Chapter 2

Software Maintenance

Contents

- Introduction
- Software Maintenance
  - Software Change
- Types of Software Maintenance
- Maintenance Process
- Maintenance Models
- Legacy System
- Software Rejuvenation
  - Re-engineering
  - Reverse Engineering
  - Refactoring

Introduction

- It is impossible to produce system of any size which do not need to be changed.
- Once software is put into use
  - New requirements emerge
  - Existing requirements changes as the business running that software changes
  - Errors must be repaired
  - New computers and equipment is added to the system
  - The performance or reliability of the system may have to be improved

- All of this means that, after delivery, software systems always evolve in response to demand for change.

Introduction

- Moreover, organizations have huge investments in their software systems - they are critical business assets.
- To maintain the value of these assets to the business, they must be changed and updated.
- The majority of the software budget in large companies is devoted to changing existing software rather than developing new software.
  - Studies indicate that up to 75% of all software professionals are involved in some form of maintenance/evolution activity.
- A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.
  - On an average, the cost of software maintenance is more than 50% of all SDLC phases.

Software Change

- There are a number of different strategies for software change.
  - Software maintenance
  - Architectural transformation
  - Software reengineering.

- **Software maintenance**
  - Changes to the software are made in response to changed requirements but the fundamental structure of the software remains stable.
  - This is most common approach used to system change.

Software Change…

- **Architectural transformation**
  - This is a more radical approach to software change than maintenance as it involves making significant change to the architecture of the system.

- **Software re-engineering**
  - This is different from other strategies in that no new functionality is added to the system.
  - System re-engineering may involve some structural modifications but dose not usually involves major architectural change.

Software maintenance: Definition

- Software maintenance is formally defined by IEEE as
  - The process of modifying the software system or component after delivery
    - to correct faults,
    - to improve performance or other attributes, or
    - to adapt to a change in the environment.

- Software maintenance does not normally involve major architectural changes to the system.
  - Changes are implemented by modifying existing system components and, where necessary, by adding new components to the system.

Types of Maintenance

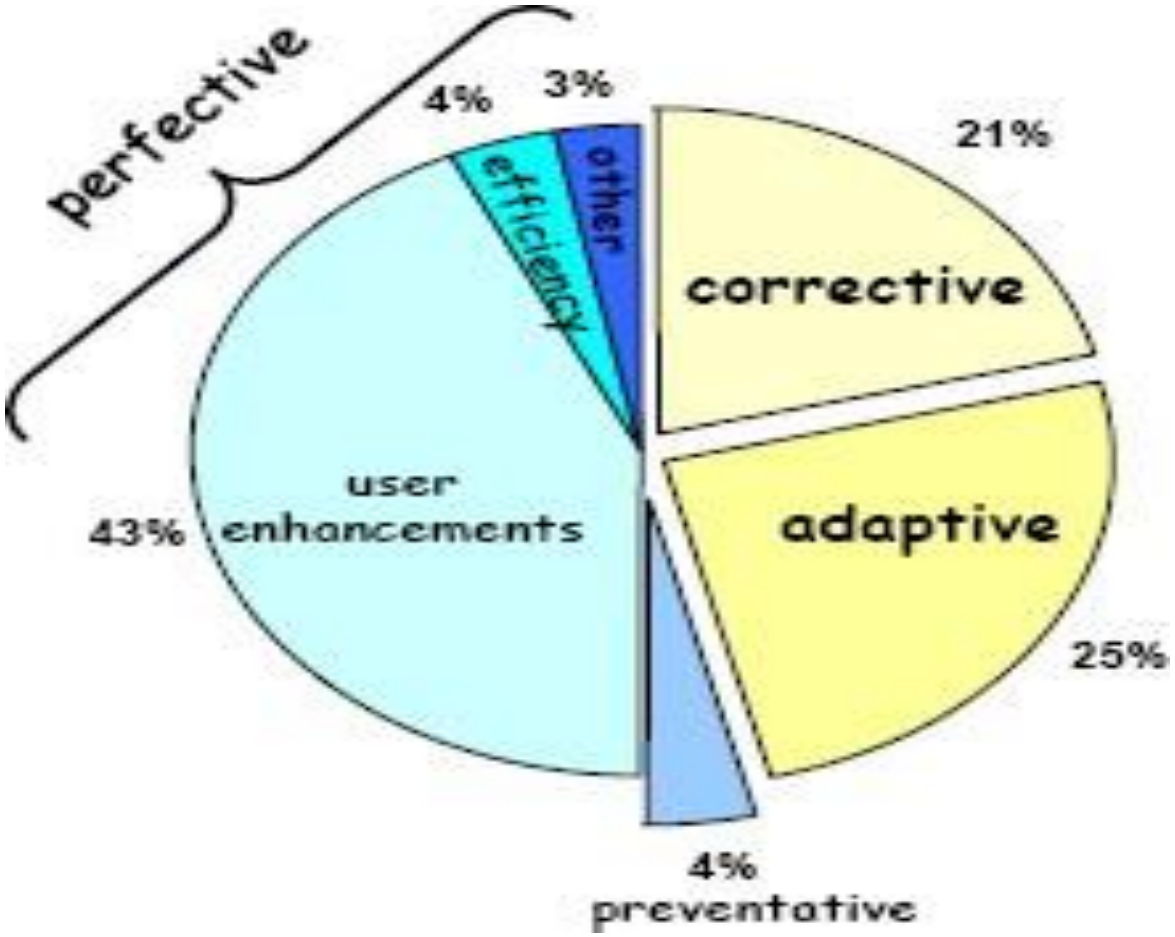
- Corrective Maintenance
  - fixing latent errors
  - Eg, patching security vulnerability, fixing crash, correcting inaccurate calculation
- Adaptive Maintenance
  - responding to external changes – new legal/regulation
  - changes in hardware platform
  - changes in support software
- Perfective Maintenance
  - improving performance, usability
  - Adding new functionality
  - efficiency improvements
- Preventative Maintenance
  - Improves (future) maintainability
  - Refactoring, implement better testing procedure, Documenting, commenting, etc.

