

Chapter 2

Software Maintenance

By Yohannes S.

Contents

Introduction

Software Maintenance

 Software Change

 Definition, history

Software Maintenance - Examples

Types of Software Maintenance

Maintenance Process

Maintenance Prediction

Maintenance Costs

Problems During Maintenance

Maintenance Side Effects

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

- Etc

All of this means that, after delivery, software systems always evolve in response

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.

Software maintenance costs 40 - 90% of the software lifecycle

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 does not usually involve major architectural change.

Program Evolution Dynamic

1 Program evolution dynamic is the study of system change.

1 From these studies, a sets of laws

Law	Description
Continuing change	As a program that is used in real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplify the structure.
Large program evolution	Program evolution is self-regulation process. System attributes such as size, time between release and the number of report errors are approximately invariant for each system release

Program Evolution Dynamic...

Law	Description
Conservation of familiarity	Over the lifetime of system, the incremental change in each release is approximately constant.
Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
.....	

Definition

Software maintenance is formally defined by IEEE91 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

History

1960s - 1970s

Inclusion of maintenance in waterfall lifecycle after delivery of the software product.

Perception that post-delivery activities only consisted of bug fixes and minor adjustments.

Did not account for the need to add functionality due to new and changed requirements.

1970s

Lehman postulated the initial laws of program evolution.

Stressed the need for continuous evolution due to changes in the software's operational environment.

Late 1970s - 1980s

Initial process models that handled change requests.

1990s

General acceptance of software evolution.

Development of new process models that accounted for evolution activities: evolutionary development, spiral model, agile software development.

Examples

Y2K

many, many systems had to be updated
language analyzers (find where changes need to be made)

Anti-Virus Software

don't usually have to update software, but must send virus definitions

Operating System Patching

Microsoft, Apple, Linux/Unix

OS is core to use of computer, so it must be constantly maintained

Commercial Software in General

customers need to be informed of updates
updates have to be easily available - web is

