage. Then the organisation decide to replace the system because of the various reasons:
        1. Too expensive to maintain.
        2. There is newer and better solution available.

Software maintenance standards

- Process of maintenance has been observed and measured.
- Reasons:
  - Dissemination of effective work practices.
  - Quick as compared to usual practice.
- Standards:
  - ISO called ISO/IEC 14764, which is a part of standard document ISO/IEC12207 for the life cycle process.
  - IEEE called IEEE/EIA 1219.
- Both describes processes to manage and executing activities for maintenance.

Phases Involved

IEEE/EIA 1219

- Problem identification
- Analysis
- Design
- Implementation
- System test
- Acceptance test
- Delivery

ISO/IEC 14764

- Process implementation
- Problem and modification analysis
- Modification implementation
- Maintenance review
- Migration
- Retirement

Software configuration management

- Configuration management is a discipline of managing changing in the large system.
- Goal : manage and control various extensions, adaptations and corrections that are applied to the system over its lifetime.
- SCM : Configuration management applied to software system.
  - For SCM, IEEE1042 std has been used.
  - SCM provides framework for managing changes in a controlled manner.
  - Purpose of SCM is to reduce communication error among personal working on different aspects of the software project.

Cont..

- SCM has 4 elements:
  - Identification of software configuration : identify different artifacts, baselines or milestones and changes to the artifacts.
  - Control of software configuration
  - Auditing s/w configuration
  - Accounting s/w configuration status : accounting of activities and history of the discussed steps.

Reengineering

- Single cycle of taking an existing system and generate new system.
- Reengineering = reverse engineering + Δ + Forward engineering

Reverse engineering includes examining the system and check for goals, models and the tools used.

Δ represents the changes performed on original system.

Forward engineering includes movement from the high level abstraction to logical and implementation level.

Legacy System

- Old programs that continues to be used because it still meets the user requirement.
- Different options available for legacy systems:
  - Freeze
  - Outsourced
  - Carry on maintenance
  - Discard and redevelop
  - Wrap
  - Migrate

Impact analysis

- Task of identifying portions of the software that can potentially be affected if the proposed change to the system is effected.
- Techniques used in impact analysis are categorized in 2 classes:
  - Traceability analysis
  - Dependency analysis

Refactoring

- Performing changes to the structure to make it easier to comprehend and cheaper to make subsequent changes without changing the observable behaviour system.
- It can be achieved through:
  - Removal of duplicate code.
  - Simplifying the code.
  - Moving to different classes.

Program comprehension and S/w reuse

- Program comprehension:
  - Prepare mental model.
  - Program understanding.
- Software reuse:
  - Data reuse
  - Architectural reuse
  - Design reuse
  - Program reuse

Outcomes are: better quality and increasing productivity.

Taxonomy of Software Maintenance and evolution

Mrs. Nisha,

SOC,

IIIT Una.

Introduction

- The ISO/IEC 14764 standard defines software maintenance as the totality of activities required to provide cost effective support to a software system.
- Development versus maintenance.
- Maintenance activities from three view points are as follow:
  1. Intention-based classification of SM activities
  2. Activity-based classification of SM activities
  3. Evidence-based classification of SM activities

Intention-based classification

- Corrective maintenance
  - Purpose is to correct failures like processing failure and performance failure.
- Adaptive maintenance
  - Enable system to adapt to changes in its environment or processing environment.
  - It includes changes, additions, deletions, modification, enhancements to meet the evolving needs of the environment.
- Perfective maintenance
  - Purpose is to make variety of improvements like user experience, processing efficiency, maintainability etc.
  - Activities for perfective maintenance: restructuring of the code, creating and updating documentations, tuning system to improve performance.

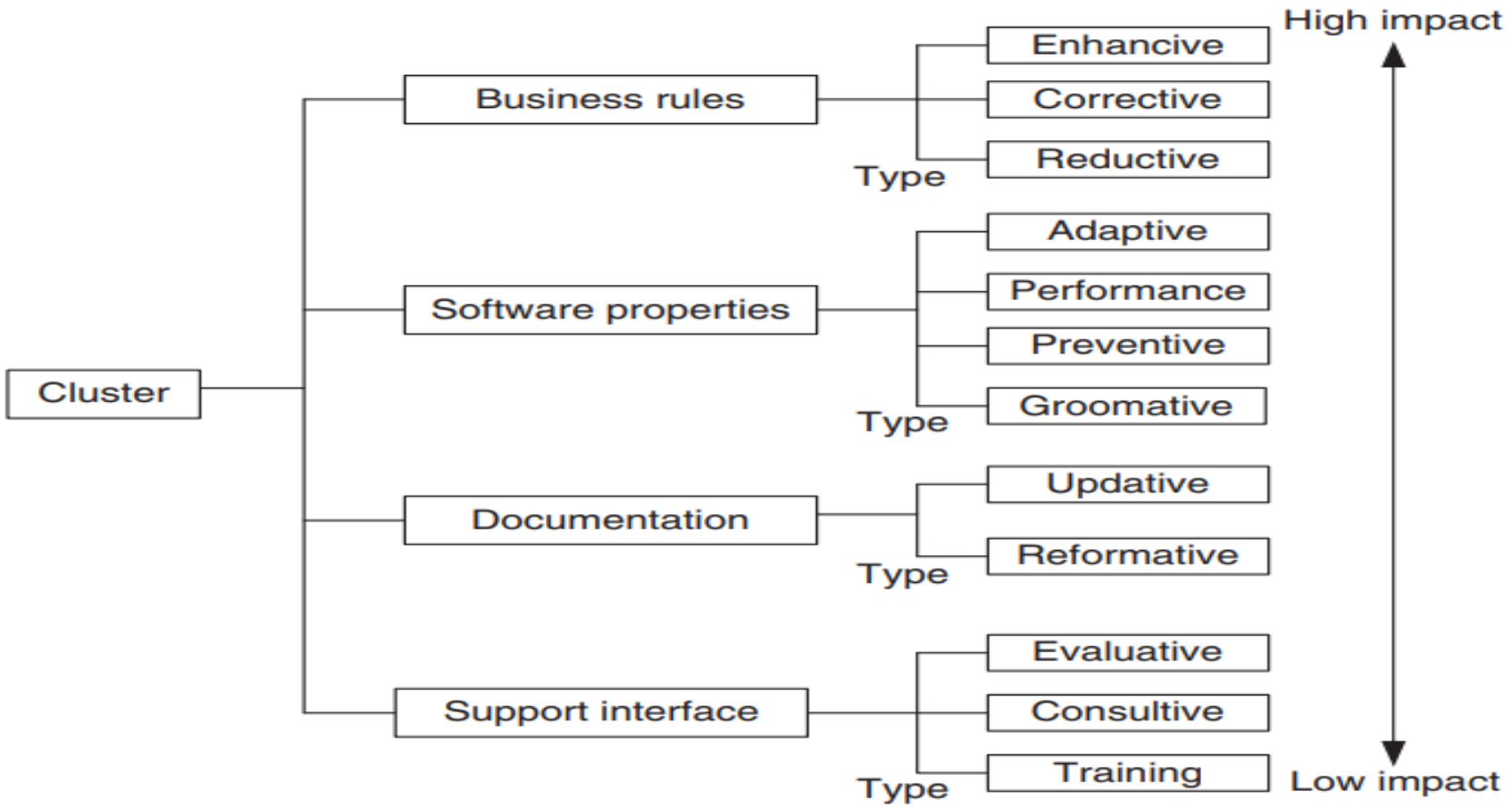
Cont..

- Preventive maintenance
  - Identify future risks and unknown problems.
  - It supports software rejuvenation, which is a preventive maintenance measure to prevent or at least postpone the occurrence of failure due to its continuous running state.
  - Also use proactive fault management framework.

Activity-based classification

1. Correction maintenance
    1. Focus is on fixing bugs.
    2. Measure the discrepancy between the expected behavior and the actual behavior of the system.
2. Enhancement maintenance :
    1. Activities that modify some of the existing requirements implemented by the systems.
    2. Activities that add new system requirement.
    3. Activities that modify the implementation without changing the requirements implemented by the systems.

Evidence-based classification : Group or cluster and their types



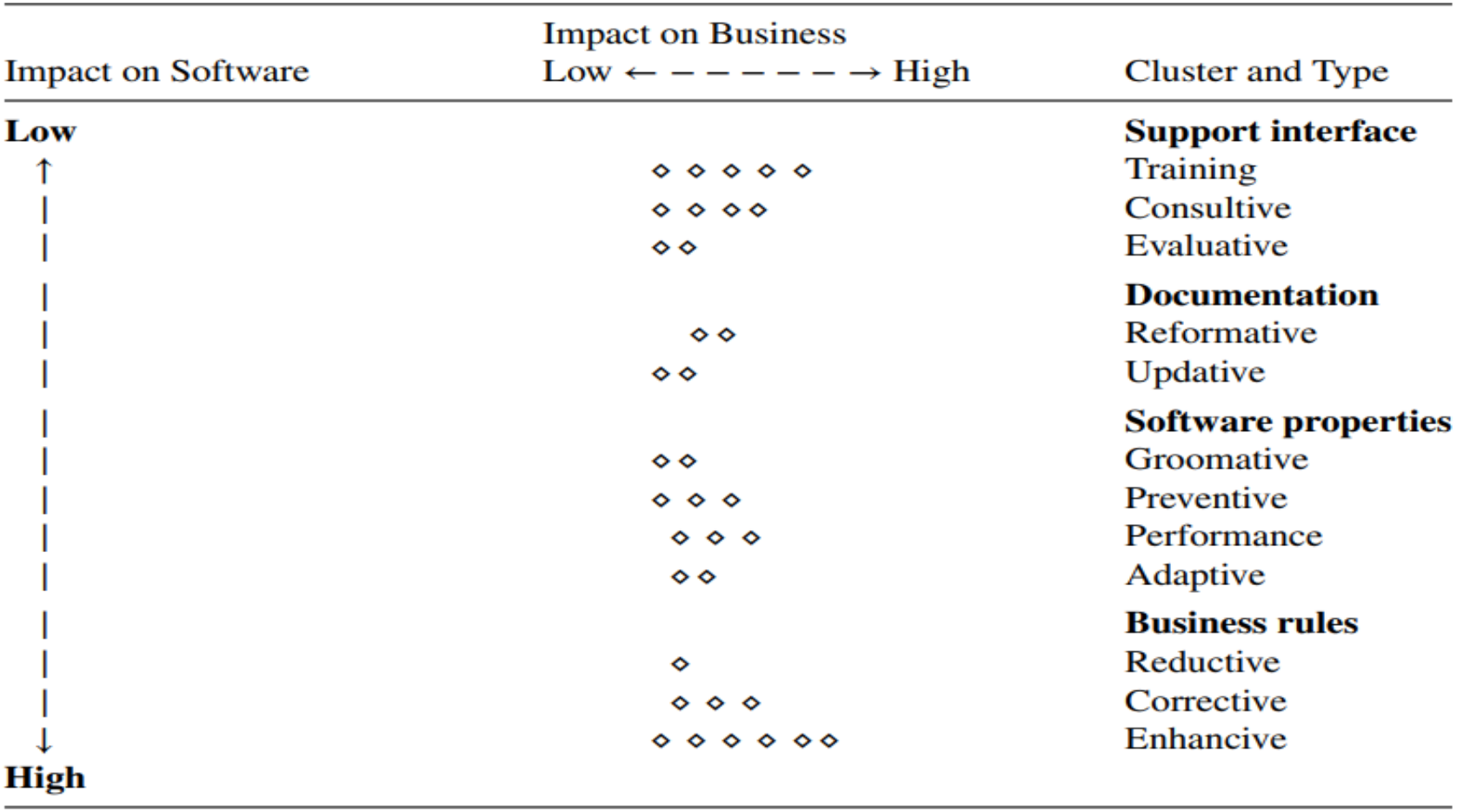
Evidence-Based 12 Mutually Exclusive Maintenance Types

1. Training : Training the stakeholders about software implementation.
2. Consultive: In this the cost and the length of time is estimated for maintenance.
3. Evaluation: Reviewing the program code and documentations, examining the ripple effects of the proposed change, designing and execute tests, finding the required data and debugging.
4. Reformative: Improve the readability of the document.
5. Updative
6. Groomative
7. Preventive
8. Performance

Cont..

9. Adaptive
10. Reductive
    1. Decreasing the generated data for customers.
    2. Decreasing the amount of input data.
    3. Decreasing the output data.
11. Corrective
12. Enhancive

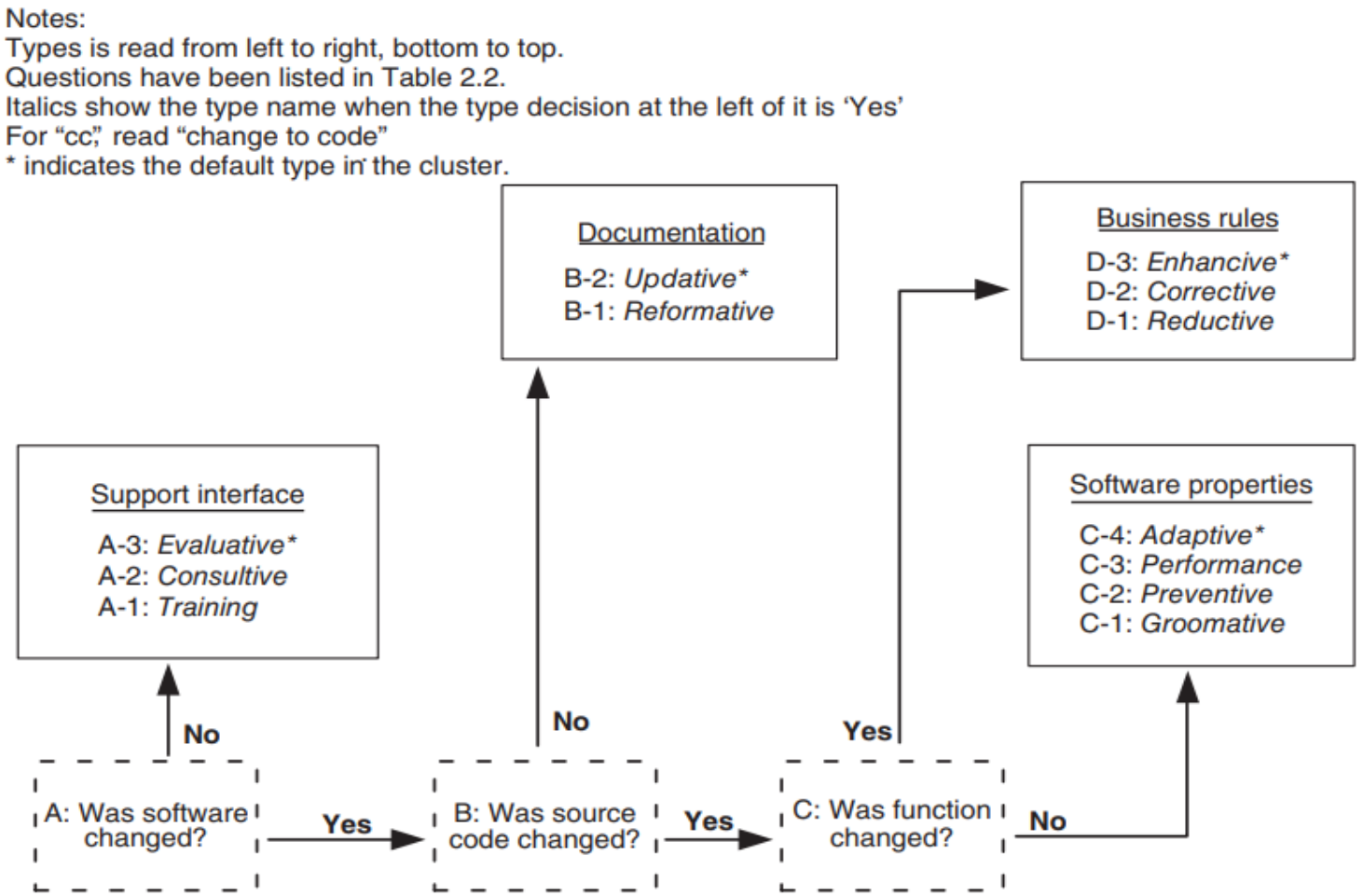
Impact of the types



Decision tree-based criteria

- Activities are classified into different types by applying a 2-step decision process:
  - First, apply criteria-based decisions to make a clusters of types.
  - Next, apply the type decisions to identify one type within the cluster.
- Three criteria decision involving A, B and C lead to the relevant cluster.
- Now identify the types within the cluster by using decision question.

Decision tree types



Summary of evidence-based types of software maintenance

Criteria	Type Decision Question	Type
A-1	To train the stakeholders, did the activities utilize the software as subject?	Training
A-2	As a basis for consultation, did the activities employ the software?	Consultive
A-3	Did the activities evaluate the software?	Evaluative
B-1	To meet stakeholder needs, did the activities modify the noncode documentation?	Reformative
B-2	To conform to implementation, did the activities modify the noncode documentation?	Updative
C-1	Was maintainability or security changed by the activities?	Groomative
C-2	Did the activities constrain the scope of future maintenance activities?	Preventive
C-3	Were performance properties or characteristics modified by the activities?	Performance
C-4	Were different technology or resources used by the activities?	Adaptive
D-1	Did the activities constrain, reduce, or remove some functionalities experienced by the customer?	Reductive
D-2	Did the activities fix bugs in customer-experienced functionality?	Corrective
D-3	Did the activities substitute, add to, or expand some functionalities experienced by the customers?	Enhancive

Support interface cluster

- Deals with the interaction between maintenance personnel with stakeholder or customers.
- If the reply for A criteria decision question is No, then it leads to the support interface cluster.
  - Type decision A-1 (training):
    - In-class lessons
    - Web-based training
  - Type decision A-2(consultive)
    - Estimating cost of the maintenance task
    - Support from the help desk
    - Specific knowledge about the system or resource
  - Type decision A-3(evaluative)
    - Searching and examining
    - Auditing
    - Diagnostic testing and regression testing
    - Understand the software without modifying
    - Computing different types of matrices

Documentation cluster

- Type decision B-1 (reformative):
  - Improves the readability of the documentation
  - Change the documentation to incorporate the effects of modifications in the manuals.
  - Prepare training material
  - Change the style of the documentation for non-code entities.
- Type decision B-2 (updative)
  - Replacing out-of-date documentation with up-to-date documentation
  - Making semi formal models to describe current source code
  - Combining test plans with documentation, without modifying code.