fixing

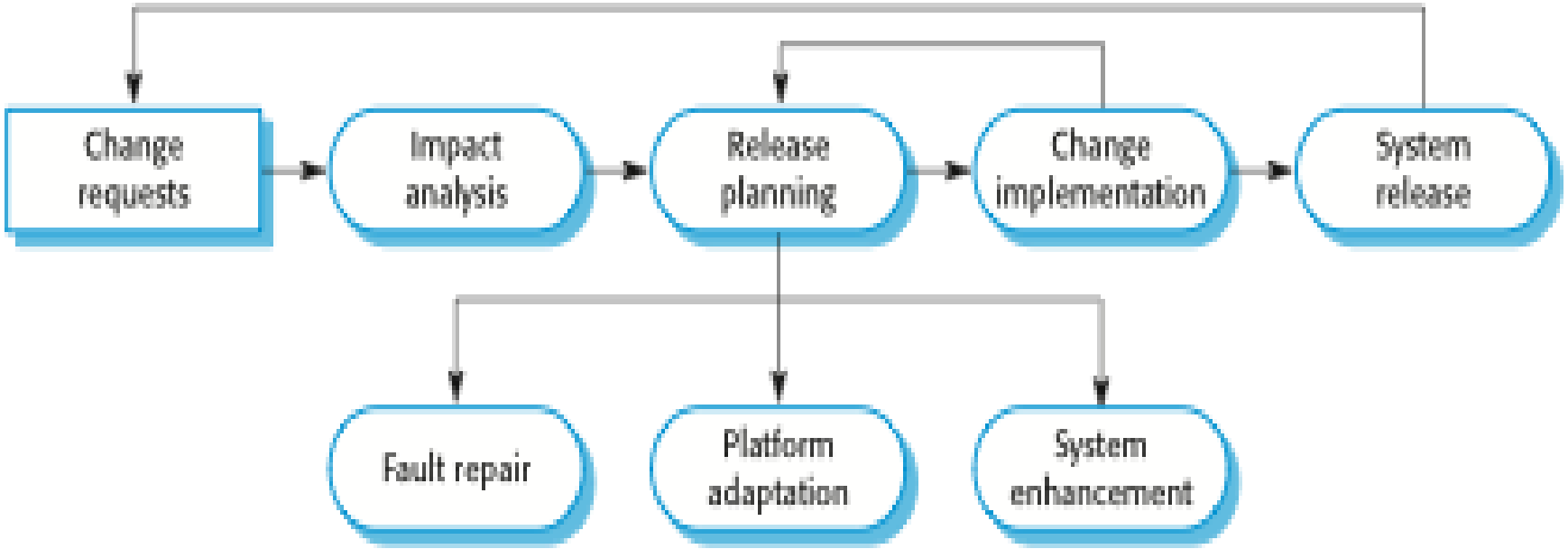
enhancing

Types of Maintenance…

- The majority of software maintenance is concerned with evolution deriving from user requested changes.



Software Maintenance Process



Software Maintenance Process…

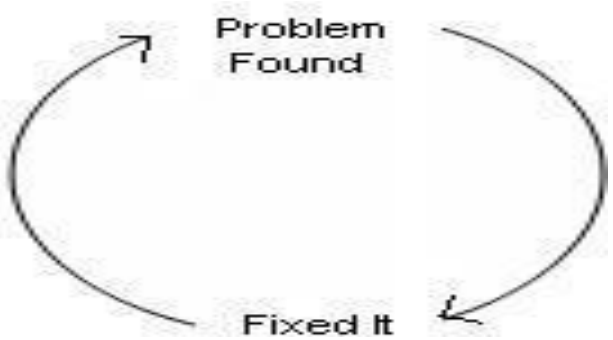
- **Change Requests:** This process is triggered by a set of change requests from system users, management, or customers.
- **Impact Analysis:** The cost and impact of these changes are assessed.
- **System release planning:** If the proposed changes are accepted, a new release of the system is planned. This release will usually involve elements of adaptive, corrective, and perfective maintenance.
- **Change Implementation:** The changes are implemented and validated
- **System release:** A new version of the system is released.
- The process then iterates with a new set of changes proposed for the new release.