Types of Maintenance

Corrective Maintenance

fixing latent errors

includes temporary patches and workarounds

Adaptive Maintenance

fixing

responding to external changes

changes in hardware platform

changes in support software

Perfective Maintenance

improving the as-delivered software

user enhancements

efficiency improvements

enhancing

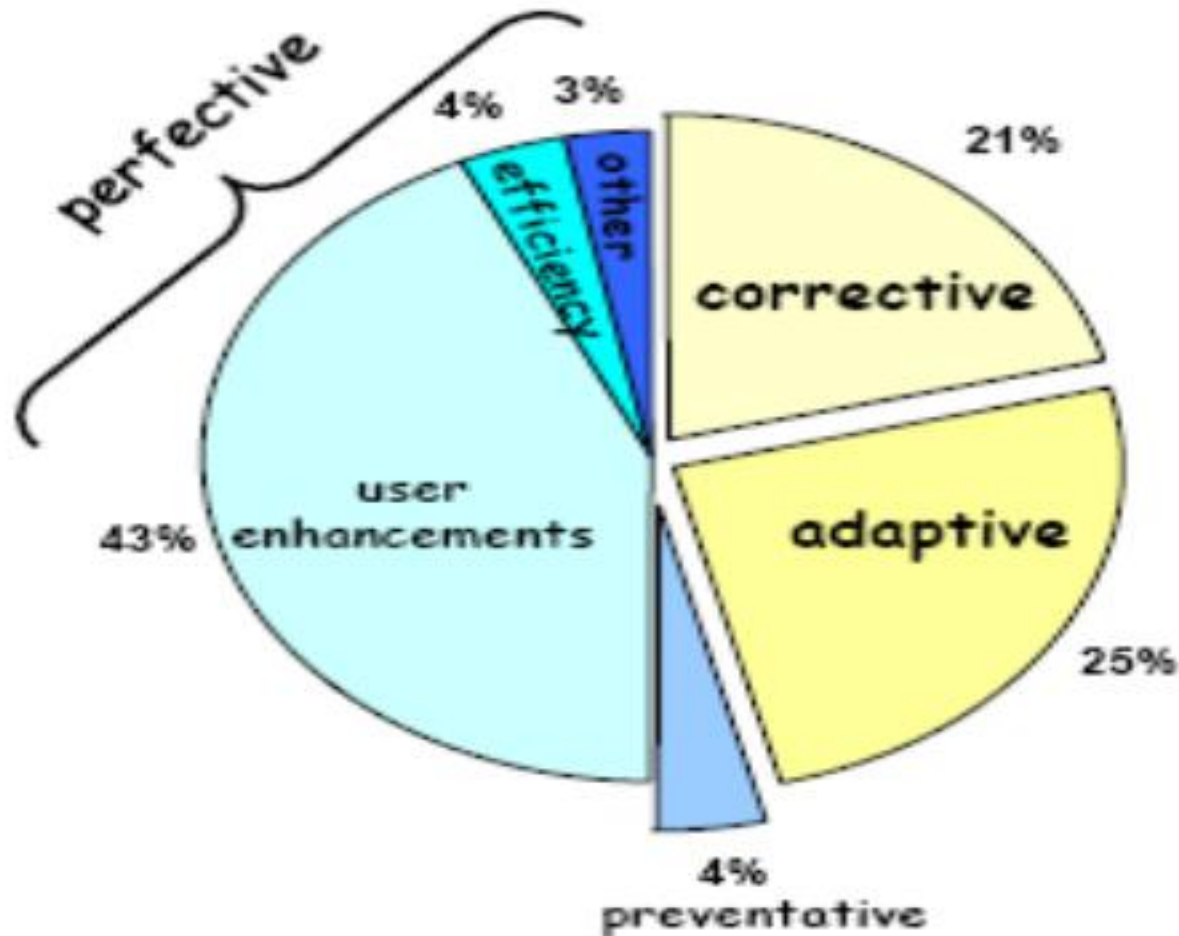
Preventative Maintenance

Improves (future) maintainability

Documenting, commenting, etc.

Types of Maintenance...

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



Software Maintenance Process

Maintenance processes depend on

- The type of software being maintained

- The development processes used

- The skills and experience of the people involved

Proposals for change are the driver for system evolution.

The maintenance process continues after

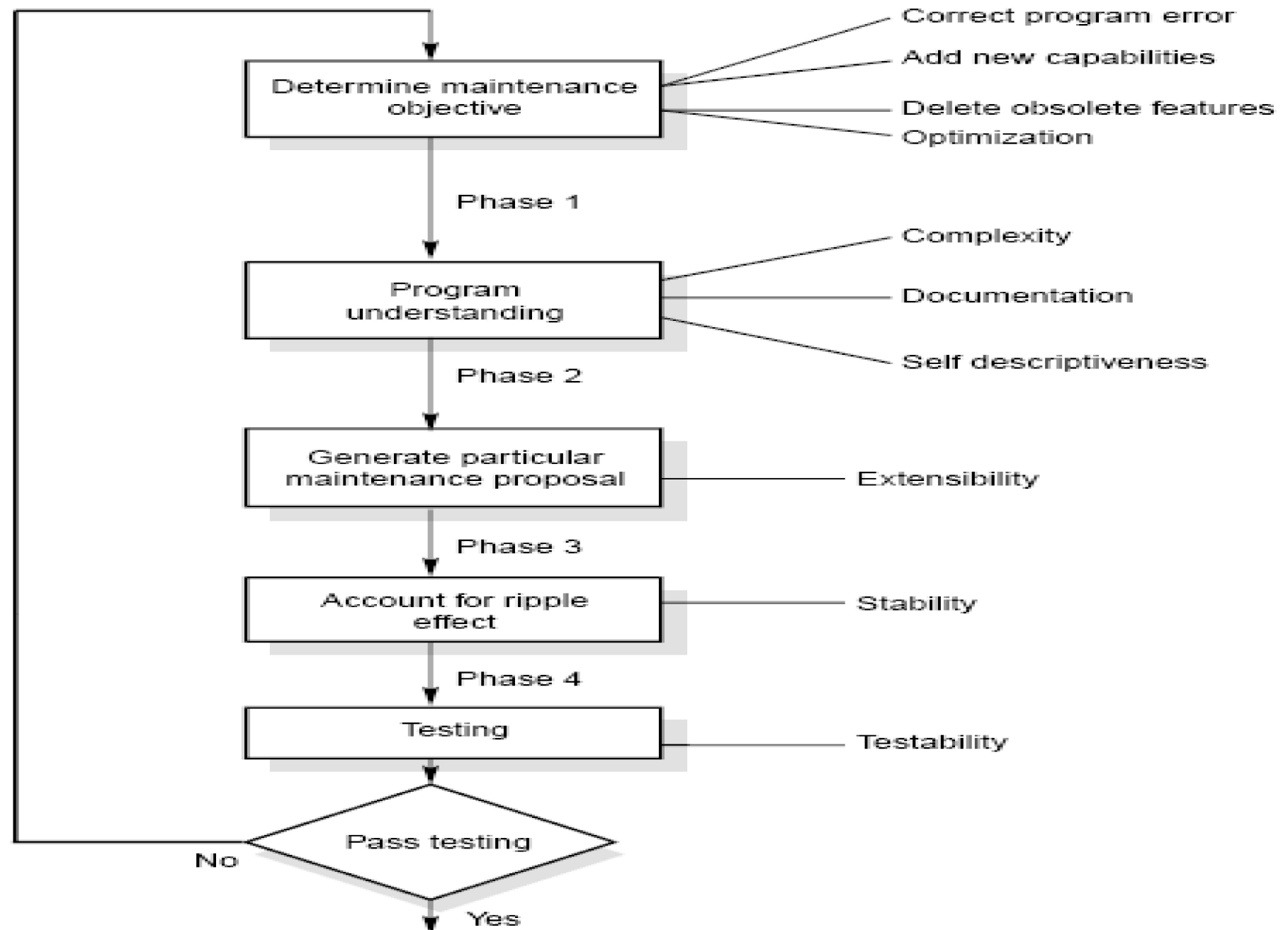
impact analysis

- Change requests comes from users, customers or management

- all change requests should be carefully analysed

Activities of Software Maintenance process are shown on next slide

Urgent changes may have to be implemented



Change Management

Change is a fact of life for large software.