Software properties cluster

- Type decision C-1 (groomative):involves anti-regression activities.
  - Substituting algorithms and modules with better ones.
  - Modifying data conventions.
  - Making backups.
  - Changing access authorities.
  - Altering code understandability.
  - Recompiling the code.
- Type decision C-2 (preventive)
  - Activities generally not visible to the user.
- Type decision C-3 (performance)
  - Substituting algorithms or methods for better efficiency.
  - Reducing storage demand.
  - Increases system's robustness and reliability.
- Type decision C-4 (adaptive)
  - Porting the software to a new execution platform.
  - Increase utilization of COTS.
  - Changing the supported communication protocols.
  - Moving to object oriented technologies.

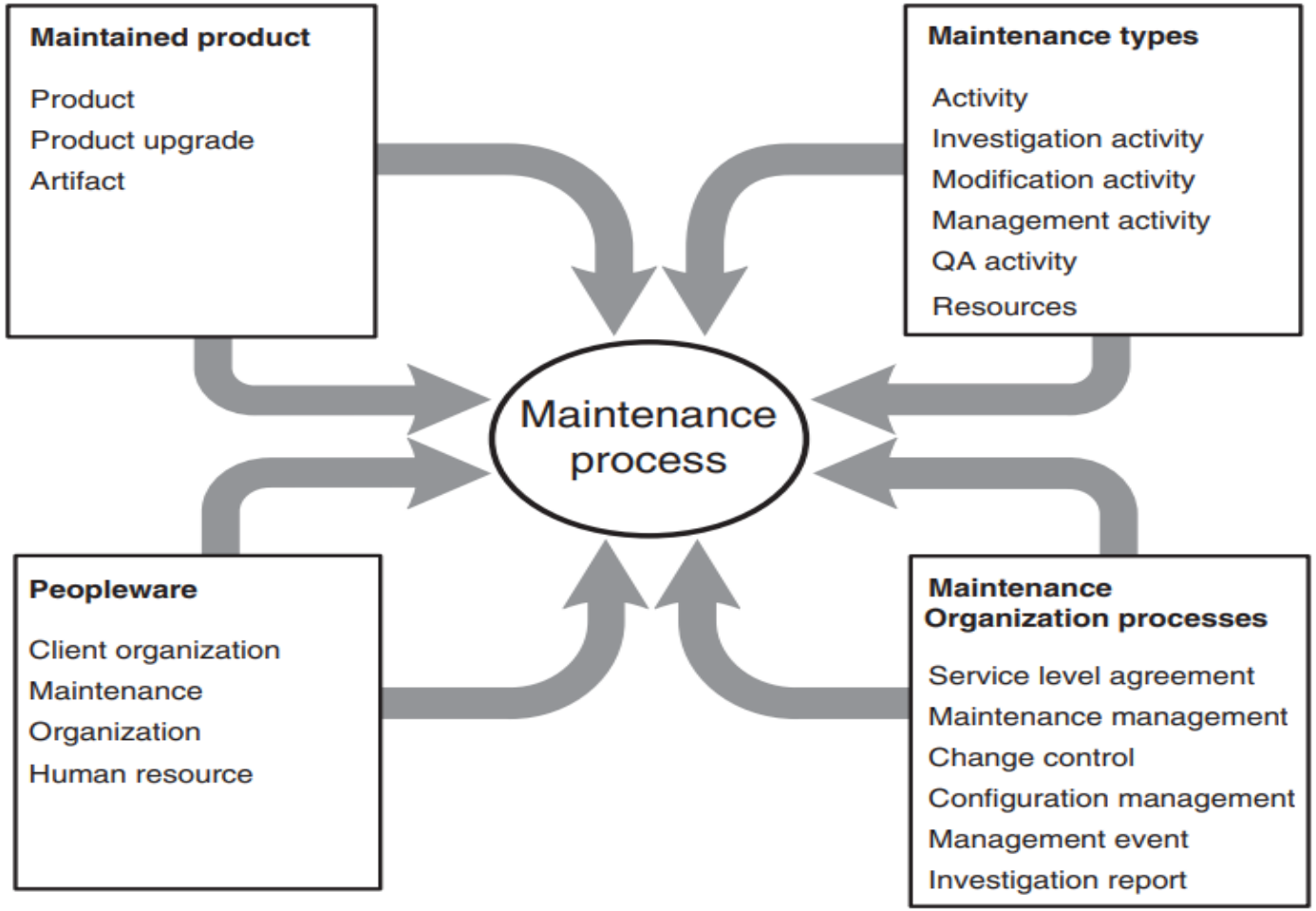
Business rules cluster

- Type decision D-1 (reductive): normally encounter when organizations merge.
  - Remove modules
  - Removes business rules
- Type decision D-2 (corrective)
- Type decision D-3 (enhancive): default type of the cluster.
  - Add new subsystem or algorithms
  - Modify some module to enhance their scope.

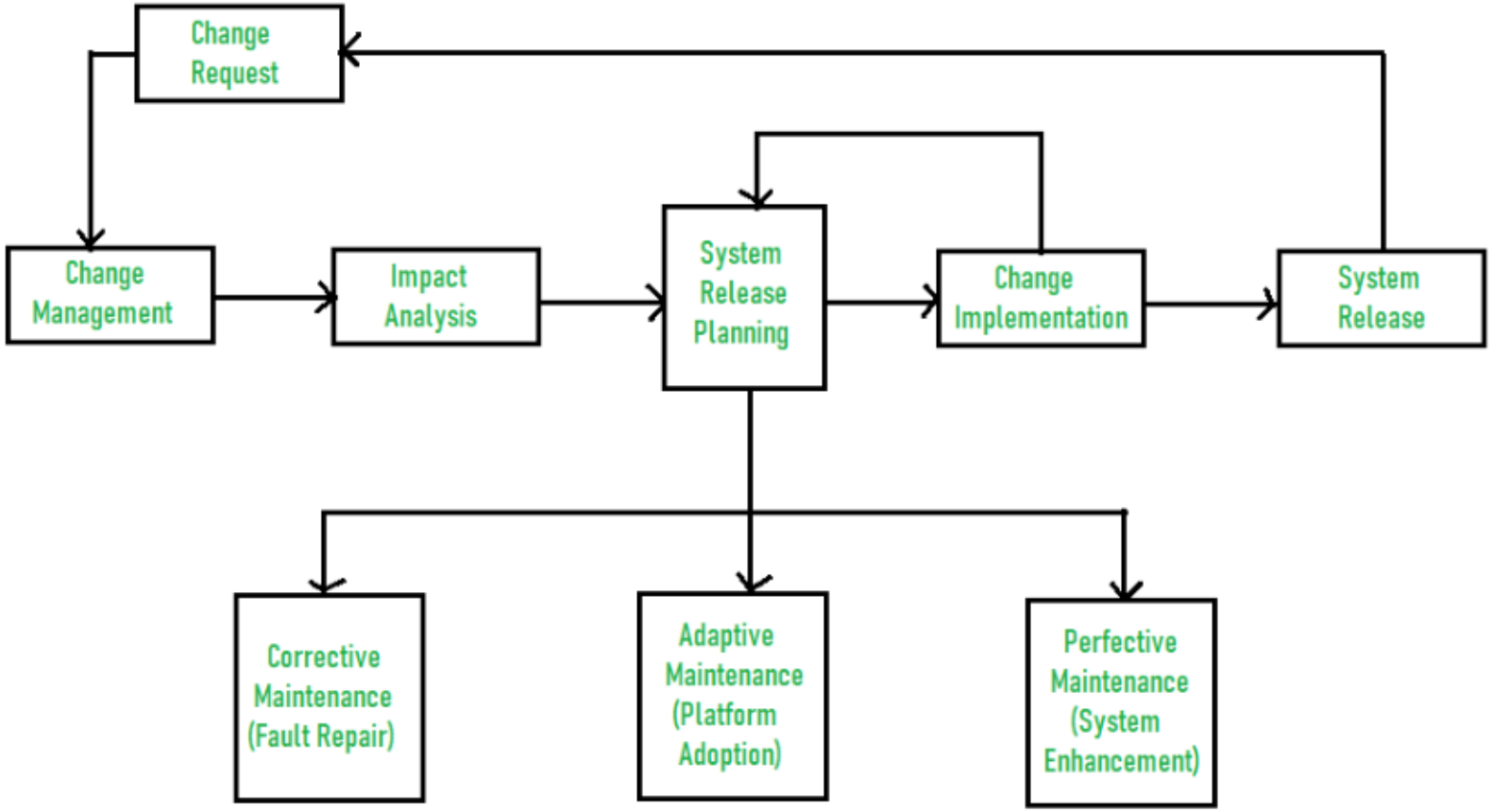
Categories affecting of maintenance concepts

- Ways to organize maintenance activities or the key factors influencing the maintenance process are identified.
  - Product to be maintained
  - Type of maintenance to be performed
  - Maintenance organization processes to be followed
  - The peopleware involved, that is, the people in the maintenance organization and in the customer/client organization.

Cont..
Categories affecting software maintenance



Software Maintenance Life Cycle

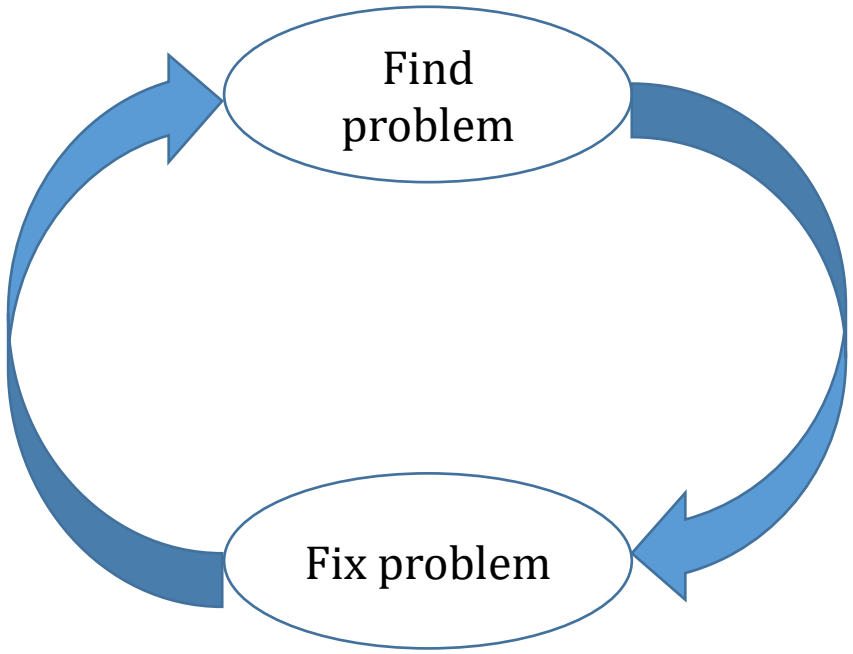


Software maintenance models

- Quick and fix model
- Iterative model
- Reuse oriented model
- Boehm model
- Taute maintenance model

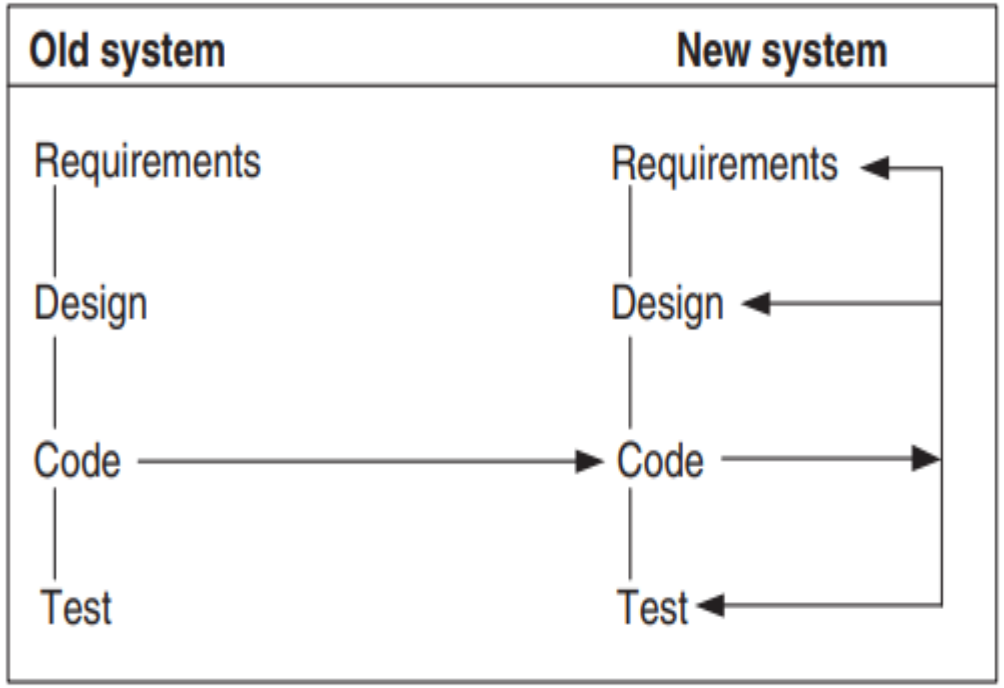
Quick and fix model

- Adhoc approach of maintenance.
- Objective of the model is to identify the problem and then fix it.
- Future consequences are ignored.
- Ignore the impact on the system.
- Advantages:
  - Quick and low cost
  - Not a time consuming process
  - Suitable for time bound and limited resources situation.

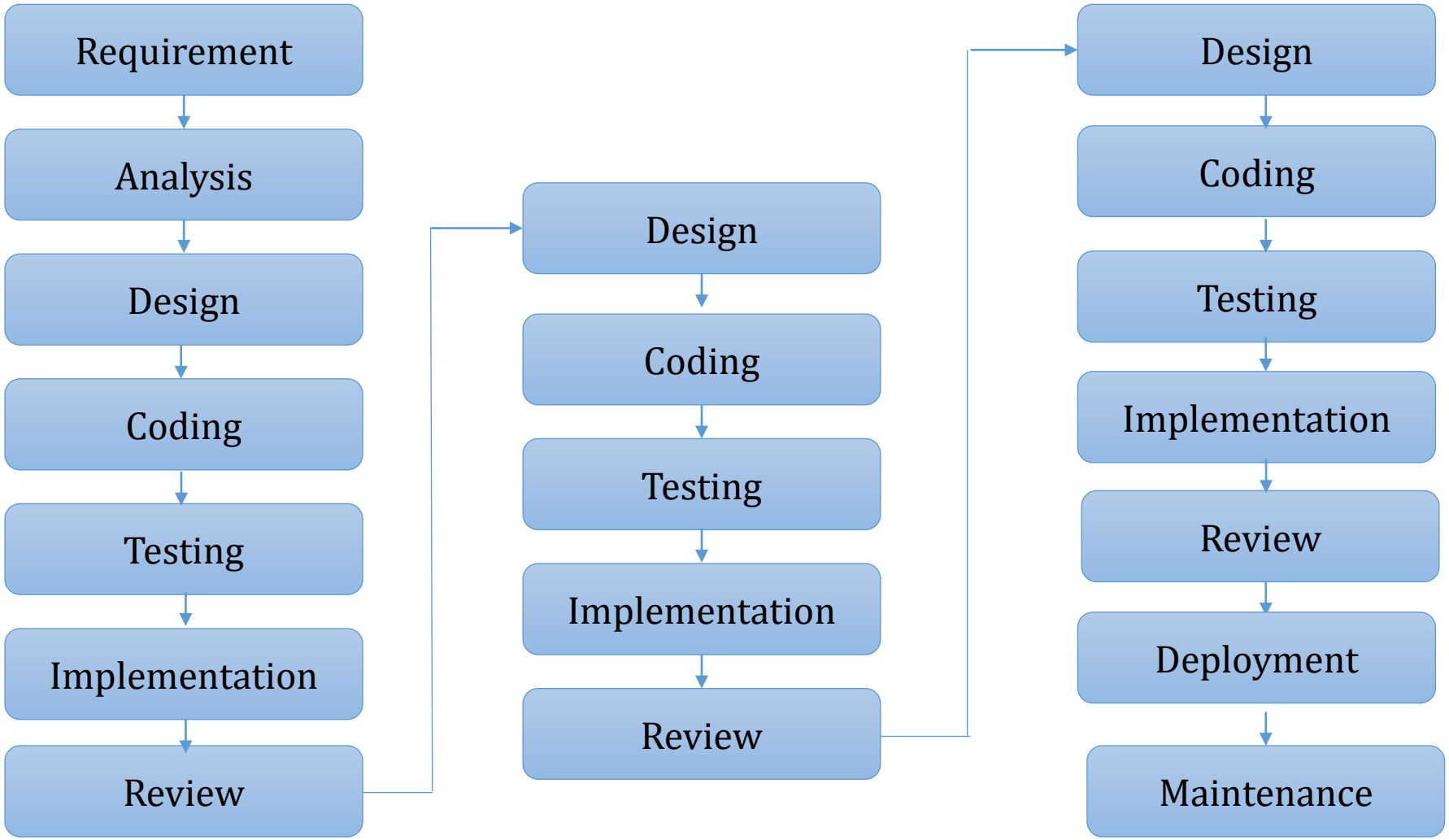


Quick and fix model (Cont..)

- Disadvantage:
  - Not suitable for large projects.
  - Unstructured way of maintenance.
  - Leads to system degradation.



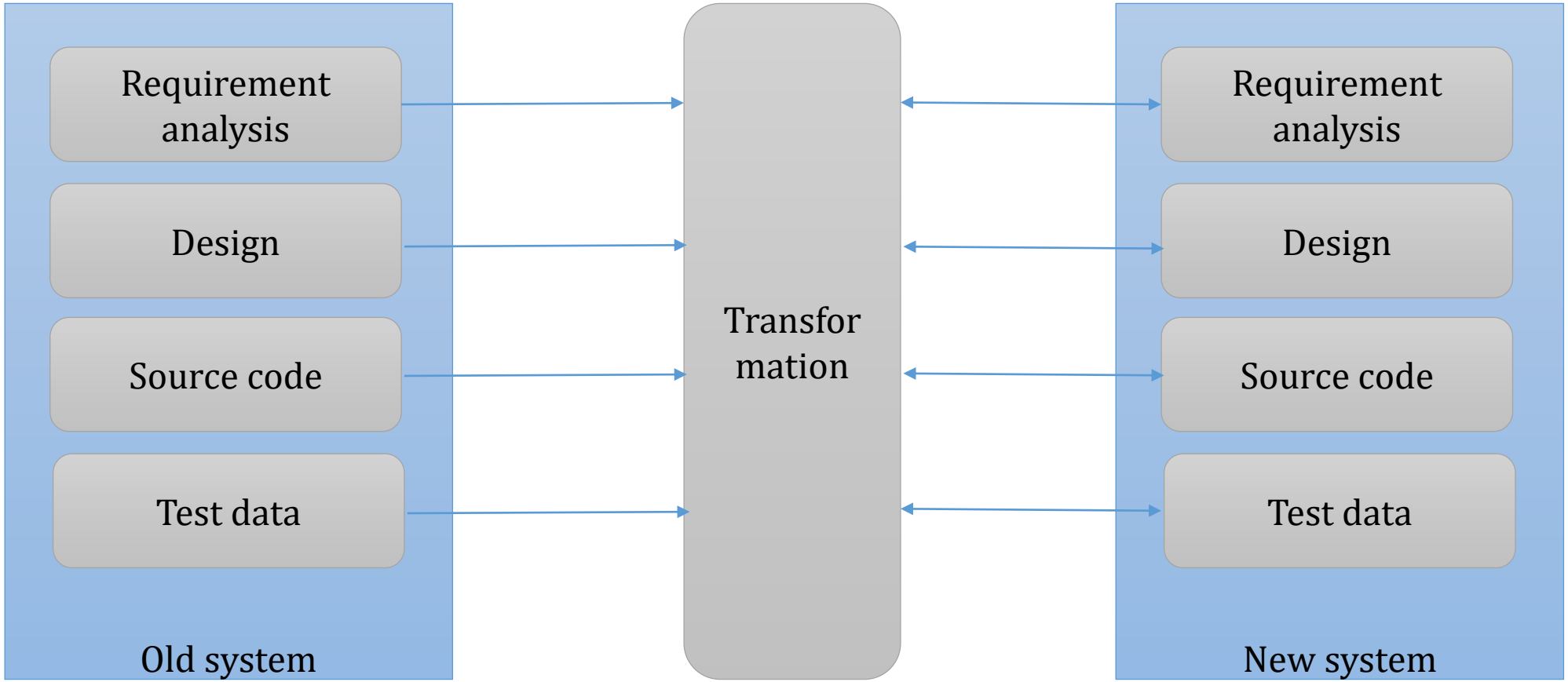
Iterative enhancement model



Iterative enhancement model (Cont.)