Maintenance Models

- Maintenance is development; so it requires special skill
- Most of the times
  - Maintenance activities are performed without requirement or design document
  - It is difficult to understand the old code
- There fore maintenance models are required
  - However, the models are neither so well developed nor so well understood as models for software development
- Some examples of maintenance models are
  - Quick-fix Model
  - Iterative Enhancement Model
  - Reuse Oriented Model
  - Boehm's Model
  - Taute Maintenance Model

Quick-fix Model

- This is basically an adhoc approach to maintaining software.
- It is a fire fighting approach, waiting for the problem to occur and then trying to fix it as quickly as possible.
- Changes are made at code level as early as possible without anticipating future problems.
- It works well if the developer and maintainer is same person
  - Can be done quickly and cheaply
- Not suitable for large software systems.



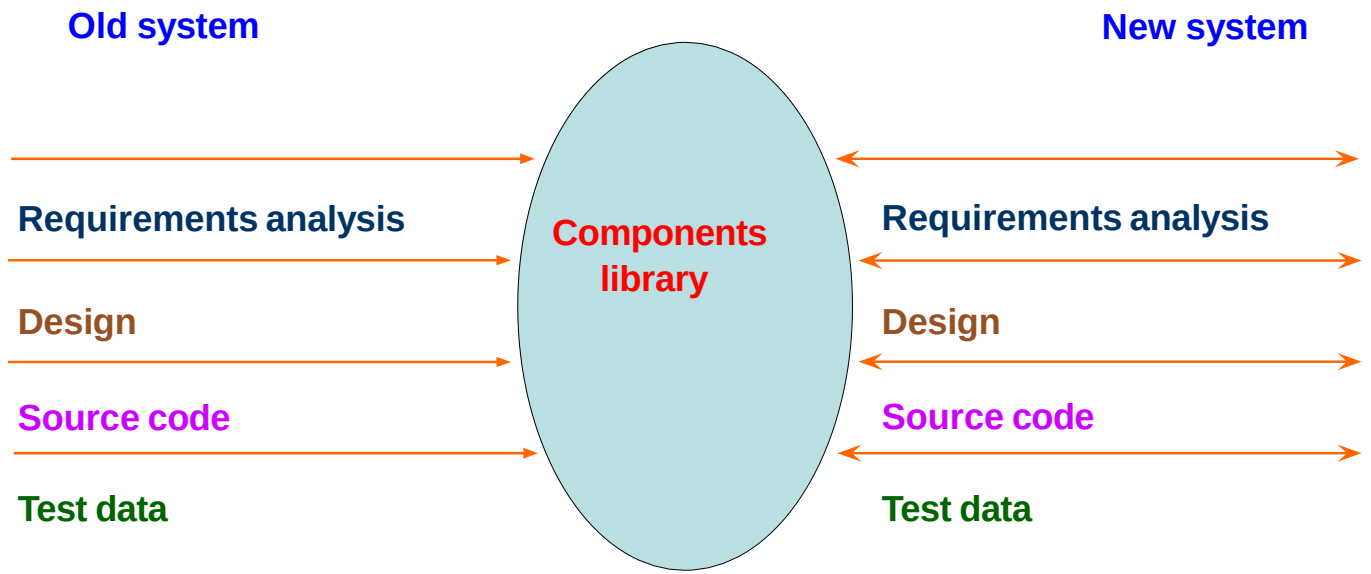
Iterative Enhancement Model

- It considers that the changes made to the software system are iterative in nature.
- The model has three stage cycle, namely, analysis of software system, classification of requested modifications, and implementation of requested modifications.
- The development initialize by keep analyzing and implementing the changes that needed. It produces a new version in every cycle.
- Incorporates changes in the software based on the analysis of the existing system.
- Documentations of each stage (requirement, design, coding, testing) is modified → the system is redesigned
  - Assumes existence of full documentation of a system

Reuse Oriented Model

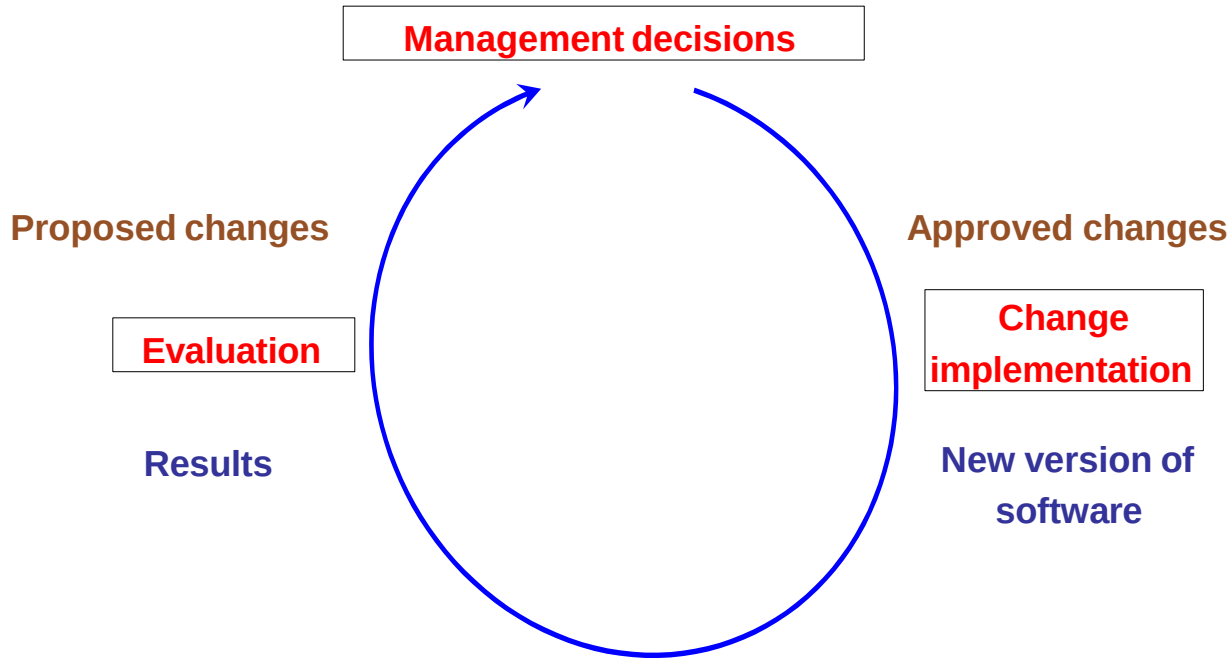
- The reuse-oriented model assumes that the existing program components can be reused to perform maintenance
- The reuse model has four main steps:

1) Identifying the components of the old system which can be reused
2) Understanding these components
3) Modifying the old system components so that they can be used in the new system
4) Integrating the modified components into the new system.



Boehm's Model

- Boehm proposed a model for the maintenance process based upon the economic models and principles.
- Boehm represent the maintenance process as a closed loop cycle.
- Changes are proposed first.
- Then changes are made.



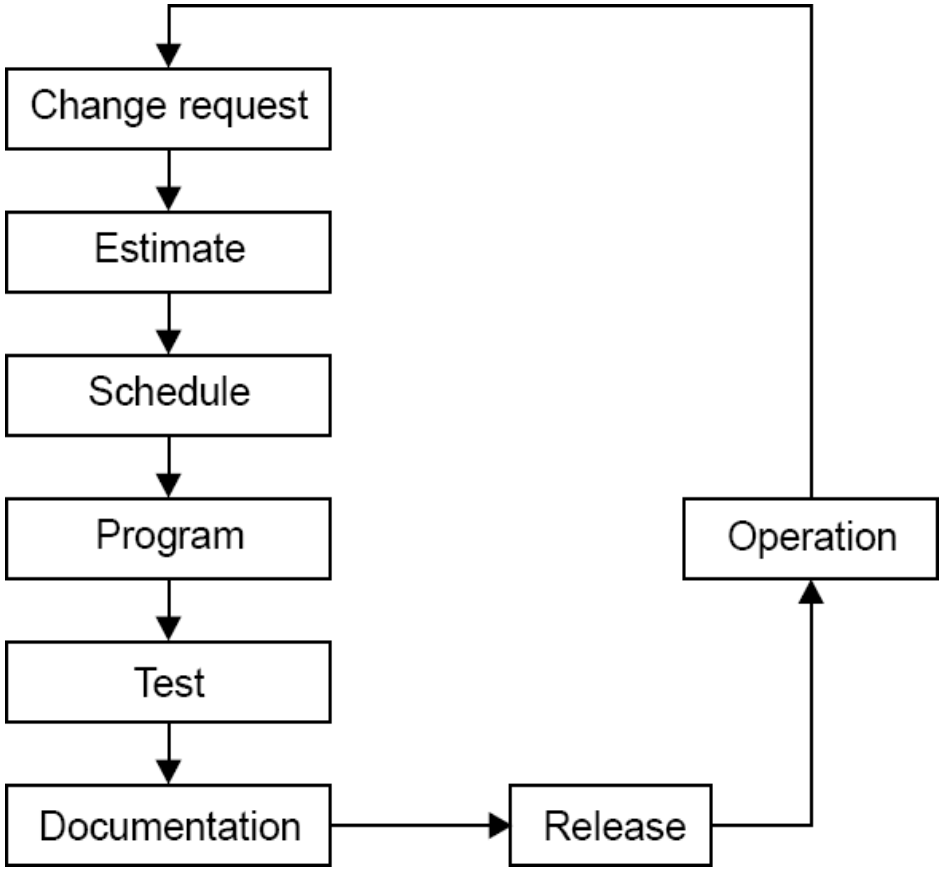
Boehm's Model…

- It clearly shows that how decisions should be made to follow up in a prescribed cycle to get a desired output at the management end. The step-by-step descriptions are as follows:

a) **Management Decisions:** In this stage, the decisions should be made by desired management team, by applying certain strategies and efforts to make new changes.
b) **Change Implementation:** In this stage, the change identified by the management will be approved by change implementation team and perform multiple changes within software and releases the new version of it.
c) **Software in use:** After the implementation, the team will check the usability to ensure how it is working at user's end.
d) **Evaluation:** At this stage, evaluating the whole process and the results will be identified. If evaluation team didn't find required results it may pass it back to the management team.

Taute Maintenance Model

- Developed by B.J Taute in 1983
- It is very easy to understand and implement
- It is a typical maintenance model and has eight phases in cycle fashion.