A defined change management process and associated CASE tools ensure that these changes are recorded and applied to the system in a cost-effective way.

The change management process should come into effect when the software associated document is put under the control of the configuration management team.

Change management procedures should be designed to ensure that the costs and benefits of change are properly analyzed and changes to a system are made in a controlled

Change Management Process

Request change by completing a change request form

Analyze change request

If change is valid then

```
{  
  Assess how change might be implemented  
  Assess change cost  
  Record change request in database  
  Submit request to change control board  
  //Change request board may accept or  
  reject the change  
  // request
```

Process...

```
If change is accepted then{
  Repeat{
    make changes to software
    record changes and link to associated
    change request
    submit changed software for quality approval
  }Until{
    software quality is adequate
  }
  create new system version
}else {
  reject change request
}
}
```

Process...

Change Request Form

Project: Proteus/PCL-Tools

Number: 23/94

Change requester: I.Sommerville

Date: 1/9/98

Requested change: when a component is selected from the structure, display the name of the file where it is stored.

Change analyzer: G.Dean

analysis Date: 10/9/98

Components affected: Display-icon.Select, Display-icon.Display

Associated component: File Table

Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low

Change implementation:

Estimated effort: 0.5 days

Date to CCB: 15/9/98

CCB decision date: 1/11/98

Change implementor:

Date of change:

Date submitted to QA:

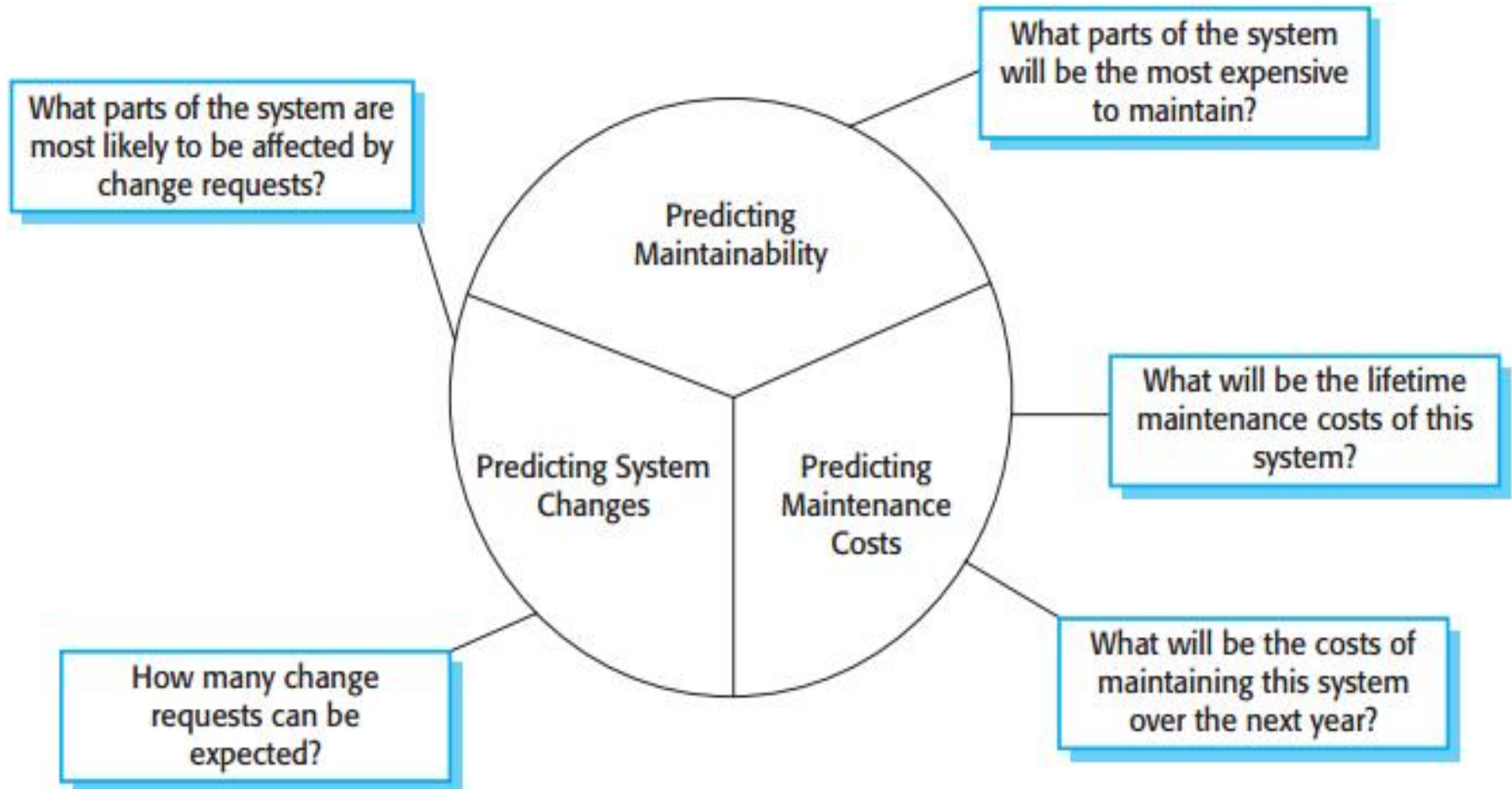
QA decision:

Date submitted to CM:

comments

Maintenance Prediction

Managers hate surprises, especially if these result in unexpectedly high costs.