- Advantages
  - Testing and debugging during smaller iteration is easy.
  - A Parallel development can plan.
  - It is easily acceptable to ever-changing needs of the project.
  - Risks are identified and resolved during iteration.
  - Limited time spent on documentation and extra time on designing.
- Disadvantages
  - It is not suitable for smaller projects.
  - More Resources may be required.
  - Design can be changed again and again because of imperfect requirements.
  - Requirement changes can cause over budget.
  - Project completion date not confirmed because of changing requirements.

Reuse based model



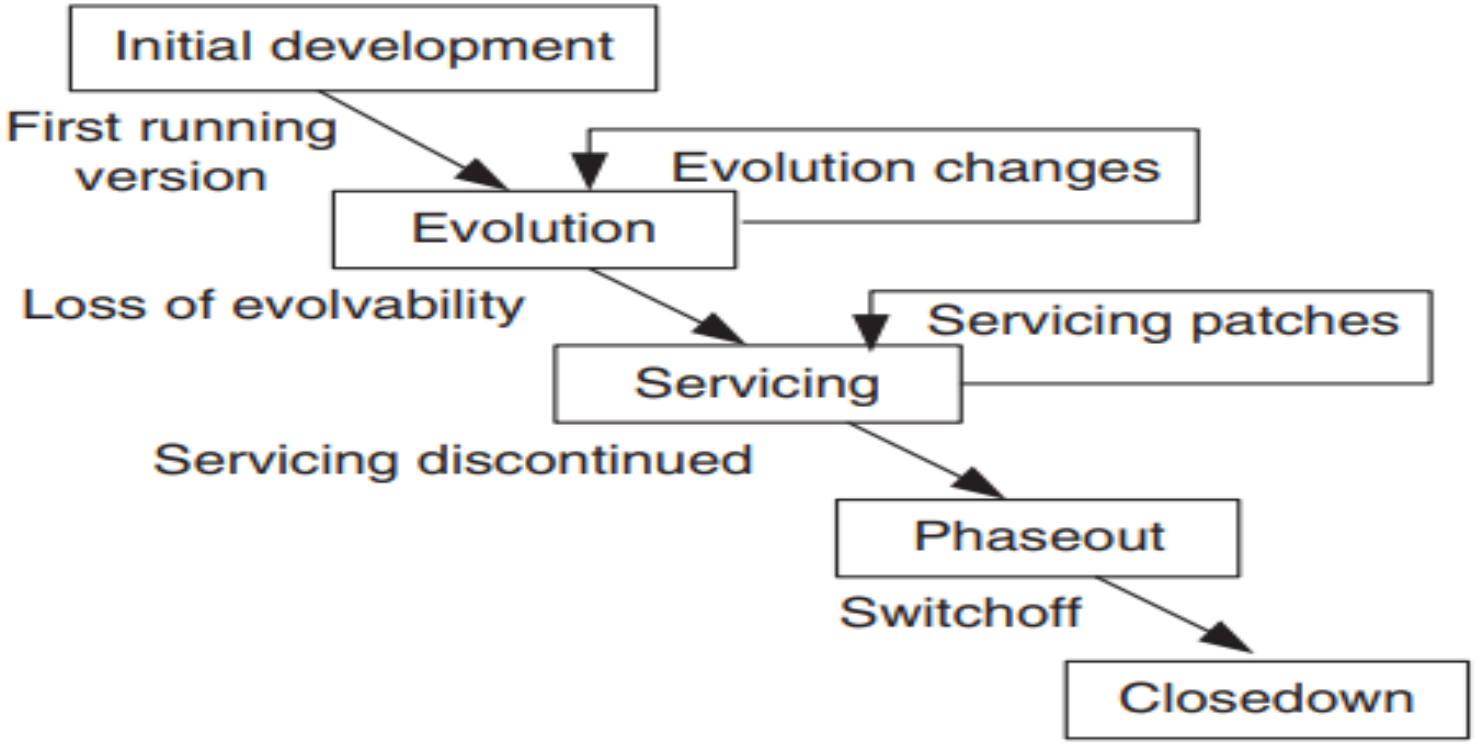
Reuse based model (Cont.)

- It consists of the following steps.
  - Identifying the components of the old system which can be reused
  - Understanding these components
  - Modifying the old system components so that they can be used in the new system
  - Integrating the modified components into the new system.
- Advantages
  - It can reduce total cost of software development.
  - The risk factor is very low.
  - It can save lots of time and effort.
  - It is very efficient in nature.

Reuse based model (Cont.)

- Disadvantages:
  - Reuse-oriented model is not always worked as a practice in its true form.
  - Compromises in requirements may lead to a system that does not fulfill requirement of user.
  - Sometimes using old system component, that is not compatible with new version of component, this may lead to an impact on system evolution.

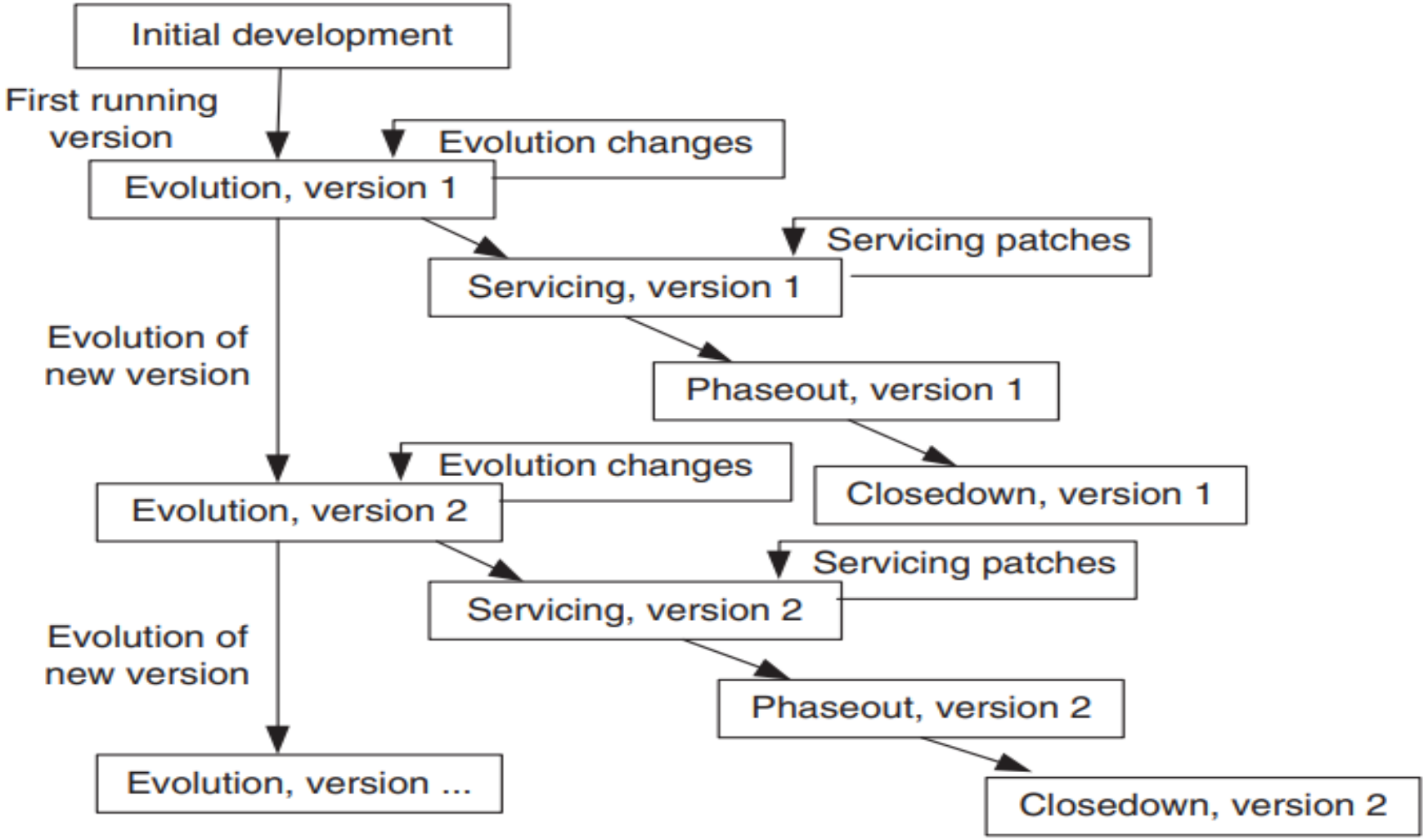
The staged model for closed source software



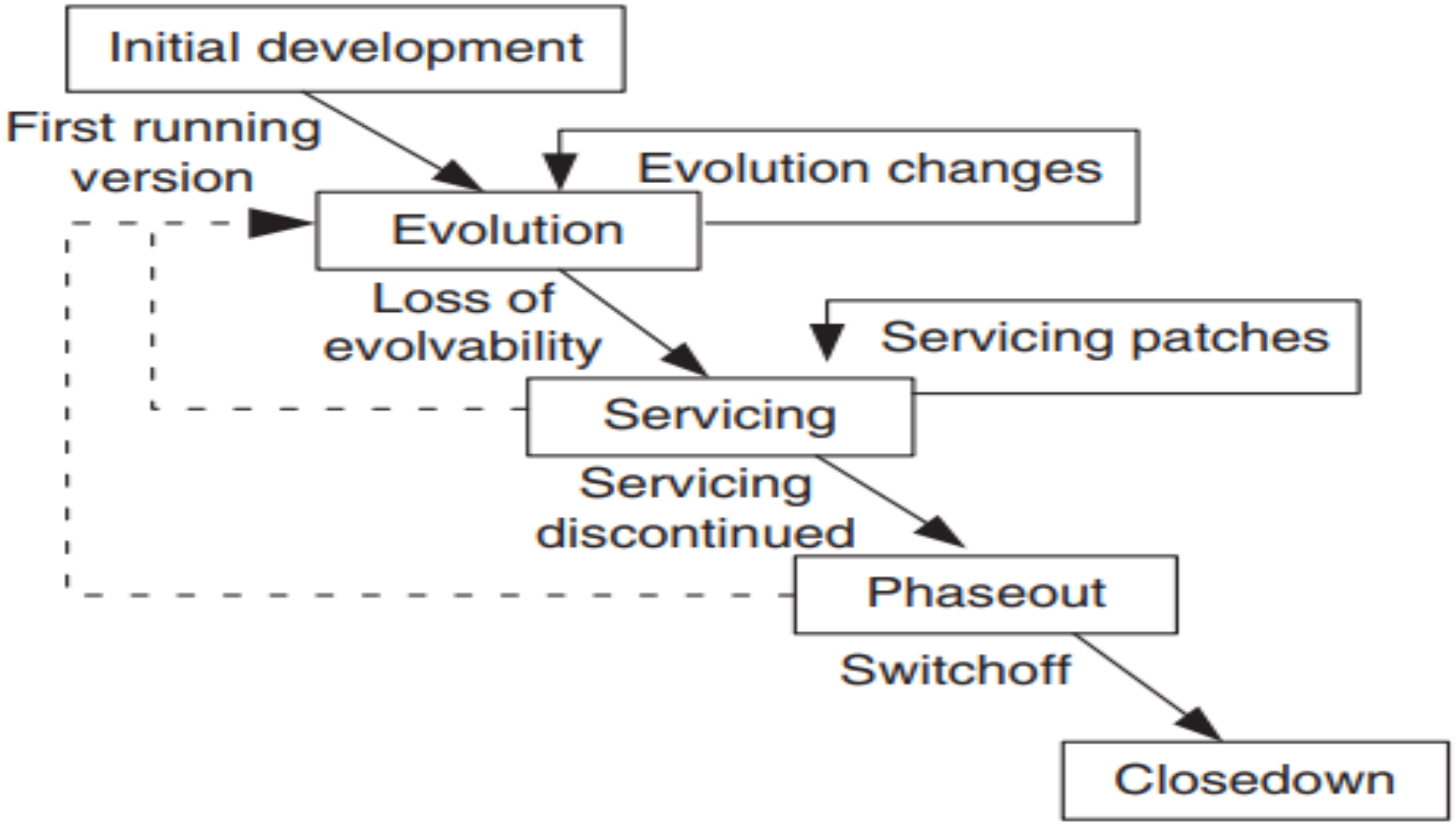
The staged model for closed source software (Cont.)

- Initial Development:
  - Develop 1st functioning version of the software.
  - Based on software architecture and team knowledge.
- Evolution : Developers improve the functionalities and capabilities of the software to meet the needs and expectations of the customer.
  - It is based on the customer feedback.
- Servicing
- Phaseout
- Closedown: withdrawn from the market.

Versioned staged model for the closed source software (CSS)



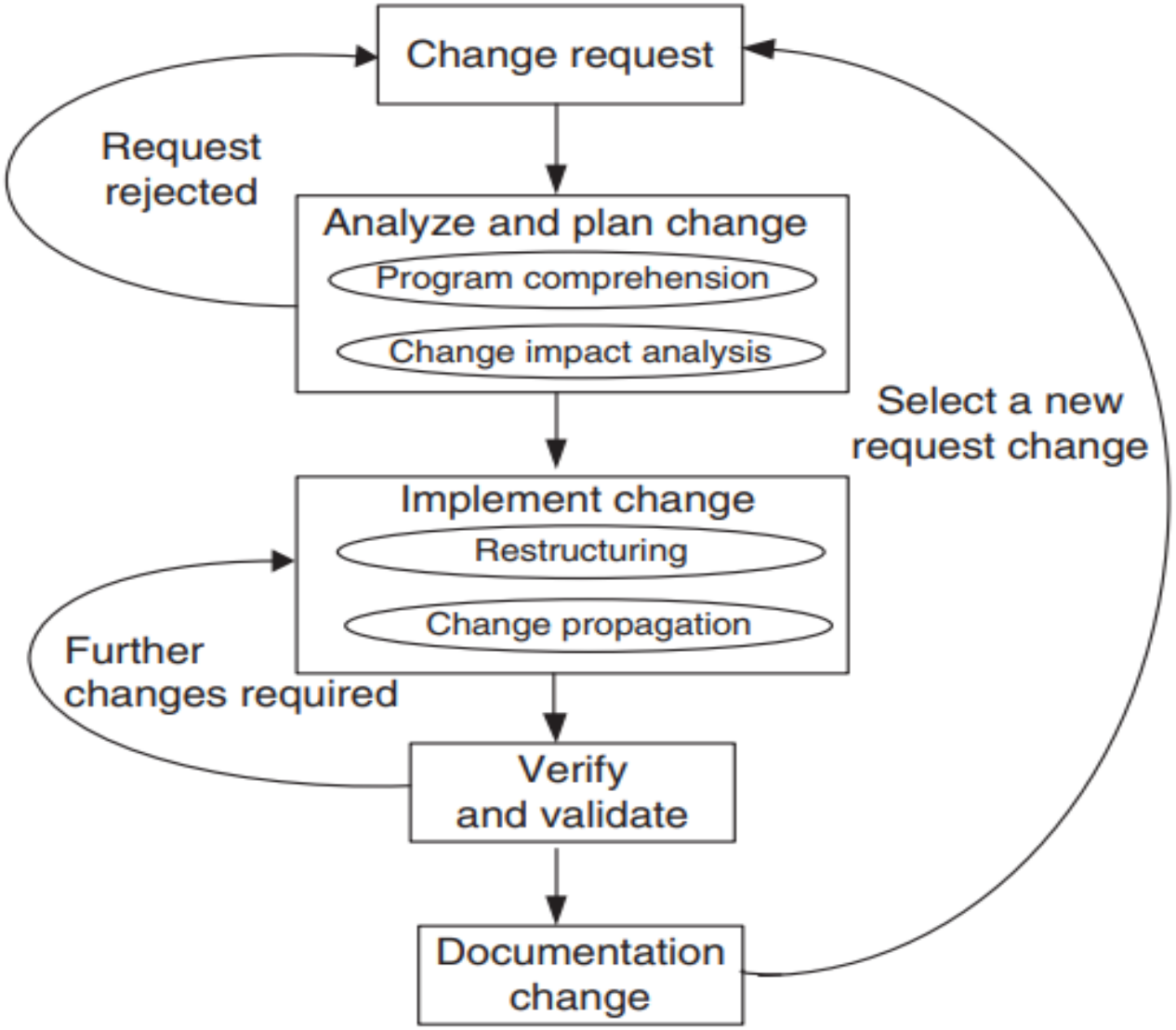
The staged model for free, Libre, open source software



CSS versus FLOSS maintenance

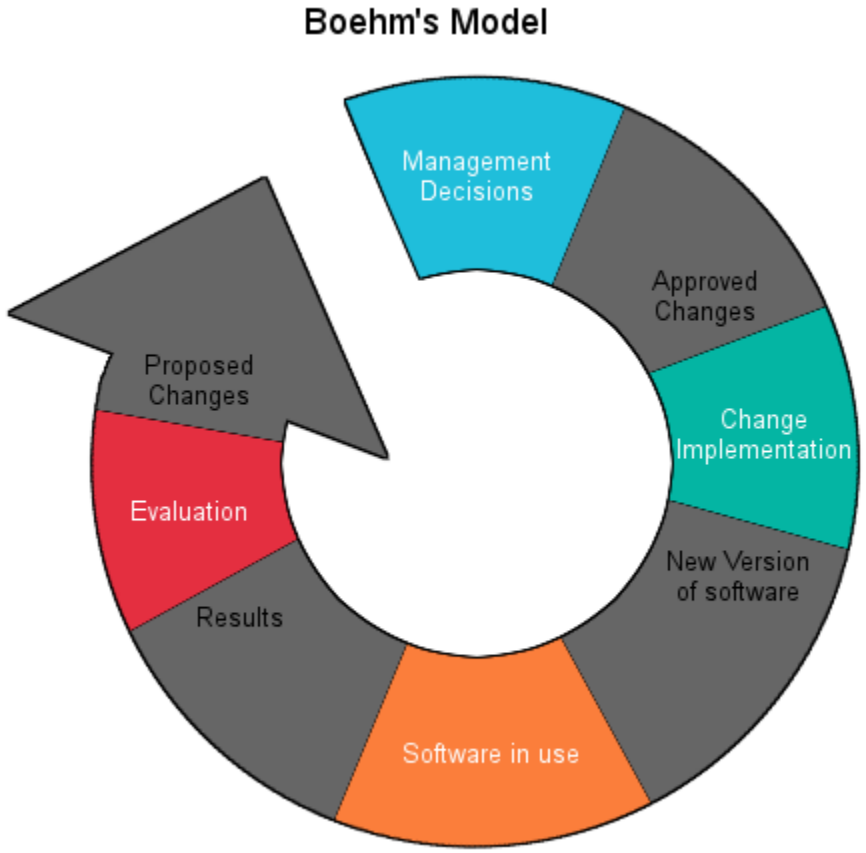
- No evolution track for FLOSS systems.
- Transition from servicing to evolution.
- Transition from phaseout to evolution.