Taute Maintenance Model…

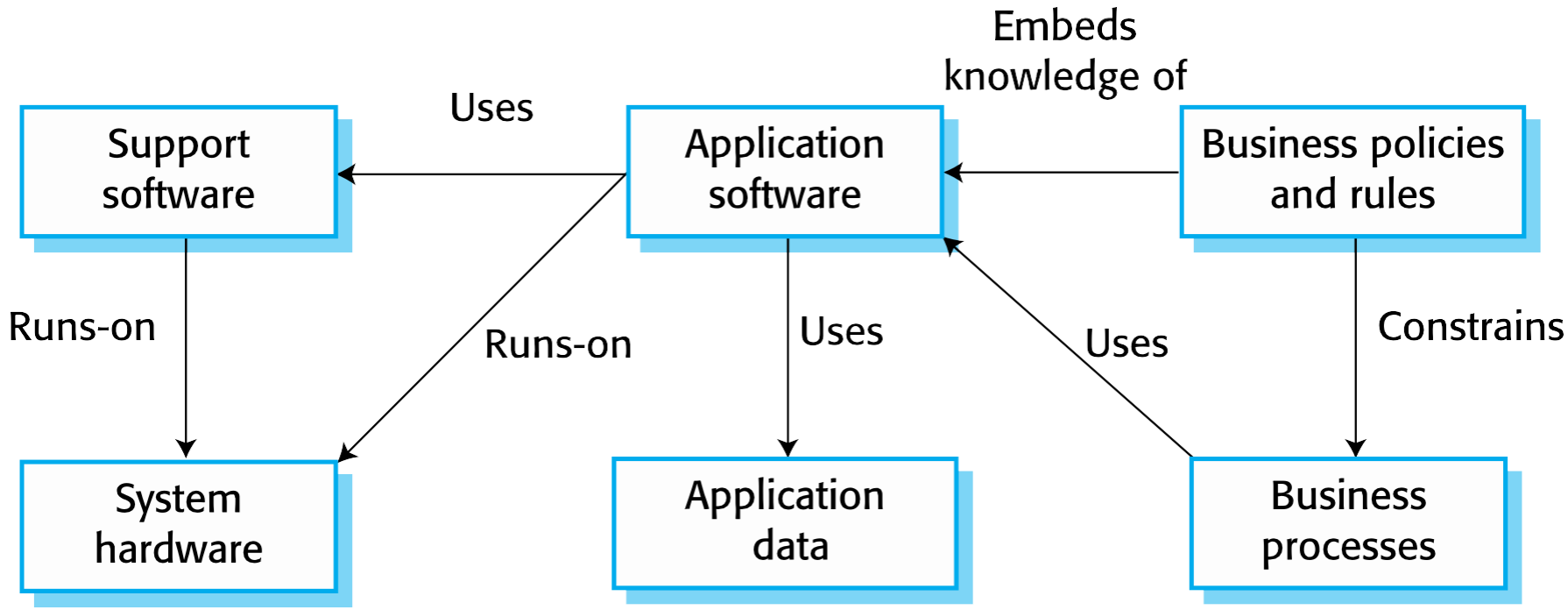
- The phases are
  - **Change request phase** – a request in a prescribed format from the client is obtained. The request may be corrective, adaptive, perfective or preventive
  - **Estimate phase** – time and effort required for the change are required
  - **Schedule phase** – identify change requests for the next scheduled release
  - **Programming phase** – source code is modified to implement the changes
  - **Test phase** – the modified code is tested (Regression testing)
  - **Documentation phase -** after regression testing, user and system documents are prepared/updated before releasing the system
  - **Release phase** – the new software product along with updated documents are delivered to the customer. Acceptance testing is carried out by the users of the system
  - **Operation phase** – software is placed under normal operation

Legacy systems

- Systems that have been in a field for a long time evolve and mutate in ways that were never planned
- As enhancements are added, bugs fixed, and piece-meal solutions tacked on, the once elegant system grows into something unmanageable and expensive to maintain.This system is called *Legacy system* – a legacy system is typically
  - very old and large
  - has been heavily modified and are based on the old technology
  - Poorly structured and documentation not available
  - were not designed for change
  - yet still in use
  - none of the original members of the software development team may still be around
  - often support very large quantities of live data
  - the software is often at the core of the business and replacing it would be a **huge** expense.

Legacy systems…

- Legacy systems are not just software systems but are broader socio-technical systems that include hardware, software, libraries and other supporting software and business processes.
- The elements of a legacy system



Legacy system components

- **System hardware**
  - Legacy systems may have been written for hardware that is no longer available.
- **Support software(customer)**
  - The legacy system may rely on a range of support software, which may be obsolete or unsupported.
- **Application software**
  - The application system that provides the business services is usually made up of a number of application programs.
- **Application data**
  - They implies any data created and managed by an application. E.g. configuration files.
  - These are data that are processed by the application system.
  - They may be inconsistent, duplicated or held in different databases.

Legacy system components…

- **Business processes**
  - These are series of steps that are used and performed in the business to achieve some business objective.
  - Business processes may be designed around a legacy system and constrained by the functionality that it provides.
- **Business policies and rules**
  - These are definitions of how the business should be carried out and constraints on the business.
  - Use of the legacy application system may be embedded in these policies and rules.