Maintenance Prediction...

Change prediction

Predicting the **number of changes** requires understanding of the relationships between a system and its environment

Tightly coupled systems require changes whenever the environment is changed

Factors influencing this relationship are

- Number and complexity of system

- interfaces

- Number of inherently volatile system

- requirements

- The business processes where the system is used

Maintenance Prediction...

Complexity metrics

Predictions of maintainability can be made by assessing the complexity of system components.

Complexity depends on

- Complexity of control structures

- Complexity of data structures

- Procedure and module size

Maintenance Prediction...

Process metrics

Process measurements may be used to assess maintainability

- Number of requests for corrective maintenance

- Average time required for impact analysis

- Average time taken to implement a change request

- Number of outstanding change requests

If any or all of these is increasing, this may indicate a decline in maintainability

Maintenance Costs

Usually greater than development costs (2* to 100* depending on the application)

Affected by both technical and non-technical factors

Increases as software is maintained.

Maintenance corrupts the software structure so makes further maintenance more difficult.

Ageing software can have high support costs (e.g. old languages, compilers etc.)

Time and money (software that costs £ 10 a line to develop costs £ 400 a line to maintain)

Maintenance Costs...

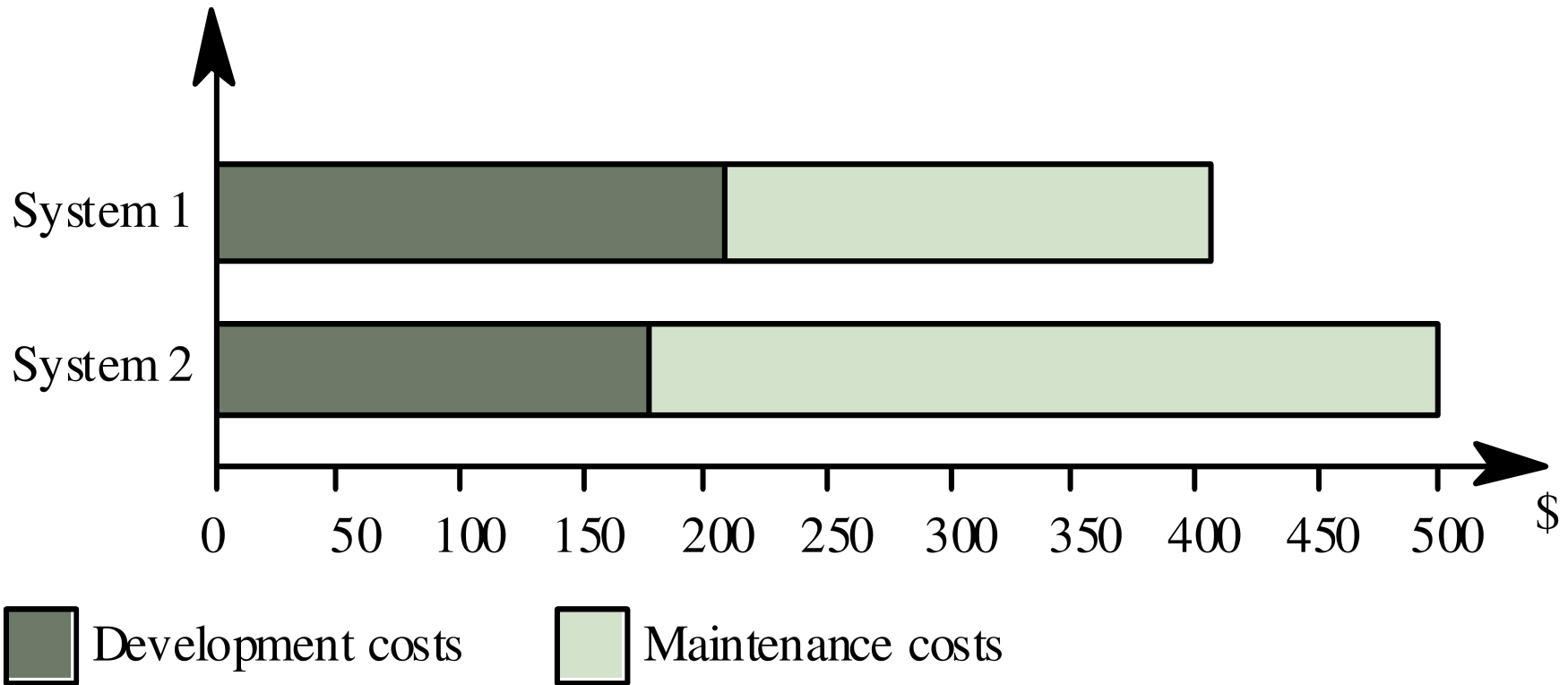
Organizations become maintenance bound and cannot produce new software

Customer dissatisfaction when seemingly legitimate requests for repair or modification cannot be addressed in a timely manner

Reduction in overall software quality as changes introduce latent errors in the maintained software

Upheaval /sudden change/disruption/ caused during development efforts when staff must be “pulled” to work on a maintenance task

Development/Maintenance Costs



Maintenance Cost Factors

Team stability

Maintenance costs are reduced if the same staff are involved with them for some time

Contractual responsibility

The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change

Staff skills

Maintenance staff are often inexperienced and have limited domain knowledge

Program age and structure

As programs age, their structure is degraded and they become harder to understand and change

Problems During Maintenance

The root problem is **complexity**.