Change mini-cycle model



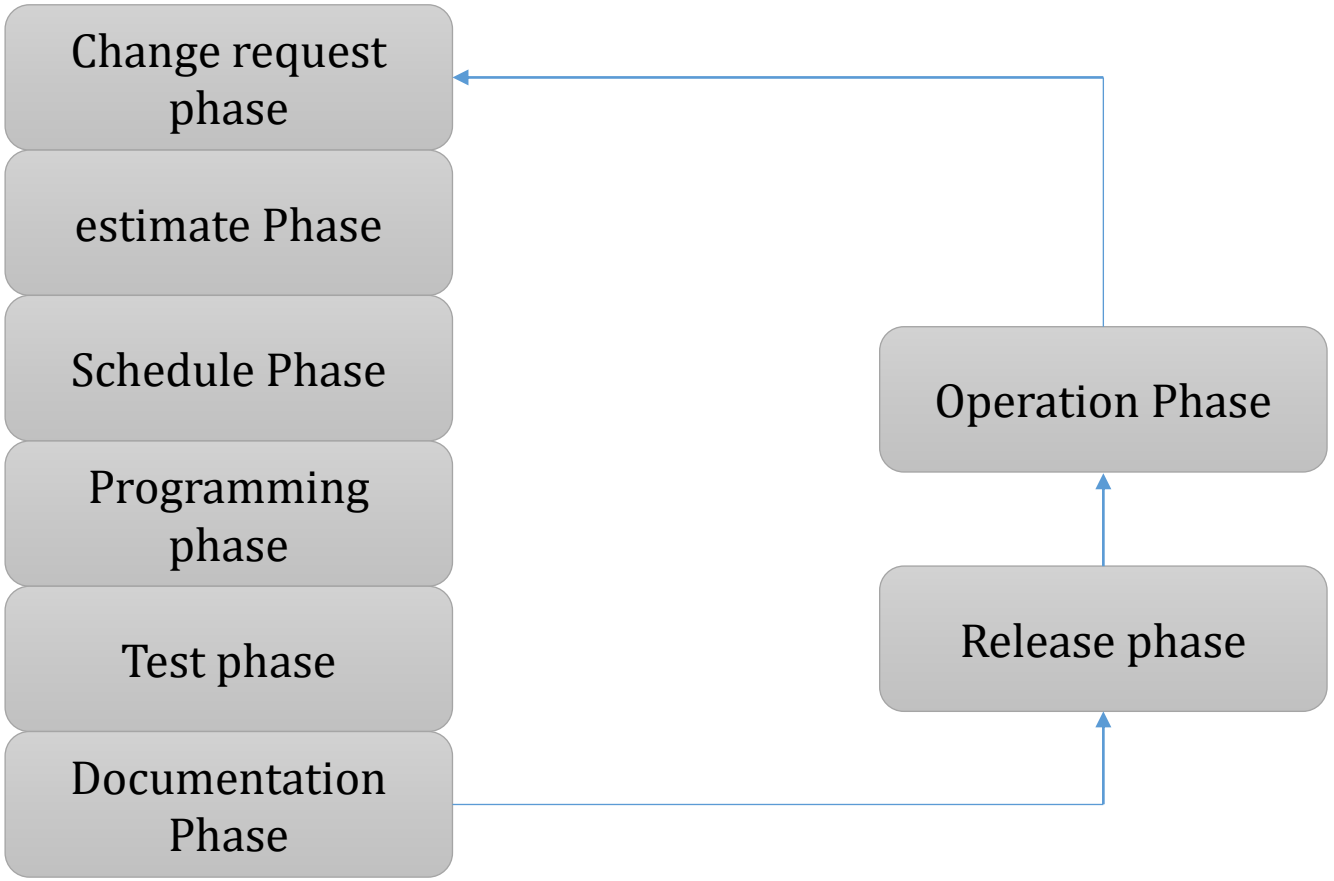
Boehm maintenance model

- Closed-loop cycle.
- Based on the economic models and principles.
- Boehm proposed a formula for calculating the maintenance cost as it is a part of the COCOMO Model.



Taute model

- Closed-loop cycle.



SCM Functions

- SCM functions can be categorized in three types:
  - Product
    - Identification
    - Version Control
    - System model and selection
  - Tools
    - Workspace control
    - Building
    - Change management
  - Process
    - Change management
    - Status accounting
    - Auditing

Version control

- Version control: Version control, also known as source control, is the practice of tracking and managing changes to software code.
  - keeps track of every modification.
  - If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to other team members.
- Version control system: Version control systems are software tools that help software teams manage changes to source code over time.
- VCS (version control system) or SCM(source code management) or RCS (revision control system)
  - version control systems help software teams work faster and smarter.

Version control (Cont..)

- Used during artifacts evolution and assign new identifiers.
- Why VC is required?
  - To track changes
  - Backtracking
- All versions are stored independently.
- VC consists of Master copy and working copies.
- Working copies can be converted to master copy by commiting.
- If multiple users accesses the files in same time. Then 2 methods are used to avoid concurrency control:
  - Lock- modify-unlock
  - Copy-modify merge
- VC also supports branching of the versions.

Version control (Cont..)

- Types of VCS:
  - Local Version Control Systems
  - Centralized Version Control Systems
    - Commit and update
    - More effective as compared LVCS.
  - Distributed Version Control Systems
    - Each user has Individual repository.
    - Commit, push, pull and update.

- Benefits of VCS:
  - Maintain change history.
  - Branching and merging.
  - Allow multiple contributors from different geographical locations.
  - Helps in recovery.
  - Merged only validated copy of the contribution.
  - Traceability
    - Uses tools like Jira for tracing change.

Unit II- Cost estimation Models

Mrs. Nisha Gautam
School of Computing
IIIT Una.

Cost Estimation Techniques

- Financial spent on software maintenance.
- Three techniques or models used for cost estimation:
  - Empirical estimation technique:
    - Uses derived formulas based on collected data, guesses and prior experience.
    - Uses size of the software to estimate efforts.
    - Example : Delphi technique.
  - Heuristic technique:
    - Means ‘to discover’ by solving problems, learning or discovery of methods used for achieving goals.
    - Flexible and simple to take decisions.
    - Example : COCOMO
  - Analytical estimation technique:
    - Used to measure work.
    - Tasks are broken for analysis.
    - Example: Halstead’s software science.

Project size estimation technique

1. Expert judgement : Expert decide based on their judgement and experience.
2. Analogous estimation : estimating project size based on the similarity between current and previously completed project.
    1. Suits best where historical data is available.
3. Bottom-up estimation : based on module estimation.
4. Three-point estimation : based on optimistic, pessimistic and most likely values. Example: PERT.
5. Function points : based on the functionality provided by the software. Considered factors are input, output, inquiries and files to arrive at the project size estimate.

Project size estimation technique (Cont.)

6. Use case points : Project size is based on the no. of use cases supported by software.
    1. UCP consider factors such as complexity of each use case.
    2. Number of actors involved.
    3. Number of use cases.
7. LOC : Units of LOC are:
    1. KLOC : Thousand lines of code.
    2. NLOC : Non-comment lines of codes.
    3. KDSI : Thousand of delivered source instructions.

    * Two similar codes may not require same effort.

Project size estimation technique (Cont.)

8. Function point analysis:
    1. Count the number of functions of each proposed type : External I/P, External O/P, external inquiries, internal files (logical files not log files), external interface files.
    2. Compute the unadjusted function points (UFP) : assign weight to the complexity (simple, average and complex).

UFP = sigma (count of each function type * weight factor)

Function type	Simple	Average	Complex
External input	3	4	6
External output	4	5	7
External inquiries	3	4	6
Internal files	7	10	15
External interface files	5	7	10

3. Find the total degree of influence (TDI) : included 14 general characteristics like data communication, performance, heavily used configuration, transaction rate, online data entry, end user efficiency and so on. and it is evaluated on the scale 0-5.
4. Compute value adjustment factors (VAF).

$$VAF = (TDI * 0.01) + 0.65$$

5. Compute function point count (FPC).

$$FPC = UFP * VAF$$

COCOMO Model

- The COCOMO (Constructive Cost Model) is one of the most popularly used software cost estimation models i.e. it estimates or predicts the **effort** required for the project, **total project cost** and **scheduled time (time to complete the project)** for the project.
- It estimates the required **number of Man-Months (MM)** for the full development of software products.
- This model depends on the **number of lines** of code for software product development.
- It was developed by a software engineer Barry Boehm in 1981.

COCOMO Model (Cont..)

- **Advantages**
  - Provides a **systematic way to estimate the cost and effort** of a software project.
  - Can be used to **estimate the cost and effort of a software project at different stages** of the development process.
  - Helps in **identifying the factors that have the greatest impact on the cost and effort** of a software project.
  - Can be used to **evaluate the feasibility** of a software project by estimating the cost and effort required to complete it.
- **Disadvantages**
  - Assumes that the **size of the software is the main factor** that determines the cost and effort of a software project, which may not always be the case.
  - **Does not take into account the specific characteristics** of the development team, which can have a significant impact on the cost and effort of a software project.
  - **Does not provide a precise estimate** of the cost and effort of a software project, as it is based on assumptions and averages.

COCOMO Model (Cont..)

- There are three models depends on the project's complexity

  1. **Organic Project**

  It belongs to **small & simple software projects** which are handled by a small team with **good domain knowledge** and few rigid requirements.

  Example: Small data processing or Inventory management system.

  2. **Semidetached Project**

  It is an **intermediate (in terms of size and complexity) project**, where the **team having mixed experience** (both experience & inexperience resources) to deals with **rigid/nonrigid requirements**.

  Example: Database design or OS development.

  3. **Embedded Project**

  This project having a **high level of complexity** with a large team size by considering all sets of **parameters (software, hardware and operational)** to deal with **rigid requirements.**

  Example: Banking software or Traffic light control software.

The Basic COCOMO model

- Type of static model to **estimates software development effort quickly and roughly.**
- It mainly deals with the **number of lines of code and the level of estimation accuracy is less** as we don't consider the all parameters belongs to the project. The estimated effort and scheduled time for the project are given by the relation:

Effort (E) = a∗(KLOC)b MM

Scheduled Time (D) = c∗(E)d Months(M)

Where, E = Total effort required for the project in Man-Months (MM).

D = Total time required for project development in Months (M).

KLOC = The size of the code for the project in Kilo lines of code.

a, b, c, d = The constant parameters for a software project.

PROJECT TYPE	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

The intermediate COCOMO

The intermediate model **estimates software development effort in terms of size of the program and other related cost drivers parameters** (product parameter, hardware parameter, resource parameter, and project parameter) of the project. The estimated effort and scheduled time are given by the relationship:

Effort (E) = a∗(KLOC)b *EAF MM

Scheduled Time (D) = c∗(E)d Months(M)

Where, E = Total effort required for the project in Man-Months (MM).

D = Total time required for project development in Months (M).

KLOC = The size of the code for the project in Kilo lines of code.

a, b, c, d = The constant parameters for the software project.

EAF = It is an **Effort Adjustment Factor**, which is calculated by multiplying the parameter values of different cost driver parameters. For ideal, the value is 1.

The intermediate COCOMO (Cont.)

COST DRIVERS PARAMETERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Parameter					
Required Software	0.75	0.88	1	1.15	1.4
Size of Project Database	NA	0.94		1.08	1.16
Complexity of The Project	0.7	0.85		1.15	1.3
Hardware Parameter					
Performance Restriction	NA	NA	1	1.11	1.3
Memory Restriction	NA	NA		1.06	1.21
virtual Machine Environment	NA	0.87		1.15	1.3
Required Turnabout Time	NA	0.94		1.07	1.15
Personnel Parameter					
Analysis Capability	1.46	1.19	1	0.86	0.71
Application Experience	1.29	1.13		0.91	0.82
Software Engineer Capability	1.42	1.17		0.86	0.7
Virtual Machine Experience	1.21	1.1		0.9	NA
Programming Experience	1.14	1.07		0.95	NA
Project Parameter					
Software Engineering Methods	1.24	1.1	1	0.91	0.82
Use of Software Tools	1.24	1.1		0.91	0.83
Development Time	1.23	1.08		1.04	1.1

The intermediate COCOMO (Cont.)

Example: For a given project was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development by considering developer having high application experience and very low experience in programming.

Ans:

Given the estimated size of the project is: 300 KLOC

Developer having highly application experience: 0.82 (as per above table)

Developer having very low experience in programming: 1.14(as per above table)

EAF = 0.82*1.14 = 0.9348

Effort (E) = $a*(KLOC)b$ *EAF = 3.0* (300)1.12 *0.9348 = 1668.07 MM

Scheduled Time (D) = $c*(E)d$ = 2.5* (1668.07)0.35 = 33.55 Months(M)

The Detailed COCOMO

It is the advanced model that estimates the software development effort like Intermediate COCOMO in each stage of the software development life cycle process.

COCOMO 1 Versus COCOMO 2

- COCOMO 1 Model
  - Also called as Basic COCOMO.
  - Estimating effort, cost, and schedule for software projects.
  - Give an **approximate estimate of the various parameters** of the project.
- COCOMO 2 Model
  - Whole software is divided into different modules.
  - Calculates the development time and effort taken as the total of the estimates of all the individual subsystems.

COCOMO I versus COCOMO II

COCOMO I

- COCOMO I is useful in the waterfall models of the software development cycle.
- It provides estimates of effort and schedule.
- This model is based upon the linear reuse formula.
- This model is also based upon the assumption of reasonably stable requirements.
- Effort equation’s exponent is determined by 3 development modes.
- Development begins with the requirements assigned to the software.
- Number of submodels in COCOMO I is 3 and 15 cost drivers are assigned
- Size of software stated in terms of Lines of code

COCOMO II

- COCOMO II is useful in non-sequential, rapid development and reuse models of software.
- It provides estimates that represent one standard deviation around the most likely estimate.
- This model is based upon the non linear reuse formula.
- This model is also based upon reuse model which looks at effort needed to understand and estimate.
- Effort equation’s exponent is determined by 5 scale factors.
- It follows a spiral type of development.
- In COCOMO II, Number of submodel are 4 and 17 cost drivers are assigned
- Size of software stated in terms of Object points, function points and lines of code

Boehm Model

- Maintenance based on economic models and principles.
- Closed loop cycle.
- Boehm proposed a maintenance cost as it is a part of COCOMO model.
- Boehm uses a quantity called Annual Change Traffic (ACT), which is defined as software product's source instruction which changes during a year either through add, delete or modify.

ACT = KLOCadded + KLOCdeleted / KLOCtotal

The annual maintenance effort (AME) in person-months is measured as:

AME = ACT * SDE

Thank you..

Unit- III
Reengineering

Mrs. Nisha Gautam,
School of Computing,
Indian Institute of Information Technology, Una.

Background

- Reengineering is the examination, analysis and restructuring of an existing software system to reconstitute it in a new form and the subsequent implementation of the new form.
- Goals of reengineering is to:
  - Understand the existing software system artifacts.
  - Improve the functionality and quality attributes of the system. Eg: evolvability, performance and reusability.
- Output of reengineering: Target system possesses better quality factors i.e. reliability, correctness, integrity, efficiency, maintainability, usability, flexibility, testability, interoperability, reusability and portability.

Background (Cont..)