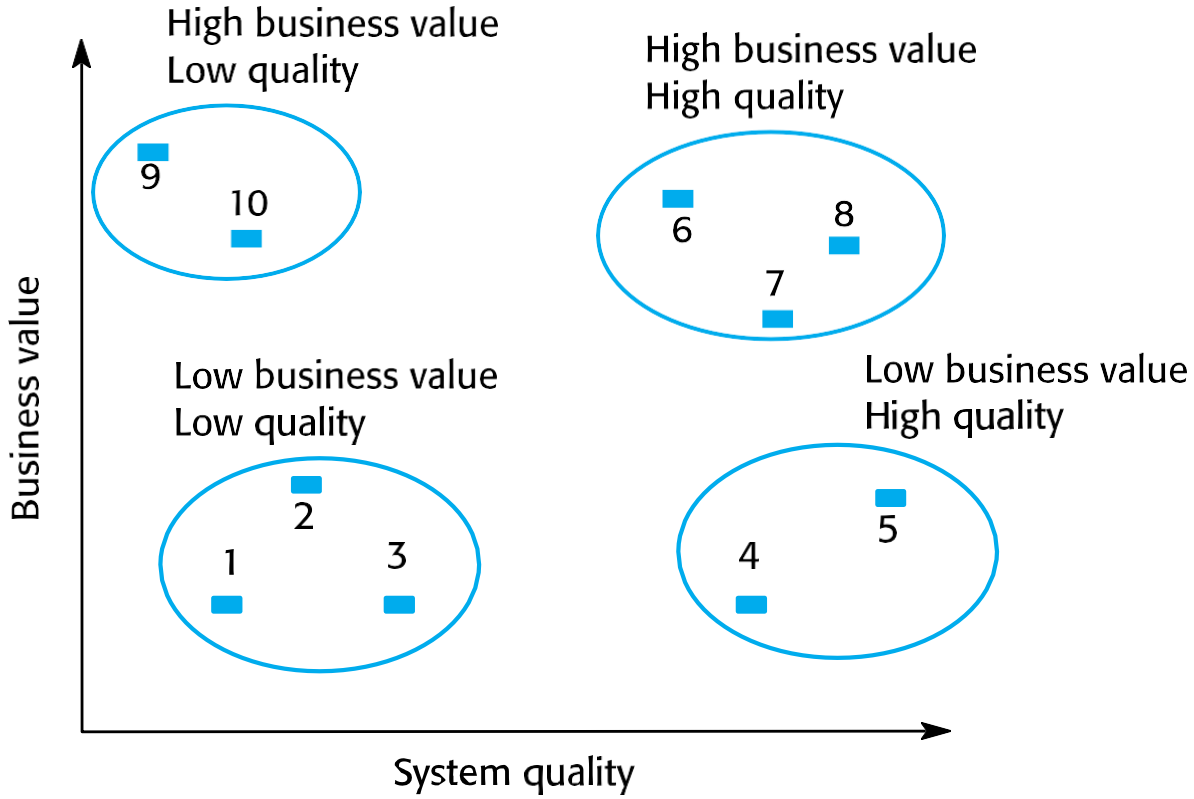
Legacy Systems Management

- Dealing with legacy system is very hard and there are some solutions.
  - ***Scrap/Dispose the system completely***
    - Best option when the system is not making an effective contribution to business processes.
  - ***Leave the system unchanged and continue with regular maintenance***
    - *Best* option when the system is still required but is fairly stable and the system users make relatively few change requests.
  - ***Replace all or part of the system with a new system***
    - Best option when factors, such as new hardware, mean that the old system cannot continue in operation or where off-the-shelf systems would allow the new system to be developed at a reasonable cost.
  - ***Reengineer the system to improve its maintainability***
    - Best option when the system quality has been degraded by change and where a new change to the system is still being proposed.

Legacy systems Management…

- The strategy chosen should depend on the system quality and its business value.

An example of a legacy system assessment



Legacy system Categories

- Low quality, low business value
  - These systems should be scrapped.
- Low-quality, high-business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain.
- High-quality, high business value
  - Continue in operation using normal system maintenance.

Software Rejuvenation

- It is the concept of gracefully terminating an application and immediately restarting it at a clean internal state.
- It helps to prevent performance degradation and other associated failures related to software aging.
- Re-documentation
  - Creation or revision of alternative representations of software at the same level of abstraction
  - Generates:
    - data interface tables, call graphs, component/variable cross references etc.
- Restructuring
  - transformation of the system's code without changing its behavior
- Re-engineering
- Reverse Engineering
- Refactoring