Limited Understanding

47% of Software Maintenance effort devoted to understanding the software

Poor documentation also negatively affects understandability

Low morale

Software maintenance is regarded as less interesting than development

Alignment with organizational objectives

Software maintenance is resources consuming and it has no clear quantifiable benefit for the organization.

Process issues

Software maintenance requires a number of additional activities not found in initial development

Maintenance...

Domino Effect (a.k.a Ripple Effect),
Continuing Change

More factors

Lack of human resources

Different programming styles conflict

Lack of documentation and tools

Bad maintenance management

Documentation policy

Turnover

Maintenance Side Effects

In this context a side effect implies an error or undesirable behavior that occurs as the result of a modification.

The three major areas where side effects are seen are

- Documentation
- code
- data structures

Documentation Side Effects

These consist of the failure to update documentation so that it no longer matches the code.

If the user doesn't know about changes frustration is inevitable.

Maintenance Side Effects...

Coding Side Effects

Any change can cause side-effects but these tend to be more error prone a subprogram is deleted or changed

A statement label is deleted or modified

An identifier is deleted or modified

Changes are made to improve execution performance

Logical operators are modified

Files are opened or closed

Design changes which translate into major code changes

Changes are made to logical tests of boundary conditions

These may be caught in testing or cause

Maintenance Side Effects...

Data Side Effects

Data side effects occur as the result of modifications made to a data structure.

The most error-prone are:

- redefinition of local and global constants

- redefinition of record or file formats

- Increase or decrease in size of array or other data structure

- modification of global data

- re initialization of control flags and pointers

- rearrangements of parameters (especially in I/O)

Chapter 2 - Software Maintenance (cont'd)

By Yohannes S.

Contents

Maintenance Models

Quick-fix Model

Iterative Enhancement Model

Reuse Oriented Model

Boehm's Model

Stable Maintenance
Model

Legacy System

Legacy System Components

Legacy System Management

Legacy system Categories

Software Rejuvenation

Re-engineering

Activities Reengineering Process

Reverse Engineering

Refactoring

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

Therefore 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

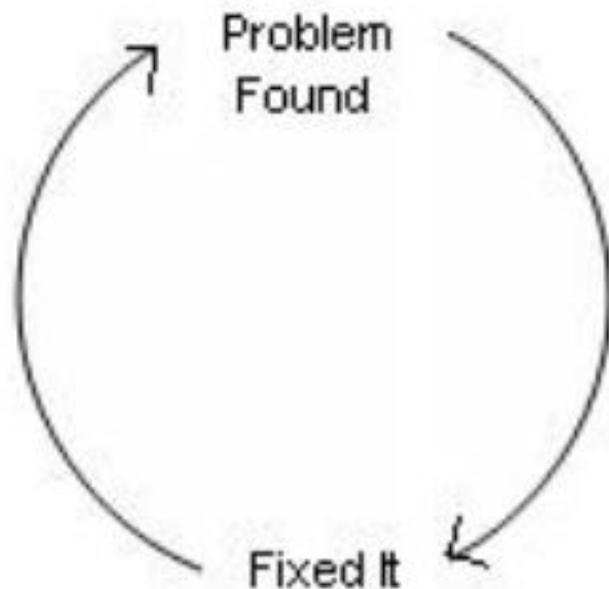
Quick-fix Model

an ad hoc approach

an unreliable model

It is a 'firefighting' approach

waiting for the problem to occur and
trying to fix it as quickly as possible



Quick-fix Model...

Why people use this model?

1. Pressure of deadlines and resources

If customers are demanding the correction of an error, for example, they may not be willing to wait for the organization to go through detailed and time consuming stages of risk analysis.

2. Often, a company will release a quick fix as a temporary measure.

The real solution will be implemented, along with other corrections and enhancements, as a major upgrade at a later date.