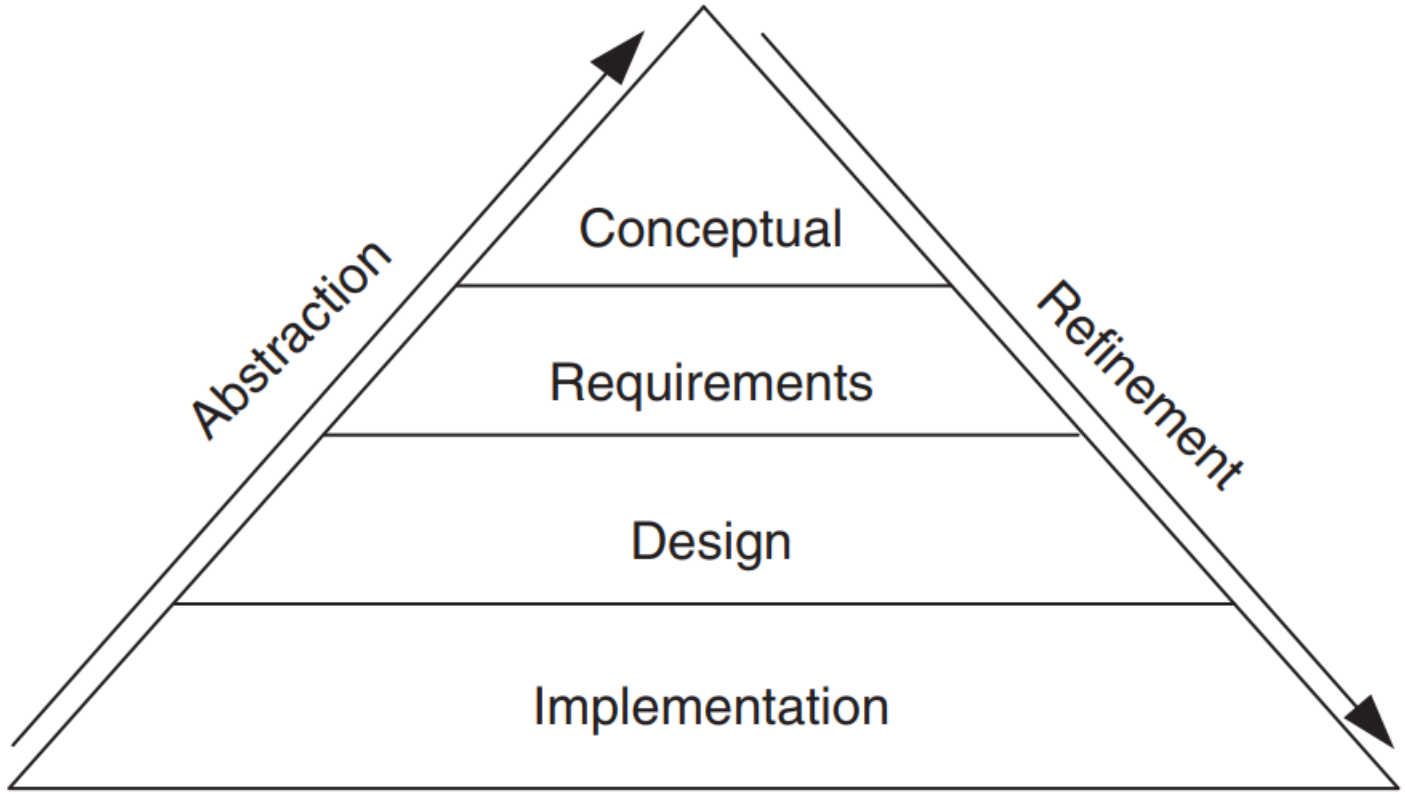
- Reengineering is done to convert an existing “bad” system into a “good” system.
- Risks involved in transformation:
  - Functionality issue
  - Quality factor
  - Benefits are not realized in a required time frame.
- Reengineering can be done by considering one or more objectives :
  - Improve maintainability : 2nd law i.e. increasing complexity.
  - Migrating to a new technology: 1st law i.e. continuing change.
  - Improve quality: 7th law i.e. declining quality which causes ripple effect.
  - Preparing for functional enhancements: 6th law i.e. continuing growth.

Reengineering

- A good comprehension may be useful in making plan for reengineering.
- Key concepts used in reengineering:
  - Abstraction: enable maintenance personnel to reduce the complexity to understand the system by:
    - Focusing on the significant information.
    - Hiding the irrelevant details.
  - Refinement : Reverse of abstraction.

- Principles:
  - **Principle of abstraction:** The representation of the system can be gradually increased by successively replacing the details with abstract information. By means of abstraction one can produce a view that focuses on the selected characteristics by hiding low level details.
  - **Principle of refinement:** The representation of the system is gradually decreased by successively replacing some aspects of the system with more details.
- Forward Engineering
- Reverse engineering: Involved steps:
  - Analyse the software to determine its components and their relationship
  - Represent the system at higher level of abstraction or in another form

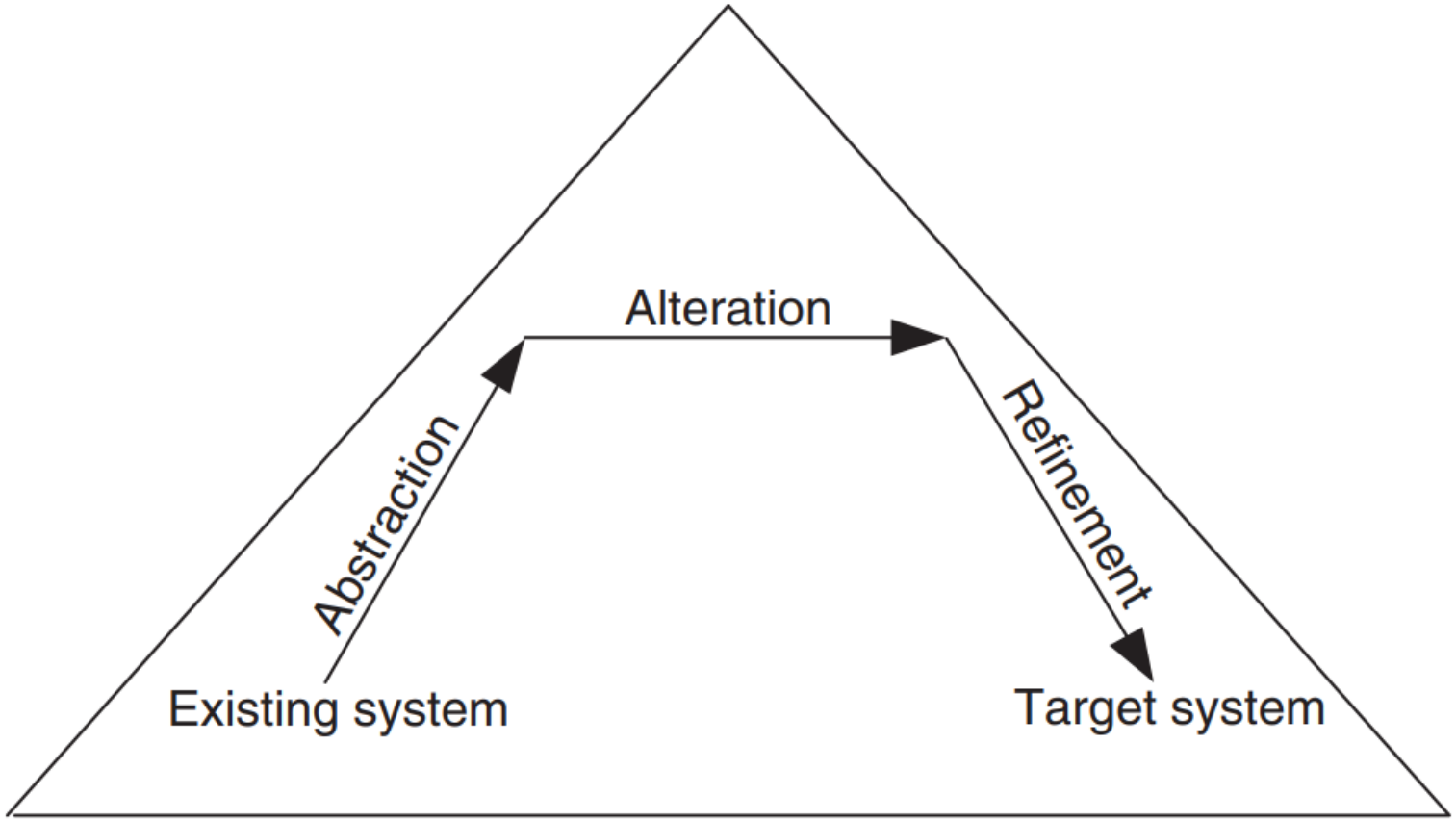
Levels of abstraction and refinement



Levels of abstraction and refinement (Cont..)

- Conceptual Level: highest level abstraction
  - Reason of existence.
- Requirement level:
  - Functional characteristics such as “what”.
- Design level: design refinement level
  - System characteristics such as “what and how”.
- Implementation level : lowest level of abstraction.
  - Level addresses “how” exactly the system is implemented.

Another principle



Another principle

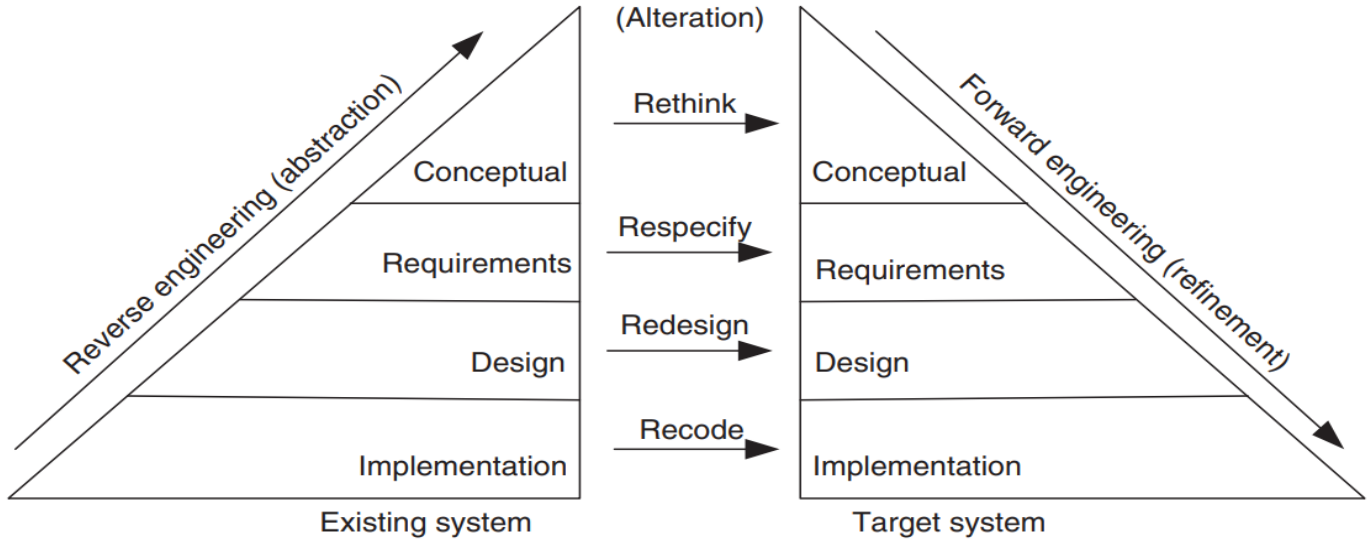
- Principle of alteration: The making of some changes to a system representation.
  - Doesn’t affect the degree of abstraction and it does not involve modification, deletion and addition of information.
- Non essential for reengineering.
- Alteration versus restructuring:
  - Restructuring is defined as the transformation from one representation to another at the same relative abstract level while preserving the subject system’s external behaviour.
  - Often used as a form of preventive maintenance.

Model for software reengineering

- Reengineering process accepts the existing system as input and produces the code of the renovated system.
- Process may be as straightforward as translating with the tool of the source code from a given language into another language.
- Reengineering process may be very complex as explained :
  - Recreate a design
  - Find the requirements of the system being reengineered;
  - Compare the existing requirements with the new ones;
  - Remove the requirements that are not needed in the renovated systems;
  - Make a new design of the desired system; and
  - Code the new system.

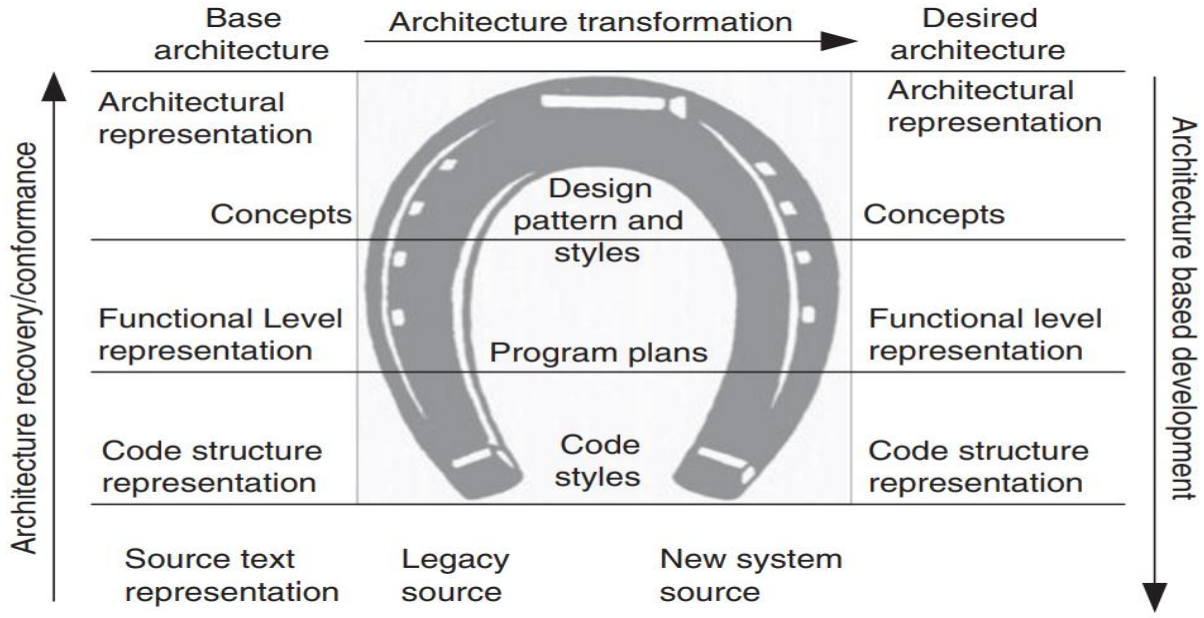
Model for software reengineering (Cont..)

- Author Eric J. Byrne depicts the processes for all abstraction levels on reengineering. The model suggests a sequence of three activities – reverse engineering, re-design and forward engineering- strongly founded in three principles namely abstraction, alteration and refinement respectively.



Model for software reengineering (Cont..)

- Anothor author Kazman et al. proposes a visual metaphor called horseshoe consists of three step architectural reengineering process.
  - Step I: Extracting the architecture from the source code
  - Step II: Architectural transformation towards the target architecture.
  - Step III: Generation of the new architecture by means of refinement.



Model for software reengineering (Cont..)

- Definitions of reengineering:
  - Software reengineering is the analysis and alteration of an operational system to represent it in a new form and to obtain a new implementation from a new form.
  - Reengineering of a software system is a process for creating a new software form the existing software so that the new system is better than the original system in some ways.
  - Reengineering is a activity that (i) improves the comprehension of the software system or (ii) raises the quality levels of the software, namely, performance, reusability and maintainability.

Model for software reengineering (Cont..)

- Reengineering = Reverse engineering + delta + Forward engineering.
  - Reverse Engineering : activity to create an easier to understand and more abstract form of the system.
  - Delta : alterations made.
  - Forward Engineering : process of moving from the high level abstraction and logical, implementation independent design to the physical implementation of the system.
- Rehosting versus reengineering

Model for software reengineering (Cont..)

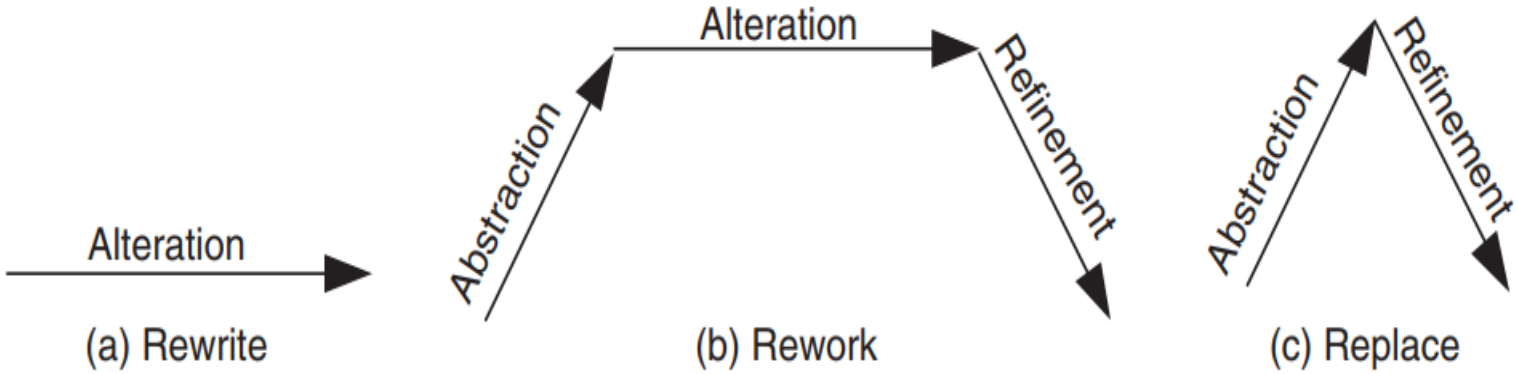
- Based on the **types of changes** required, system characteristics are divided into groups:
  - Recode
  - Redesign
  - Respecify
  - Rethink
- Modifications performed on any level may not affect the higher level of abstraction but affects the lower level of abstraction (if the highest level is implementation level).

OR

The modifications performed on any level may not affect the lower level of abstraction but affects the higher level of abstraction. (of the HLA is conceptual level).

Model for software reengineering (Cont..)

- Software reengineering strategies:
  a) Rewrite
  b) Rework
  c) Replace



Model for software reengineering (Cont..)

- **Reengineering process variations:** If A is the abstraction level of representation of the system to be reengineered and the plan is to make a B type, can I use strategy C?

		Reengineering Strategy		
Starting Abstraction Level	Type Change	Rewrite	Rework	Replace
Implementation level	Recode	Yes	Yes	Yes
	Redesign	Bad	Yes	Yes
	Respecify	Bad	Yes	Yes
	Rethink	Bad	Yes*	Yes*
Design level	Recode	No	No	No
	Redesign	Yes	Yes	Yes
	Respecify	Bad	Yes	Yes
	Rethink	Bad	Yes*	Yes*
Requirement level	Recode	No	No	No
	Redesign	No	No	No
	Respecify	Yes	Yes	Yes
	Rethink	Bad	Yes*	Yes*
Conceptual level	Recode	No	No	No
	Redesign	No	No	No
	Respecify	No	No	No
	Rethink	Yes	Yes*	Yes*

Model for software reengineering (Cont..)

Yes—One can produce a target system.
Yes*—Same as Yes, but the starting degree of abstraction is lower than the uppermost degree of abstraction within the conceptual abstraction level.
No—One cannot start at abstraction level *A*, make *B* type of changes by using strategy *C*, because the starting abstraction level is higher than the abstraction level required by the particular type of change.
Bad—A target system can be created, but the likelihood of achieving a good result is low.

Reengineering Process

- An **ordered set of activities designed to perform a specific task** is called a process. And for the ease of understanding and communication, processes are described by means of process models.
- In a reengineering process, the concept of **approach impacts the overall process structure.** There are **five major considered approaches** : Big Bang approach, Incremental approach, Partial approach, Iterative approach, Evolutionary approach.
- While selecting a particular approach, several considerations have been made:
  - Objective of the project.
  - Availability of resources.
  - The present state of the system being reengineered; and
  - The risks in the reengineering project.

Reengineering Process (Cont..)

- The five approaches are different in two aspects:
  - The extent of reengineering performed.
  - The rate of substitution of the operational system with the new one.
- Big Bang Approach:
  - Approach replaces the whole system at once i.e. reengineering cannot be done in parts.
  - Once a reengineering effort is initiated, it is continued until all objectives of the project are achieved and the target system is constructed.
  - Eg. :- if there is a need to move to a different system architecture.
  - Advantage: the system is brought into its new environment all at once.
  - Disadvantage:
    - reengineering project will become monolithic task,
    - consumes too much resources at once and require long stretch of time.