Reengineering

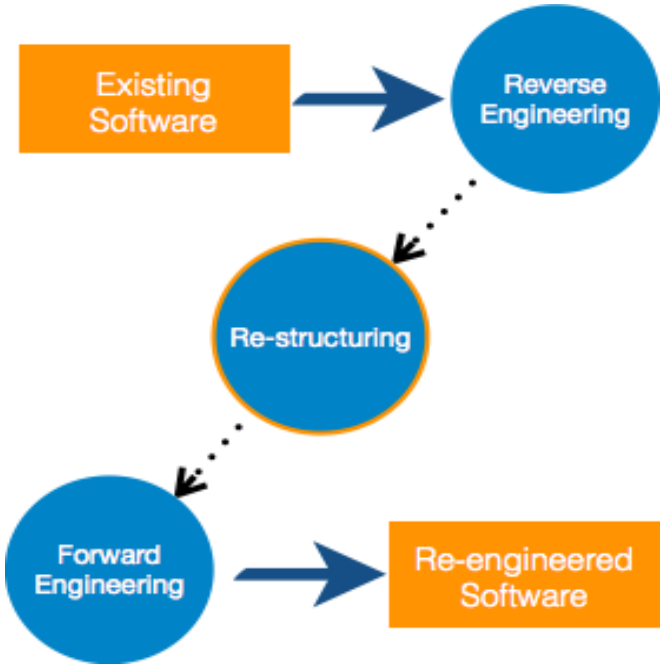
- Also known as renovation, reclamation
- Rewriting parts or all of a legacy system to make it more evolvable, so that it can more easily accommodate future change requests.
- Reengineering is applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain.
  - The system may be restructured and re-documented.
- When to Re-engineer
  - System changes are mostly restricted 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

Reengineering…

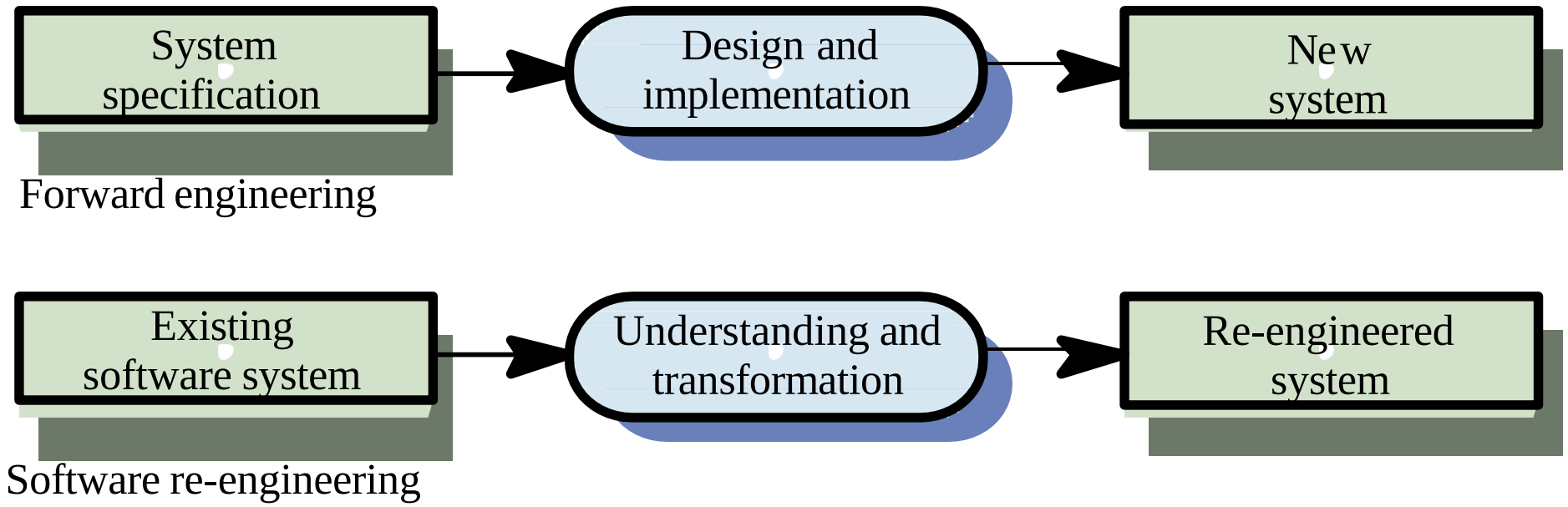
- Reengineering has the following advantages
  - Reduced risk
    - There is a high risk in new software development.
    - There may be development problems, staffing problems and specification problems
  - Reduced cost
    - The cost of re-engineering is often significantly less than the costs of developing new software
- **Re-Engineering Cost Factors**
  - The quality of the software to be re-engineered
  - The tool support available for re-engineering
  - The extent of the data conversion which is required
  - The availability of expert staff for re-engineering

Reengineering Process

- Decide what to re-engineer. Is it whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.
- Restructure Program if required. For example,
  - changing function-oriented programs into object-oriented programs.
    - re-arranging the source code, either in same programming language or from one programming language to a different one
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.



Forward Engineering VS Re-engineering



- **Forward Engineering**
  - It is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering.
  - It assumes that there was some software engineering already done in the past.