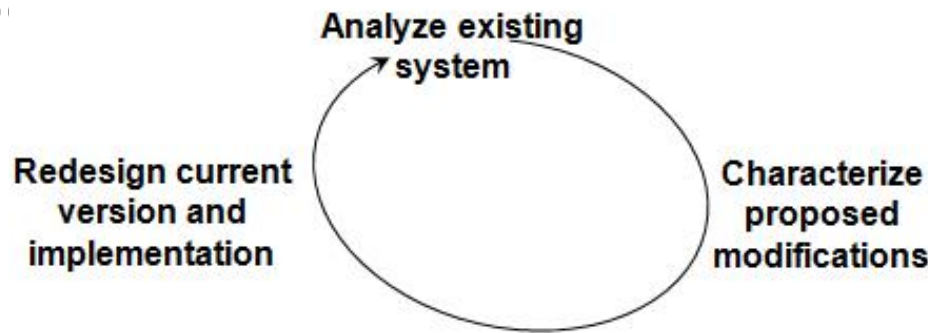
Changes made at the code level, as easily as possible

It works well if the developer and maintainer is same person

Iterative Enhancement Model

Changes in software system throughout its lifetime are an iterative process – adapted from development model

The model



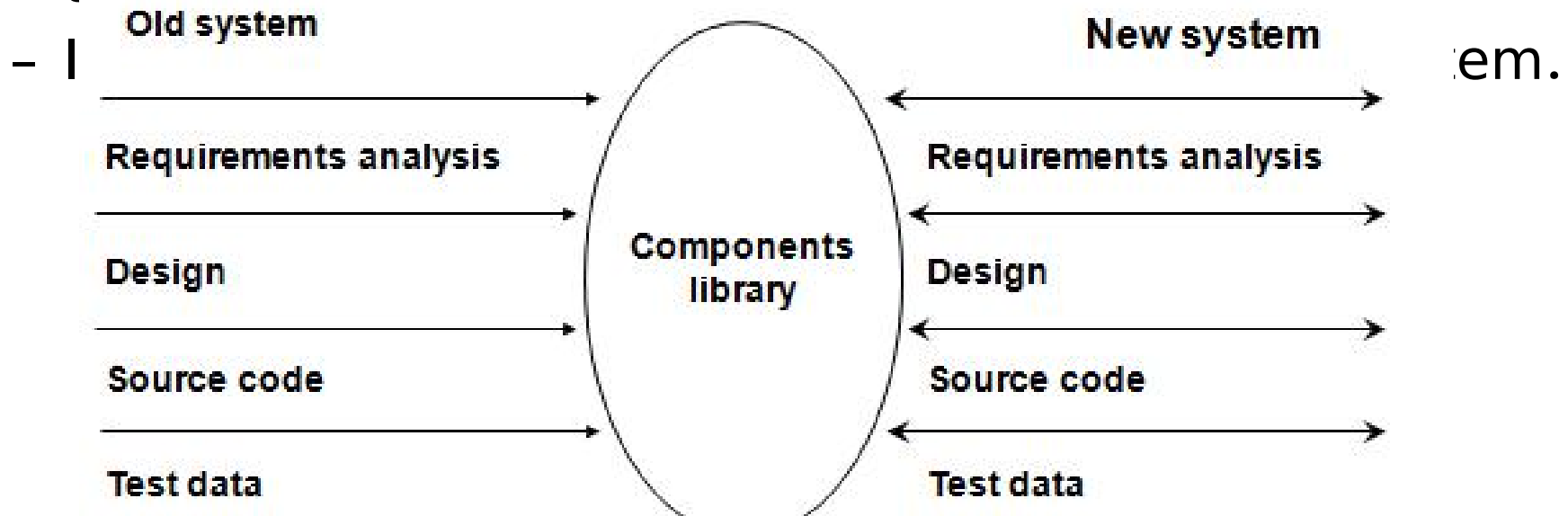
Documentations of each stage (requirement, design, coding, testing) is modified the system is redesigned

- attempts to control complexity and maintain good design
- Assumes existence of full documentation of a

Reuse Oriented Model

The reuse model has four main steps:

- Identification of the parts of the old system that are candidates for reuse.
- Understanding these system parts.
- Modification of the old system parts appropriate to the new requirements