Reengineering Process (Cont..)

- Incremental Approach:
  - Reengineering gradually one step closer to the target system at a time.
  - Several interim versions are produced and released.
  - The desired system is said to be generated after all the project goals are achieved.
  - Advantages:
    - locating errors becomes easier,
    - customers can easily track progress,
    - lower risk as compared to big bang approach.
  - Disadvantages:
    - With multiple interim versions and **their careful version controls**, the incremental approach takes more time to complete.
    - **Even if there is a need, the entire architecture of the system cannot be changed**.

Reengineering Process (Cont..)

- Partial approach:
  - Only a part of the system is reengineered.
  - **Three steps:**
    - The existing system is portioned into **two parts**.
    - Reengineering work is **performed using either the big bang approach or incremental approach**.
    - **Integrate** the two parts to make up the new system.
  - **Advantages:**
    - **Reducing the scope of reengineering**
    - Takes **less time and cost** as compared to big bang or incremental approach.
  - **Disadvantage:**
    - Modifications are not performed to the interface between the portion modified and the portion not modified.

Reengineering Process (Cont..)

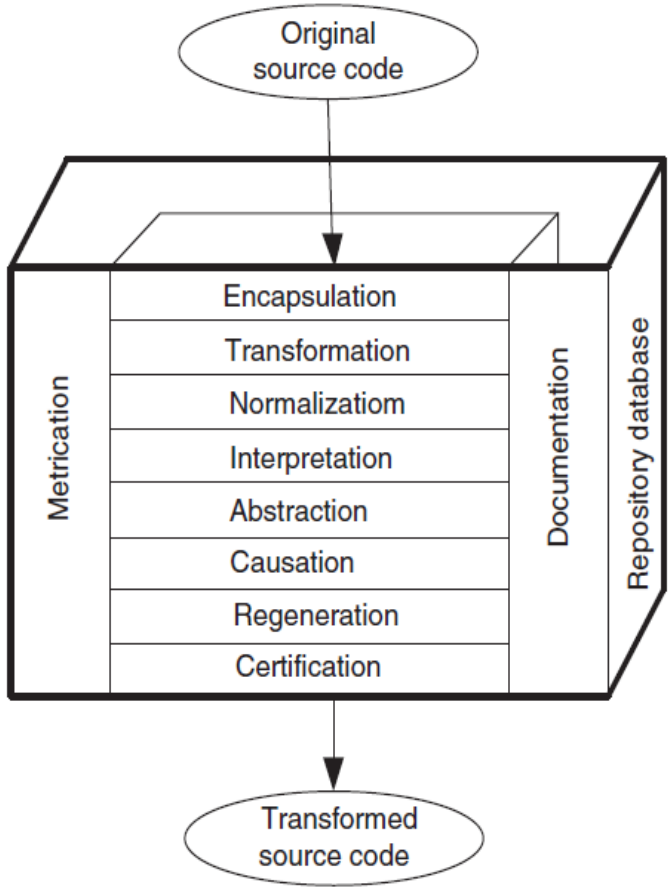
- Iterative approach:
  - Reengineering process is applied on the source code of a few procedures at a time, with each reengineering operation lasting for a short time.
  - **Process is repeatedly executed** on different components in different stage.
  - **Four components are:**
    - Old component not reengineered
    - Component currently being reengineered
    - Components already reengineered
    - New component added to the system
  - **Advantages:**
    - guarantees the continued operations during reengineering process.
    - Familiarity of maintainers and the users with the system is preserved.
  - **Disadvantages:**
    - Need to keep a track of all four types of components during reengineering process.
    - Both the old and the newly reengineered components need to be maintained.

Reengineering Process (Cont..)

- Evolutionary approach:
  - Similar to incremental approach but in evolutionary approach components of the original system are substituted with reengineered components.
  - **Advantages**:
    - The resulting design is more cohesive.
    - The scope of individual components is reduced.
  - **Disadvantage**:
    - All the functions with much similarities must be first identified throughout the operational system and then those functions are refined as one unit in the new system.

Source code reengineering reference model (SCORE/RM)

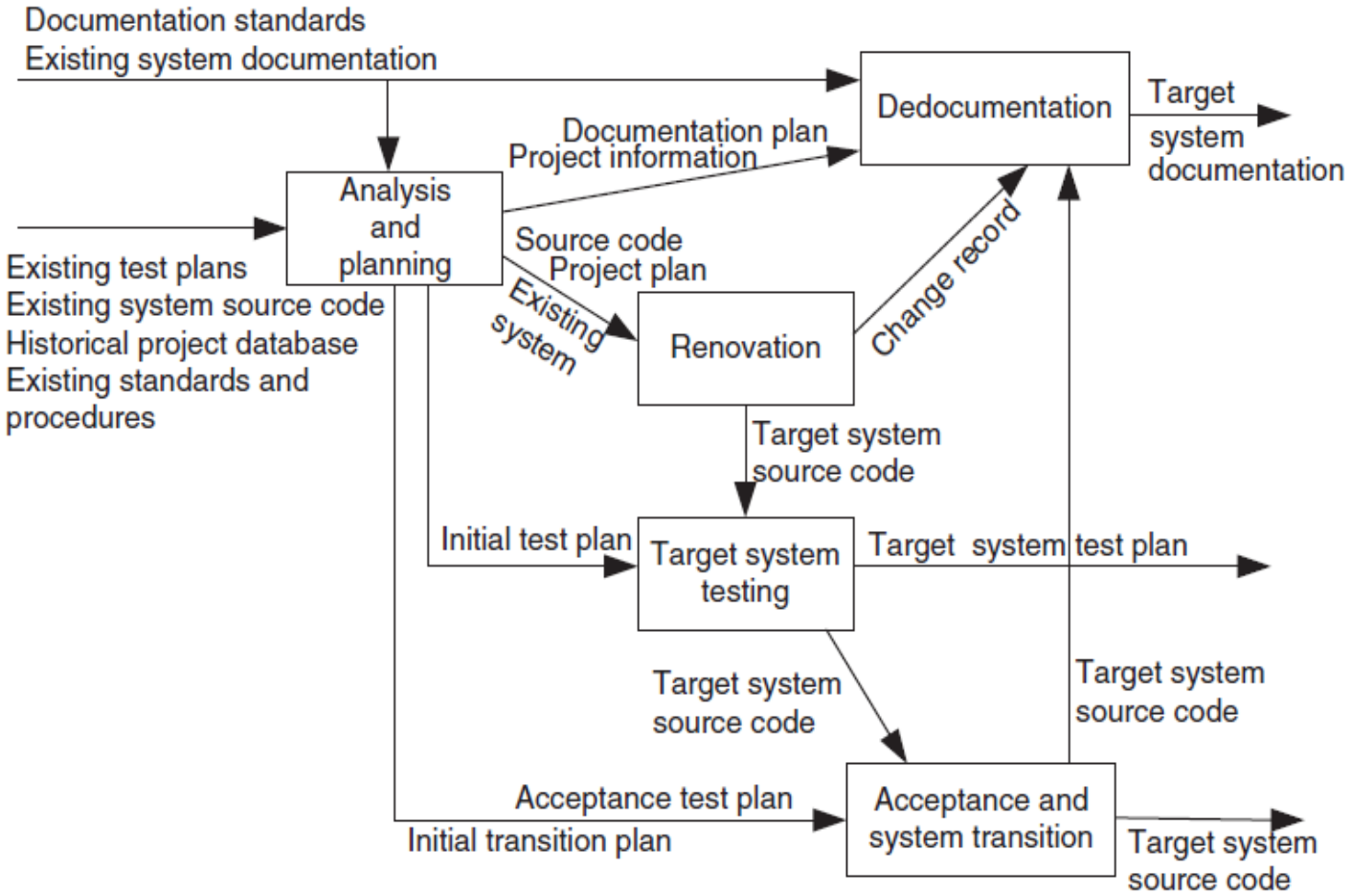
- **Four kinds of elements:** function, documentation, repository database, and metrication.
- The top six of the eight layers shown in Figure constitute a process for **reverse engineering**, and the bottom three layers constitute a process for **Forward engineering**. Both the processes include causation, because it represents the derivation of requirements specification for the software.
- **Metrication** : Relevant software matrices before and after executing a same layer.
- **Documentation:** The specification, constraints, and implementation details of both the old and the new versions of the software are described in the documentation element.
- **Repository**: metrication, documentation, and both the old and the new source codes.



Source code reengineering reference model (SCORE/RM) (cont..)

- Functions are discussed below:
  - **Encapsulation** : Reference baseline has been created from the given source code and identify the software version for reengineering. We consider functions such as configuration management, analysis, parsing and text generation.
  - **Transformation** : To make the code structured, its control flow is changed. The functions are rationalization of control flow, isolation and procedural granularity.
  - **Normalization**: The involved functions are data reduced and data represented.
  - **Interpretation**: Functionalization and program reading.
  - **Abstraction:** Object identification and object interpretation.
  - **Causation:** Specification of actions, specification of constraints and modification of specification.
  - **Regeneration :** generation of designs, generation of codes and test generation.
  - **Certification:** verification and validation, conformance.

Phase reengineering model



Need of reverse engineering

- The original programmers have left the organization.
- Implementation language has become obsolete.
- Insufficient documentation.
- Business relies on software, which many cannot understand.
- Lack of access to all the source code.
- The system requires adaptations and/or enhancements.
- System does not operated as expected.

Code reverse engineering

- Reverse engineering can be defined as the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system.
- Reverse engineering is a process to
  - Identify the components of an operational software.
  - Identify the relationship among those components.
  - Represent the system at higher level of abstraction.

Reverse engineering (Cont..)

- Reverse engineering comprises of
  - **High level reverse engineering** :creating abstractions of source code in a form of design, architecture and documentation.
  - **Low level reverse engineering**: creating source code from object code or assembly code.
- Reverse engineering is performed to achieve **two key objectives**:
  - Redocumentation of artifacts.
  - Design recovery.
- **Objectives of reverse engineering**:
  - Generating alternative views.
  - Recovering lost information.
  - Synthesizing higher levels of abstraction.
  - Detecting side effects.
  - Facilitating reuse.
  - Coping with complexities.

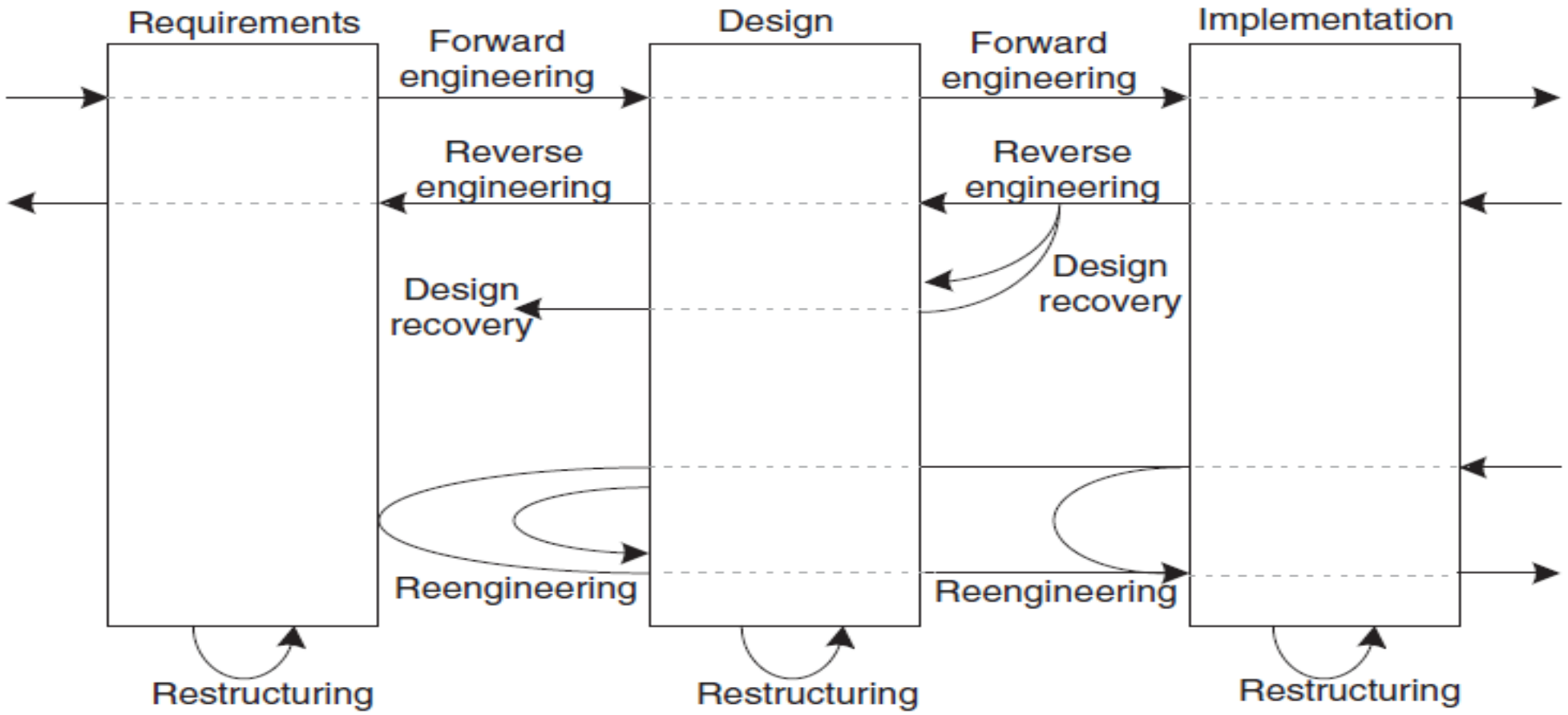
Reverse engineering (Cont..)

- **Six key steps of reverse engineering**:
  1. partition source code into units;
  2. describe the meanings of those units and identify the functional units;
  3. create the input and output schematics of the units identified before;
  4. describe the connected units;
  5. describe the system application; and
  6. create an internal structure of the system.

Reverse engineering is applied in the following problem areas:

- redocumenting programs ;
- identifying reusable assets ;
- discovering design architectures ;
- recovering design patterns ;
- building traceability between code and documentation ;
- finding objects in procedural programs ;
- deriving conceptual data models ;
- detecting duplications and clones ;
- cleaning up code smells ;
- aspect-oriented software development ;
- computing change impact ;
- transforming binary code into source code;
- redesigning user interfaces ;
- parallelizing largely sequential programs;
- translating a program to another language;
- migrating data ;
- extracting business rules ;
- wrapping legacy code ;
- auditing security and vulnerability ;and
- extracting protocols of network applications .

Relationship between reengineering and reverse engineering



Techniques used for reverse engineering

- Purpose: Fact finding and information gathering.
- Techniques:
  - Lexical analysis : Generators like lex, flex.
  - Syntactic analysis : use parsers.
  - Control flow analysis : uses CFGs.
  - Data flow analysis
  - Program Slicing
  - Visualization
  - Program metrics

- **Lexical Analysis:**
  - Decomposition of information into sequence of characteristics.
  - Uses Regular Grammer and Expressions etc.
  - Uses generators like lex, flex.
- **Syntactic Analysis:**
  - Logical meaning of any sentence/ phrase.
  - Uses parsing/ syntax analysis and grammers to resolve involved issues.
  - Top down parsing
  - Bottom up parsing: shift reduce parsers, LR(0), SLR(1), LALR, CLR.
- **Control flow analysis:**
  - Static code analysis is used for CFA
  - Uses call graphs, dead code elimination, duplicate code.
- **Program Slicing :**
  - Can be done while debugging.
  - Also uses forward slicing and backward slicing.

- **Visualization:**
  - Visualization verses representation
  - **Key attributes of representation:** Focus on individuality, Distinctive appearance, High information content, low visual complexity, scalability of visual content, flexibility for integration into visualizations, suitability for automation.
  - **Requirements of visualization:** Simple navigation, high information content, Low visual complexity, varying levels of details, resilience to change, effective visual metaphors, friendly user interface, Integration with other information sources, good use of interactions, suitability for automation.
- **Program metrics :**
  - consider complexity matrices.

  C(P) = (fan-in * fan-out)^2
  - Six performance matrices: Weighted methods per class, response for the class, Lack of cohesion in methods, coupling between object class, depth of inheritance tree and the number of children.

Reverse Engineering tools

- Ada SDA (System Dependency Analyzer)
- Code Crawler
- DMS (Design maintenance system) Toolkit
- FermaT
- GXL (Graph eXchange Language)
- IDA Pro Disassembler and Debugger
- Hex-Rays Decompiler
- Imagix 4D
- IRAP (Input-output Reengineering and Problem Crafting)
- JAD (Java Decompiler)
- ManSART
- RE-Analyzer
- And many more…

Thank you..

Unit 4 - Software reuse and Reuse landscape

Mrs. Nisha Gautam
School of Computing
IIIT Una.

Software Reuse

- Software reuse means using existing assets in the development of a new system.
- Software reuse involves two main activities: software development with reuse and software development for reuse.
- Reusable assets can be both **reusable artifacts** and **software knowledge.**
- Four types of reusable artifacts as follows :
  - **Data reuse** : Standardization of data formats
  - **Architectural reuse**:
    - Set of generic **design styles** about the logical structure of software; and
    - A set of **functional elements** and reuse those elements in new systems.
  - **Design reuse:** Reuse of abstract design i.e. meet the application requirements.
  - **Program reuse:** Reusing executable code.

Software Reuse (Contd.)

- The **reusability property of a software asset** indicates the degree to which the asset can be reused in another project.
- For a software component to be reusable, it needs to **exhibit the following properties that directly encourage its use** in similar situations.
  - **Environmental independence:** Reused irrespective of the environment from which they were originally captured.
  - **High cohesion:** Subsystems cooperate with each other to achieve a single objective.
  - **Low coupling:** Less impact on other components makes it more usable.
  - **Adaptability:** To run in any new environment.
  - **Understandability:** Easily comprehended, so the programmers can quickly make decisions about its reuse.
  - **Reliability:** Consistently perform its intended function without degradation or failure.
  - **Portability:** Usable in different environment.

Benefits of Reuse

- Economic benefits
- Increased reliability: Reduces effort in design & implementation.
- Reduced process risks: Reuse fault free components.
- Increased Productivity : Reduced cost and time.
- Compliance with standards
- Accelerated development
- Improved maintainability
- Less maintenance effort and time: modified during reusability.

Reuse Models

- The organization can select one or more reuse models that best meet their business objectives, engineering realities, and management styles.
- Reuse Models are:
  - **Proactive approach**
    - System is designed for all receiving variations.
    - Process to create reusable software assets (RSA) called domain engineering.
    - This approach might be adopted by organizations that can accurately estimate the long-term requirements for their product line.
  - **Reactive approach**
    - Reusable assets are developed if a reuse opportunity arises.
    - This approach works, if
      - It is **difficult to perform long-term predictions** of requirements for product variations; or
      - An organization needs to **maintain an aggressive production schedule with not much resources to develop** reusable assets.
  - **Extractive approach**
    - Falls in between proactive and reactive approach and accumulated both artifacts and experiences in a domain.