Reengineering process Activities

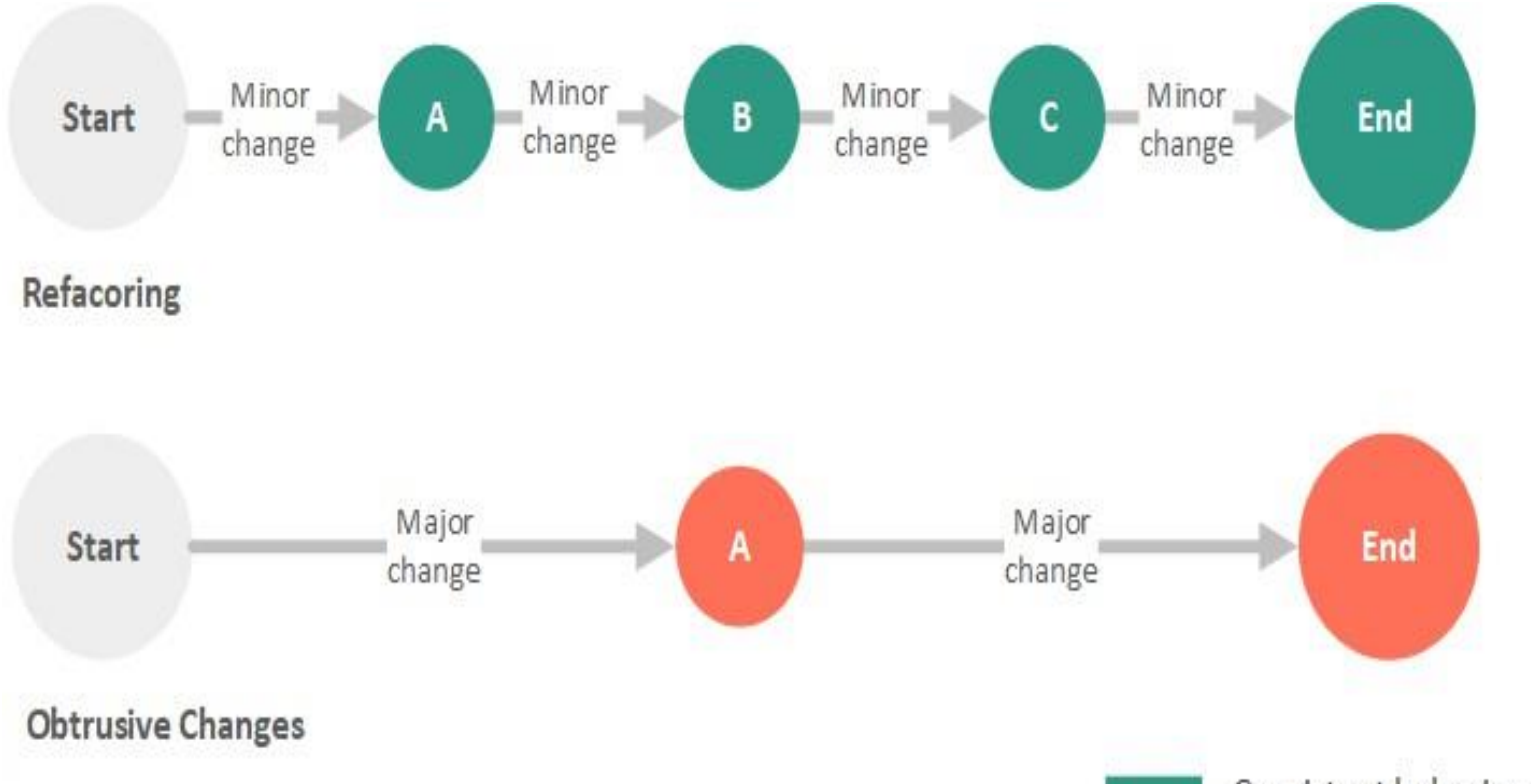
- Source code translation
  - Convert code to a new language.
- Reverse engineering
  - Analyze the program to understand it;
- Program structure improvement
  - Restructure automatically for understandability;
- Program modularization
  - Reorganize the program structure;
- Data reengineering
  - Clean-up and restructure system data.

Preventative maintenance by Refactoring

- Refactoring is the process of making improvements to a program to slow down degradation through change.
- It is the process of restructuring existing computer code without changing its external behavior.
- You can think of refactoring as ‘preventative maintenance’ that reduces the problems of future change.
- Refactoring involves modifying a program to improve its structure, reduce its complexity or make it easier to understand.
- When you refactor a program, you should not add functionality but rather concentrate on program improvement.
- Re-engineering takes place after a system has been maintained for some time and maintenance costs are increasing.
- You use automated tools to process and re-engineer a legacy system to create a new system that is more maintainable.

Refactoring…

- Refactoring is a continuous process of improvement throughout the development and evolution process.
- It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.
- Refactor when you
  - Add new functionality
  - Fix an existing code
  - Review someone else's code
  - Implement Test-Driven Development(TDD)



Refactoring…

- 'Bad smells' in program code indicate need of refactoring
  - Duplicate code
    - The same or very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.
  - Long methods
    - If a method is too long, it should be redesigned as a number of shorter methods.
  - Switch (case) statements
    - These often involve duplication, where the switch depends on the type of a value.
    - The switch statements may be scattered around a program.
    - In object-oriented languages, you can often use polymorphism to achieve the same thing.
  - Data clumping
    - Data clumps occur when the same group of data items (fields in classes, parameters in methods) re-occur in several places in a program. These can often be replaced with an object that encapsulates all of the data.
  - Speculative generality - This occurs when developers include generality in a program in case it is required in the future. This can often simply be removed.