This model needs a mature reuse culture to

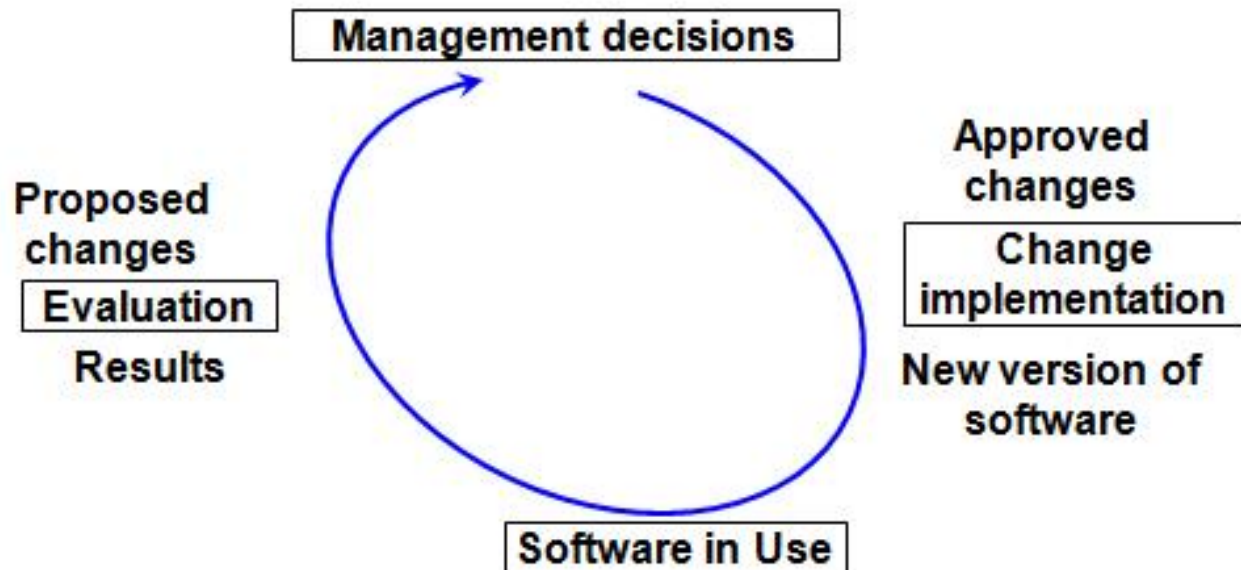
Boehm's Model

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.

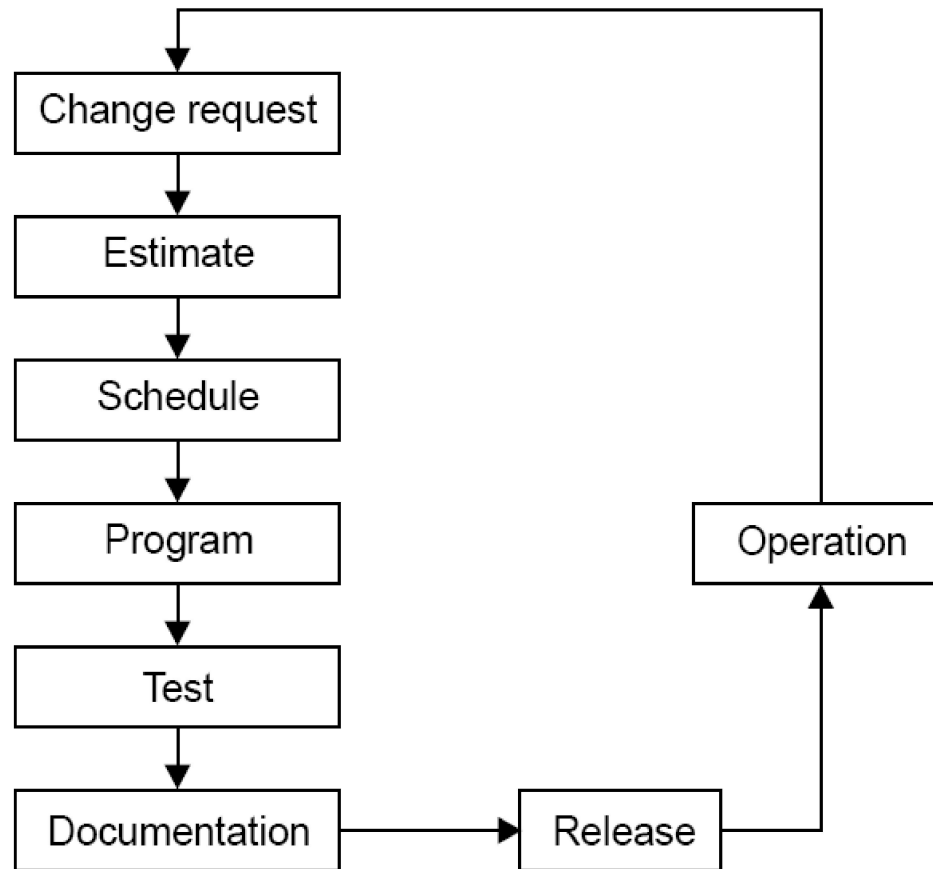
- Management process drives the process
approved changes are determined by applying
partic