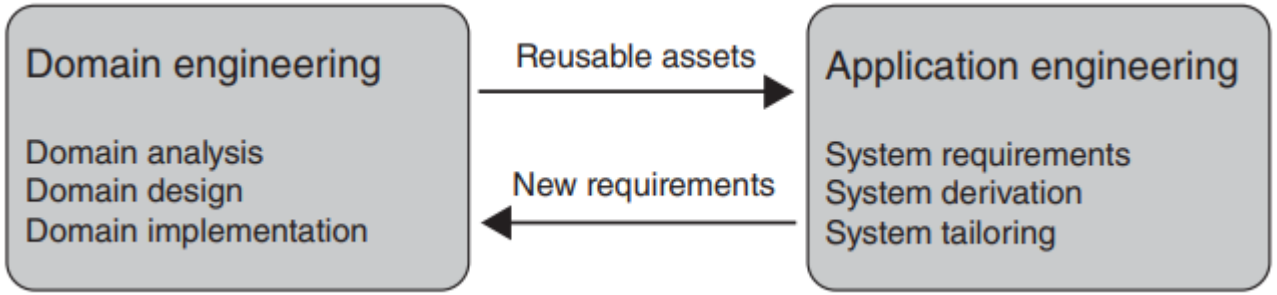
Factors influencing Reuse

- **Managerial:**
  - Less management support:
    - need years of investment before it pays off
    - involves changes in organization funding and management structure
- **Legal:**
  - Proprietary and copyright issues, liabilities and responsibilities of reusable software, and contractual requirements involving reuse.
- **Economic:**
  - Artifacts need to be reused more than 13 times to recoup the extra cost of developing reusable components.
- **Technical:**
  - **Received much attention from the researchers** actively engaged in library development, object-oriented development paradigm and domain engineering.
  - **Collect library assets** in a number of ways: reengineering, design and build new assets, and purchase assets from other sources.
  - **Certification** process through verification and testing.

Success Factors of Reuse

- Develop software with the product line approach.
- Develop software architectures to standardize data formats and product interfaces.
- Develop generic software architectures for product lines.
- Incorporate off-the-shelf components.
- Perform domain modeling of reusable components.
- Follow a software reuse methodology and measurement process.
- Ensure that management understands reuse issues at technical and nontechnical levels.
- Support reuse by means of tools and methods.
- Support reuse by placing reuse advocates in senior management.
- Practice reusing requirements and design in addition to reusing code.

Domain Engineering



- Refers to a “development-for-reuse” process to create RSA.
- Domain engineering, is the entire process of reusing domain knowledge in the production of new software systems.
- Domain engineering consists of:
  - Domain Analysis:
    - Domain Analysis consists of:
      - Identify the family of products to be constructed;
      - Determine the variable and common features in the family of products
      - Develop the specifications of the product family.
    - Eg : Feature-oriented domain analysis (FODA) method : The method describes a process for domain analysis to discover, analyze, and document commonality and differences within a domain.

Domain Engineering

- Domain Design :
  - Develop a generic software architecture for the family of **products under consideration**;
  - develop a **plan to create individual systems** based on reusable assets.
  - Consider both **functional and non-functional requirements**.
  - **Limitation for this phase is context use** in terms of its applicability or its insufficiency in reuse.
  - **Objective of domain** design is to satisfy as many domain requirements as possible while retaining the flexibility offered by the developed feature model.
- Domain Implementation
  - **Identify reusable components** based on the outcome of domain analysis;
  - **Acquire and create reusable assets** by applying the domain knowledge acquired in the process of domain analysis and the generic software architecture constructed in the domain design phase;
  - **Catalogue the reusable assets into a component library**.

Domain Engineering Approaches

- Draco:
  - First prototype in domain engineering.
  - Based on **transformation technology.**
  - **Domain specific language, optimized transformations**
  - Very complex to apply in production environment.
- DARE:
  - DARE is both a Domain Analysis method and a tool suite supporting the method.
  - DARE Includes lexical analysis tools for extracting domain vocabulary: code, documents, and expert knowledge.
  - The generic architectures, feature tables, and facet tables, and all models and information are stored in a domain book [main work products of the DARE].
- FAST
  - FAST to develop telecommunication infrastructure.
  - Three sub-processes constitute FAST:
    - domain qualification (DQ-worthy for investment is identified),
    - domain engineering (DE-enables the development of product line environments and assets), and
    - application engineering (AE- for developing products rapidly).

Domain Engineering Approaches (Contd.)

- FORM
  - FORM finds commonalities and differences in a product line in terms of features, and uses those findings to develop architectures and components for product lines.
  - Two processes are key to FORM: asset development and product development.
- KobrA
  - Method for component-based application development
  - Two main activities in KobrA are: framework engineering and application engineering.
    - Framework engineering, one makes a common framework that manifests all variations in products making up the family.
    - Application engineering is applied on the framework to build specific applications.

Domain Engineering Approaches (Contd.)

- PLUS
  - In PLUS we do:
    - **For requirements analysis activities, use-case modeling and feature modeling**;
    - Mechanisms to model the static aspects, dynamic interactions, state machines, and class dependency for product lines; and
- PuLSE (product line software engineering)
  - Developed to **enable the conceptualization and deployment of software product lines for large enterprises.**
  - PuLSE methodology comprises three key elements:
    - **The deployment phases**: describe activities for initialization, construction of infrastructure, usage of infrastructure, and management and evolution
    - **The technical components**: how to operationalize the development.
    - **The support components:** guidelines enabling better evolution, adaptation, and deployment.

Domain Engineering Approaches (Contd.)

- RSEB
  - use-case-driven reuse method based on UML
  - designed to facilitate both asset reuse and the development of reusable software.
  - RSEB supports both domain engineering and application engineering.
- Coala
  - Koala is a language to **describe architectures for product lines**
  - developed at Philips Corporation
  - To **support product variations, diversity interfaces and switches are provided** in Koala
- Organization Domain modelling (ODM): reuse library frameworks or knowledge based reuse model.
- Capture tool : hypertext based tool using navigation.
- DSSA (Domain- specific software architecture) : central role of software architecture.

Reuse Capability

Reuse Landscape

- Although reuse is often simply thought of as the **reuse of system components,** there are many different approaches to reuse that may be used.
- **Possible range of reuse** varies from simple functions to complete application systems
- The reuse landscape covers the range of possible reuse techniques.

Reuse approaches

Design patterns	Generic abstractions that occur across applications are represented as design patterns that show abstract and concrete objects and interactions.
Component-based development	Systems are developed by integrating components (collections of objects) that conform to component-model standards. This is covered in Chapter 19.
Application frameworks	Collections of abstract and concrete classes that can be adapted and extended to create application systems.
Legacy system wrapping	Legacy systems (see Chapter 2) that can be 'wrapped' by defining a set of interfaces and providing access to these legacy systems through these interfaces.
Service-oriented systems	Systems are developed by linking shared services that may be externally provided.

Reuse approaches

Application product lines	An application type is generalised around a common architecture so that it can be adapted in different ways for different customers.
COTS integration	Systems are developed by integrating existing application systems.
Configurable vertical applications	A generic system is designed so that it can be configured to the needs of specific system customers.
Program libraries	Class and function libraries implementing commonly-used abstractions are available for reuse.
Program generators	A generator system embeds knowledge of a particular types of application and can generate systems or system fragments in that domain.
Aspect-oriented software development	Shared components are woven into an application at different places when the program is compiled.

Concept Reuse

- **Limit the reuse opportunities,** when you reuse program or design components, you have to follow the design decisions made by the original developer of the component.
- So, **abstract form of reuse** is concept reuse when a particular approach is described in an implementation independent way and an implementation is then developed.
- The two main approaches to concept reuse are:
  - Design patterns;
  - Generative programming

Design patterns

- A design pattern is a way of **reusing abstract knowledge** about a problem and its solution.
- A pattern is a **description of the problem** and the essence of its solution.
- Patterns **rely on object characteristics** such as inheritance and polymorphism.

Pattern elements

- **Name**
  - A meaningful pattern identifier.
- **Problem description**
- **Solution description**
  - Not a concrete design but a template for a design solution that can be instantiated in different ways.
- **Consequences**
  - The results and trade-offs of applying the pattern

The observer pattern

- **Name**
  - Observer.
- **Description**
  - Separates the display of object state from the object itself.
- **Problem description**
  - Used when multiple displays of state are needed.
- **Solution description**
  - See slide with UML description.
- **Consequences**
  - Optimizations to enhance display performance are impractical

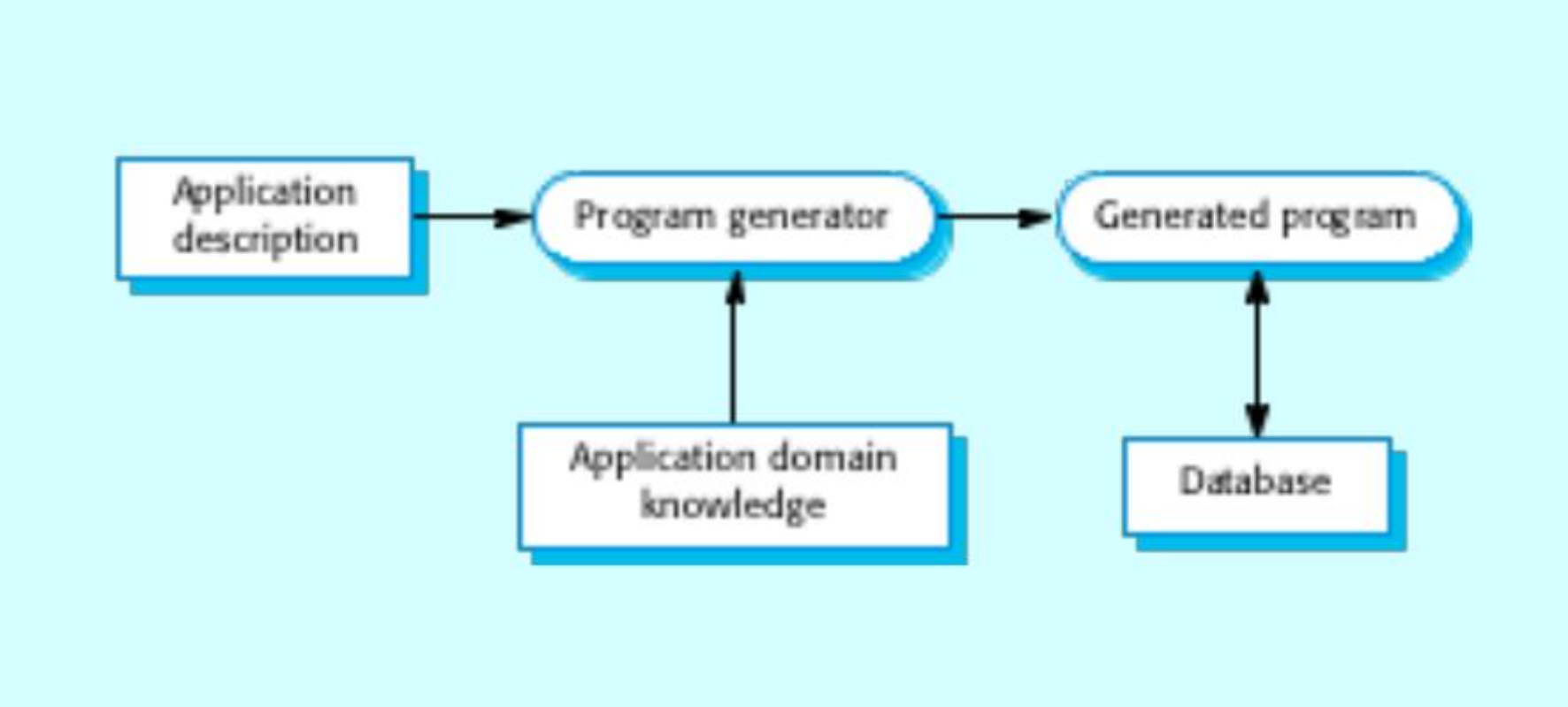
Generator-based Reuse

- Program generators involve the **reuse of standard patterns and algorithms.**
- These are embedded in the generator and parameterized by user commands. A program is then automatically generated.
- Generator-based reuse is possible **when domain abstractions and their mapping to executable code can be identified.**
- A domain specific language is used to compose and control these abstractions.

Types of program generator

- Types of program generator
  - **Application generators** for business data processing;
  - **Parser and lexical analyzer generators** for language processing;
- **Generator-based reuse is very cost-effective** but its applicability is limited to a relatively small number of application domains.
- It is easier for end-users to develop programs using generators compared to other component-based approaches to reuse.

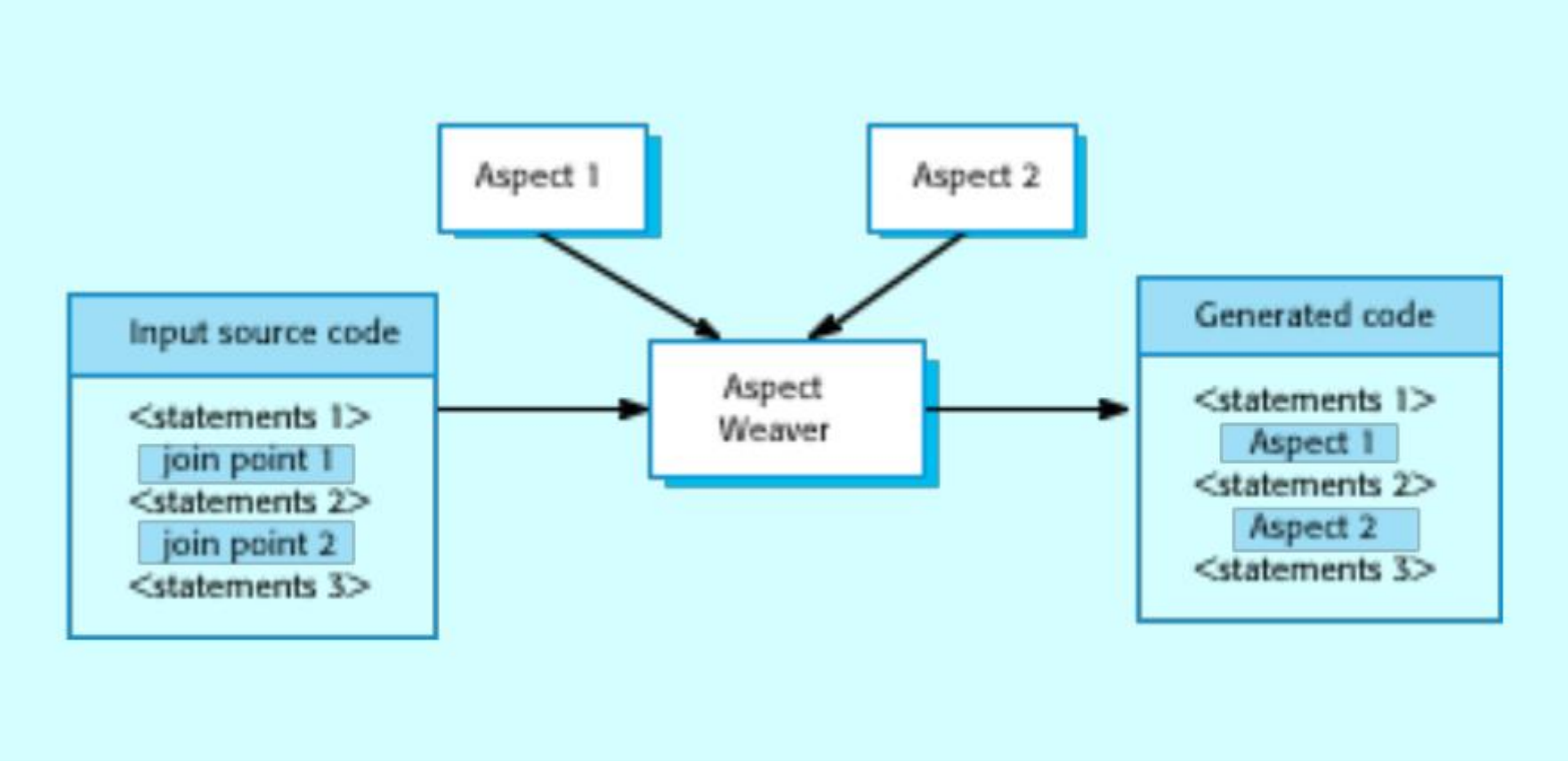
Reuse through program generation



Aspect-oriented development

- Aspect-oriented development addresses a major software engineering problem - **the separation of concerns.**
- Concerns are often not simply associated with application functionality but are cross-cutting - e.g. all components may monitor their own operation, all components may have to maintain security, etc.
- **Cross-cutting concerns are implemented as aspects** and are dynamically woven into a program. The concern code is reuse and the new system is generated by the aspect weaver.

Aspect-oriented development



THANK YOU..

Unit 5 - Software Metrics

Mrs. Nisha Gautam
School of Computing
IIIT Una.

Software metrics and measurement

- Software is measured to:
  - **Establish the quality** of the current product.
  - To **predict future qualities** of the product.
  - **To improve the quality** of the product or process.
  - To **determine the state of the project** in relation to budget and schedule.
- So, **measurement** is an indication of the size, quantity, amount and dimension of particular attribute of the product or process. Example: number of errors.
  - **Software measurement activities**: Formation, Collection, Analysis, Interpretation, feedback.
- A **metric is a measurement** of the degree of any attribute belong to a system/ software/process.

Cont..

- **Software measurement gives rise to software metrics.**
- **Metrics are related to four functions of management:**
  - Planning
  - Organizing
  - Controlling
  - Improving
- **Metric classification:** Product and process metric
  - **Product metrics:** State of the product, tracking risks , discovering potential problem areas.
  - **Process metrics:** focuses on improving the long term process of the team.

Goal Question metric (GQM)

- Steps:
  - **Develop a set of goals** :
    - **Purpose:** the process/metric is characterized/evaluated/understood in order to understand/improve it.
    - **Perspective:** the [cost/defects/changes] are examined from the point of view of [customer, manager, developer].
    - **Environment:** The environment in which measurement takes place is evaluated in terms of people, process, problem factor, tools and constraints.
  - Develop a **set of questions** that characterize the goals.
  - Specify the **metrics needed to answer the questions.**
  - Develop mechanisms for **data collection and analysis.**
  - **Validate and analyze** the data.

Goal Question metric (GQM)[Cont..]

- **Analyze in a post mortem fashion**: Confirmation to the goals.
- Provide **feedback to stakeholders**.

Six sigma

- Introduced by Robert Galvin in 1987 and addresses the issues related to reduce time, defects and variability.
- **Produces effective product 99.9996% of the time, allowing 3.4 errors/ million opportunities, improves customer loyalty, improves employee morale.**
- **Top-down methodology** or strategy to accelerate improvements in the software process and software product quality.
- **Uses analysis tool and product metrics to evaluate the software process** and software product quality.

Six sigma (Cont..)

- **Sub-methodologies:** These are the most popular framework used in DFSS (Design of Six Sigma) projects.
  - **DMAIC:** Define requirements, Measure performance, Analyze relationships, Improve performance, Control performance.
  - **DMADV:** Define requirements, Measure performance, Analyze relationship, Design solution, Verify functionality.
    - Works on feedback

Six sigma (Cont..)

Define	Measure	Analyse	Improve	Control
Benchmark	7 basic tools	cause & effect diagrams	Design of experiments	Statistical control
Baseline contract/charter	Defect metrics	Failure mode & effects analysis	Modelling	Control charts
Kano model	Data collection forms	Decision & risk analysis	Tolerance	Time series methods
Voice of the costumer	Sampling techniques	Statistical inférence	Robust design	procedural adherence performance
Voice of the business		Control charts		Preventive activities
Quality function deployment		Capability		
Process flow map		Reliability analysis		
Project management		Root cause analysis		
"Management by fact"		System thinking		

Metrics in project estimation

- Source Byte size
- Source Line of Code

  Physical SLOC = Lines of Source code + comments + blank lines

  Logical SLOC = Number of lines of functional code (“statements”)
  - K-LOC, K-DLOC, K-SLOC, M-LOC, G-LOC, T-LOC
- Function Pointers
  - Five categories: internal logical files, External Interface files, External inputs, external outputs, external inquiries.
- GUI features.

Software testing metrics

- Need of test metrics:
  - Take decision for the next phase.
  - Evidence to claim or prediction.
  - Understand the type of improvement required.
- Quantitative measures used to estimate the progress, quality, productivity and health of the software testing process.
- **Goal:** Improve efficiency and effectiveness for better decision making.
- **Example:** Total number of defects.
- **Types of test metrics:**
  - **Process metrics:** Process efficiency.
  - **Product metrics:** quality of products.
  - **Project metrics:** efficiency of project team, tool used by members.

People metrics

- Also called **personnel metrics** and helpful in **assisting appropriate allocation of resources among various project activities.**
- **Categories of people metrics:** Staffing- staffing metrics, Retention- retention metrics, Training and development, Recruiting- recruiting metrics.
- The **goal of the people metrics** is to keep staff happy, motivated and focused on the task at hand.
- These metrics are:
  - **Programming experience metrics:** Programming language experience, Development methods experience, management experience.
  - **Communication level metrics:** Teamwork experience, communication hardware software level, personal availability.
  - **Productivity metrics:** size productivity, productivity statistics, quality vs. productivity.
  - **Team structure metrics:** hierarchy metrics, team stability metrics.

Process metrics

- **Measure the development process that creates a body of software.** Example: time length of the software creation [Bohem, COCOMO], boat's COPMO [prediction about the need for additional effort on large projects].
- Types of process metrics:
  - **Static process metrics:** related to defined process. Example: no. of types of roles, types of artifacts etc.
  - **Dynamic process metrics:** related to the properties of process performance. Example: no. of activities performed, no. of artifacts created.
  - **Process evolution metrics:** related to the process of making changes over a period of time. Example: ho many iterations are there within the process.

- Manual test metrics Classification: Base metrics (given or collected) and calculated metrics.
- Test metrics life cycle
  - **Analysis:** identification of the metrics, Define the identified QA metrics.
  - **Communication:** Explain the need of metrics to stakeholder or testing team, educate team about data points.
  - **Evaluate:** capture and verify the data, Calculating the metrics value using the data captured.
  - **Report:** report with effective conclusion and distribute among related stakeholders, take the feedback.

How to calculate test metrics

Sr#	Steps to test metrics	Example
1	Identify the key software testing processes to be measured	Testing progress tracking process
2	In this Step, the tester uses the data as a baseline to define the metrics	The number of test cases planned to be executed per day
3	Determination of the information to be followed, a frequency of tracking and the person responsible	The actual test execution per day will be captured by the test manager at the end of the day
4	Effective calculation, management, and interpretation of the defined metrics	The actual test cases executed per day
5	Identify the areas of improvement depending on the interpretation of defined metrics	The Test Case execution falls below the goal set, we need to investigate the reason and suggest the improvement measures

Test metrics glossary

- **Rework Effort Ratio =** (Actual rework efforts spent in that phase/ total actual efforts spent in that phase) X 100
- **Requirement Creep =** (Total number of requirements added/No of initial requirements)X100
- **Schedule Variance =** (Actual Date of Delivery **–** Planned Date of Delivery)
- **Cost of finding a defect in testing =** (Total effort spent on testing/ defects found in testing)
- **Schedule slippage =** (Actual end date – Estimated end date) / (Planned End Date – Planned Start Date) X 100
- **Passed Test Cases Percentage** = (Number of Passed Tests/Total number of tests executed) X 100
- **Failed Test Cases Percentage** = (Number of Failed Tests/Total number of tests executed) X 100
- **Blocked Test Cases Percentage** = (Number of Blocked Tests/Total number of tests executed) X 100
- **Fixed Defects Percentage** = (Defects Fixed/Defects Reported) X 100

Test metrics glossary

- **Accepted Defects Percentage** = (Defects Accepted as Valid by Dev Team /Total Defects Reported) X 100
- **Defects Deferred Percentage** = (Defects deferred for future releases /Total Defects Reported) X 100
- **Critical Defects Percentage** = (Critical Defects / Total Defects Reported) X 100
- **Average time for a development team to repair defects** = (Total time taken for bugfixes/Number of bugs)
- **Number of tests run per time period** = Number of tests run/Total time
- **Test design efficiency** = Number of tests designed /Total time
- **Test review efficiency** = Number of tests reviewed /Total time
- **Bug find rote or Number of defects per test hour** = Total number of defects/Total number of test hours

Metrics for software maintenance

- When development of a software product is complete and it is released to the market, then it enters the maintenance phase of its lifecycle.
- Defects arrival by time interval and customer problem calls by time interval are the de facto metrics. The defects has been fixed up as soon as possible with excellent fix quality.
- It will not able to improve the defect rate of the product, but can improve the customer satisfaction to large extent.

Cont..

- Important metrics:
  - Fix backlog and backlog management index
  - Fix response time and fix responsiveness
  - Percent delinquent fixed
  - Fix quality

Fix backlog and backlog management index

- **Fix backlog** is a workload statement for software maintenance. It is related to both the rate of defect arrivals and the rate at which fixes for reported problems become available.
- Metric to manage the backlog of open, unresolved, problems is a backlog management index (BMI).

BMI = (number of problems closed during the month/number of problems arrivals during the month) *100%

If BMI>100, then the backlog is reduced.

If BMI <100, then the backlog increased.

Fix response time and fix responsiveness

- For many software development organizations, guidelines are established on the time limit within which the fixes should be available for the reported defects. So, criteria sets have been proposed with the problem's severity.
- Less severe defects uses fix response time.

Percent delinquent fixes

- The **mean (or median) response time metric** is a central tendency measure. A more sensitive metric is a percentage of delinquent fixes. For each fix, the turnaround time greatly exceeds the required response time, then it is classified as delinquent:

Percent delinquent fixes = (Number of fixes that exceeded the response time criterial by severity level/ number of fixes delivered in a specific time)*100%

Fix quality

- Fix quality or number of defective fixes is another important quality metric for the maintenance phase.
- Fix is known as defective when it did not fix the reported problem or if it fixes the problem but injected a new defects.
- The metric of percent defective fixes is simply the percentage of all fixes in a time interval (e.g. 1 month) that are defective.

Halstead’s software metrics

Cyclomatic complexity

- Used to measure the complexity of a program.
- Interprets a computer program as a set of a strongly connected directed graph.
- Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution.
- McCabe proposed the cyclomatic number, V (G) of graph theory as an indicator of software complexity. The cyclomatic number is equal to the number of linearly independent paths through a program in its graphs representation

Cyclomatic complexity

- There are three methods of computing Cyclomatic complexities.
  - Method 1: Total number of regions in the flow graph is a Cyclomatic complexity.
  - Method 2: The Cyclomatic complexity, V (G) for a flow graph G can be defined as

    V (G) = E - N + 2

    Where: E is total number of edges in the flow graph.

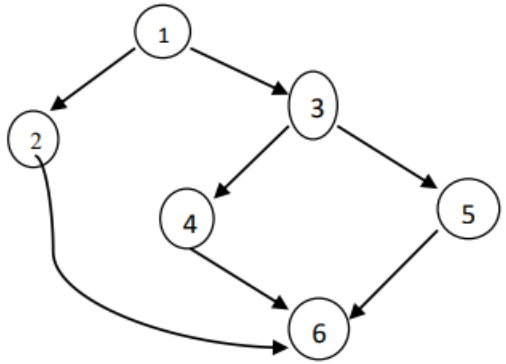
    N is the total number of nodes in the flow graph.
  - Method 3: The Cyclomatic complexity V (G) for a flow graph G can be defined as

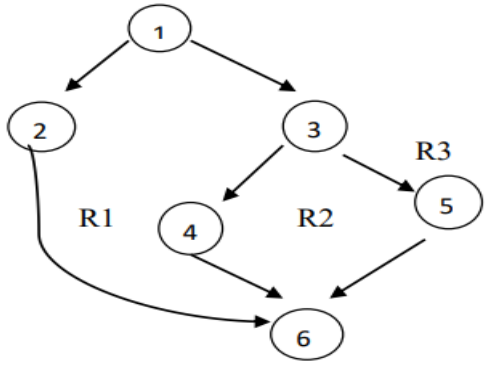
    V (G) = P + 1

    Where: P is the total number of predicate nodes contained in the flow G.

- Consider following code fragment with line numbered

```
{
1. If (a<b)
2. F1 () ;
else
{
3. If (a<c)
4. F2 () ;
else
5. F3 () ;
}
6. }
```





- To compute Cyclomatic complexity we will follow these steps –

Step 1. Design flow graph for given code fragment.

Step 2. Compute region, Predicate (i.e decision nodes) edges and total nodes in the flow graph

Step 3. Apply formulas in order to compute Cyclomatic complexity.

Causes of software maintenance problem

Legacy Information system

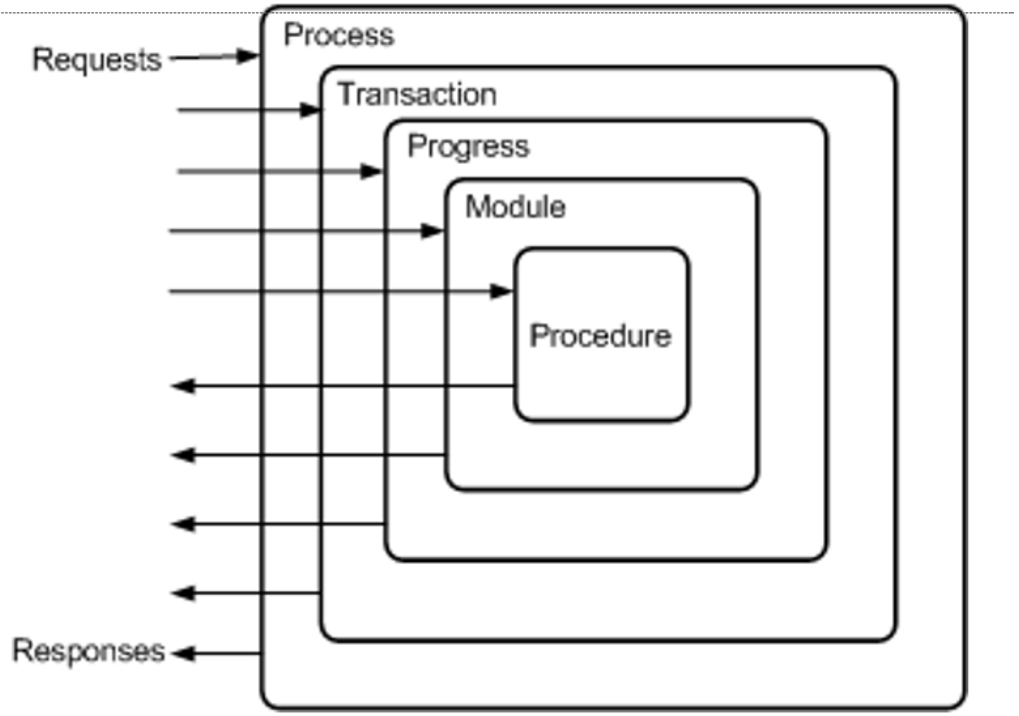
- Legacy software systems are large software systems that we don’t know how to cope with but that are vital to our organization.
- There are several categories of solutions for legacy information system (LIS). These solutions generally fall into six categories as follows:
  - Freeze.
  - Outsource.
  - Carry on maintenance.
  - Discard and redevelop.
  - Wrap.
  - Migrate.

Legacy Information system

- Wrapping means encapsulating the legacy component with a new software layer that provides a new interface and hides the complexity of the old component.
- Orfali et al. classified wrappers into four categories:
  - **Database wrappers:** Database wrappers can be further classified into forward wrappers (f-wrappers) and backward wrappers (b-wrappers)
  - **System service wrappers:** This kind of wrappers support customized access to commonly used system services, namely, routing, sorting, and printing.
  - **Application wrappers**: This kind of wrappers encapsulate online transactions or batch processes. These wrappers enable new clients to include legacy components as objects and invoke those objects to produce reports or update files.
  - **Function wrappers:** This kind of wrappers provide an interface to call functions in a wrapped entity. In this mechanism, only certain parts of a program – and not the full program – are invoked from the client applications. Therefore, limited access is provided by function wrappers.

Levels of encapsulation

- Sneed classified **five** levels of granularity: **procedures** are at the lowest level and **processes** are at the highest level.



Migration

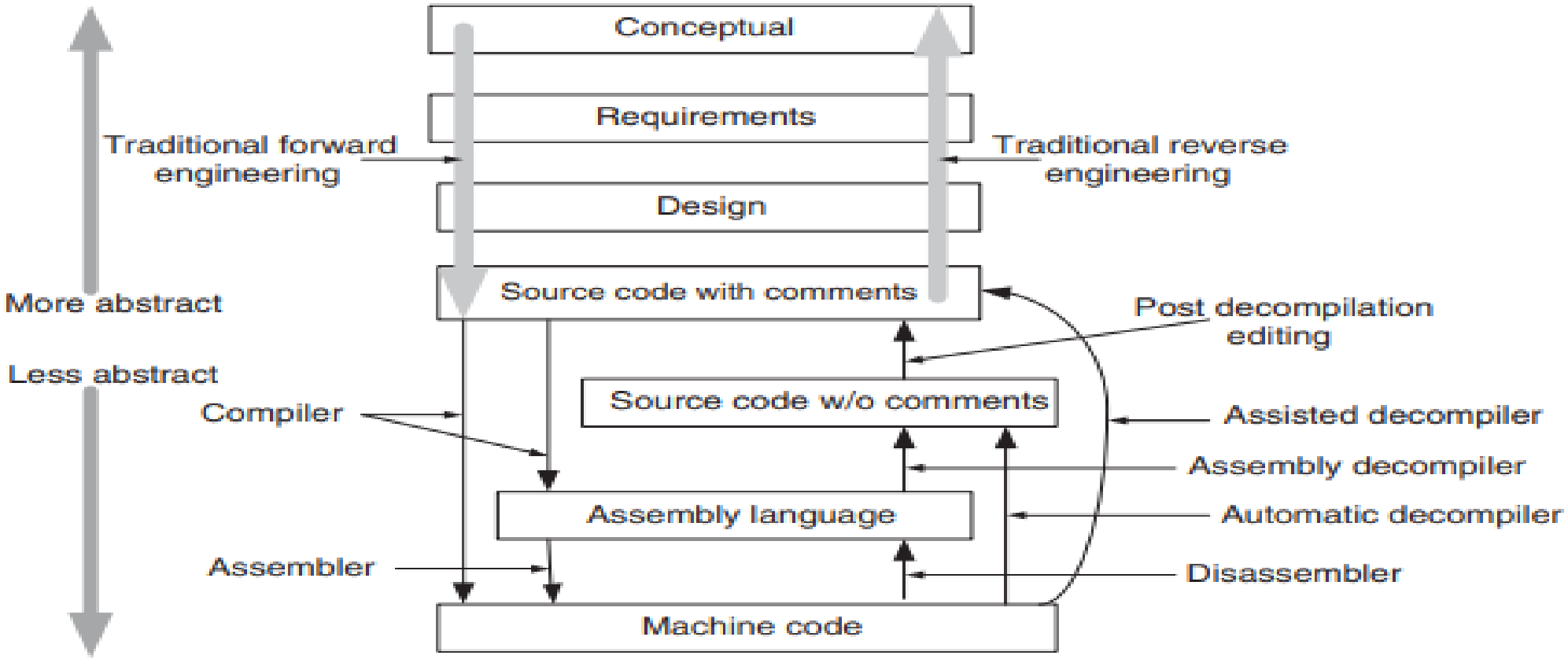
- Migration of LIS is the best alternative, when wrapping is unsuitable and redevelopment is not acceptable due to substantial risk.
- However, it is a very complex process typically lasting five to ten years.
- It offers better system understanding, easier maintenance, reduced cost, and more flexibility to meet future business requirements.
- Migration involves changes, often including restructuring the system, enhancing the functionality, or modifying the attributes.
- It retains the basic functionality of the existing system.

Migration

- The seven approaches are as follows:
  - **Cold turkey:** The Cold turkey strategy is also referred to as the Big Bang approach.
  - **Database first**: The Database first approach is also known as forward migration method.
  - **Database last**: In this approach, legacy applications are incrementally migrated to the target platform, but the legacy database stays on the original platform. Migration of the database is done last.
  - **Composite database:** The Composite database approach is applicable to fully decomposable, semi decomposable, and non-decomposable legacy information systems. In the composite database approach, the target information system is run in parallel with the legacy system during the migration process.
  - **Chicken little:** The Chicken little strategy refines the composite database strategy, by proposing migration solutions for fully decomposable, semidecomposable, and nondecomposable legacy systems with different kinds of gateways.

- **Butterfly:** The Butterfly methodology does not require simultaneous accesses of both the legacy database system and the target database system. The target system is not operated in the production mode, at the time of reengineering the legacy system. During the migration process, live data are not simultaneously stored in both the new system and the legacy system.
- **Iterative:** The iterative method implies that one component at a time is reengineered. Thus, the legacy system gradually evolves over a period of time.

Reengineering : Decompilation versus reverse engineering



THANK YOU…