a set
are in



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



Fast 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

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

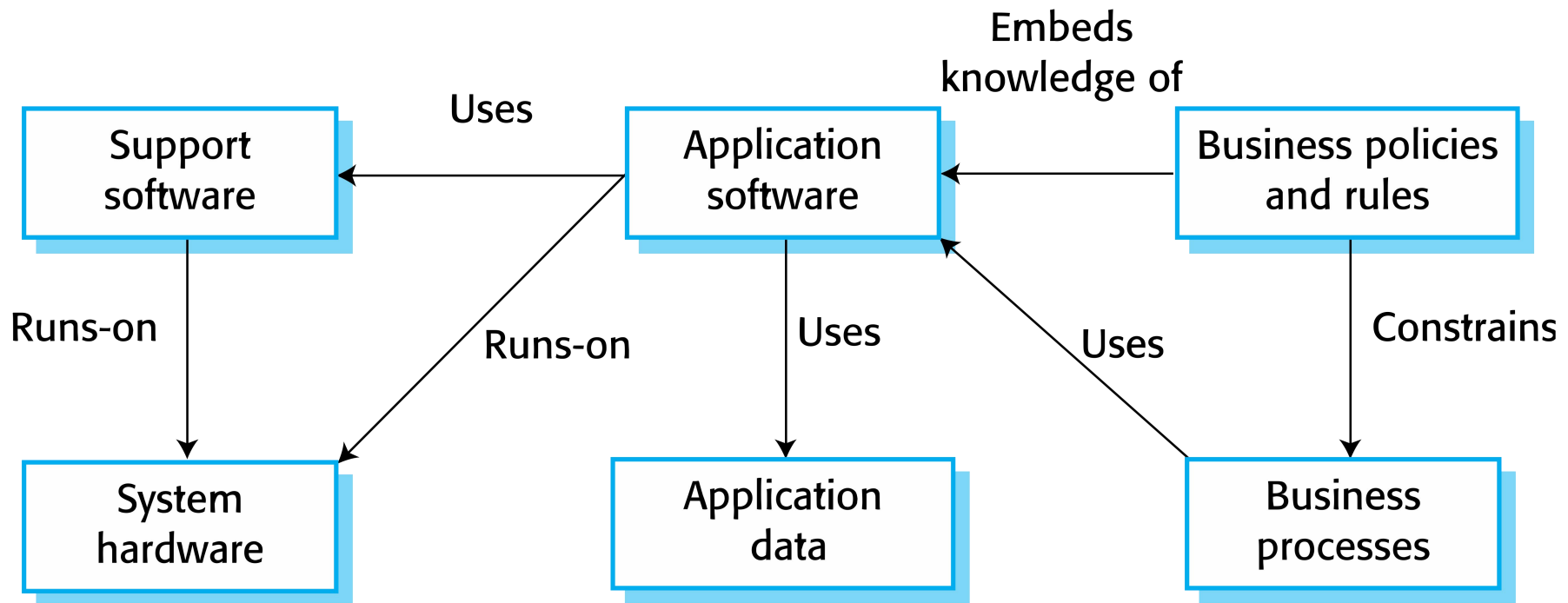
- Poorly structured and documentation not available
- were not designed for change

- none of the original members of the software development team may still be around

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

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

These are data that are processed by the application system. They may be inconsistent, duplicated or held in

Legacy system components...

Business processes

These are processes that are used 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.

Legacy Systems Management ...

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.

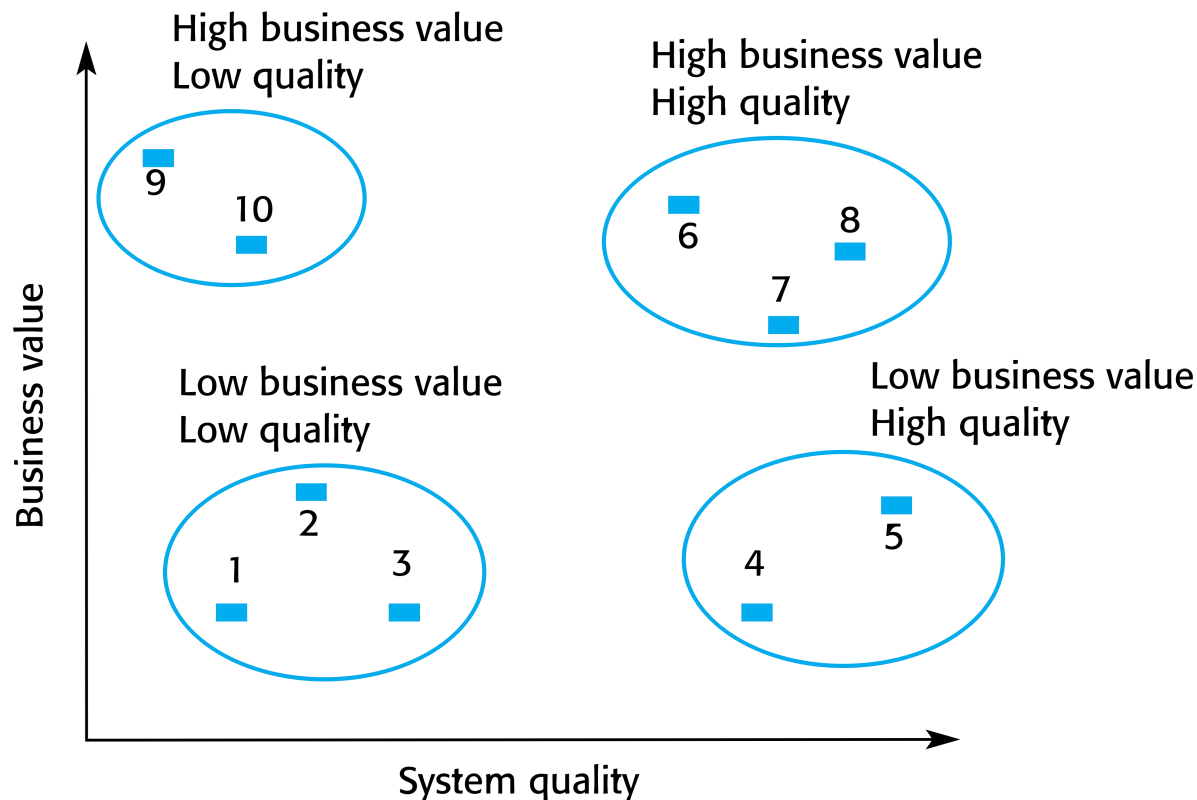
Reengineer the system to improve its maintainability

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

is an approach to help 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 confined to part of the system then re-engineer that part

When hardware or software support becomes obsolete

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

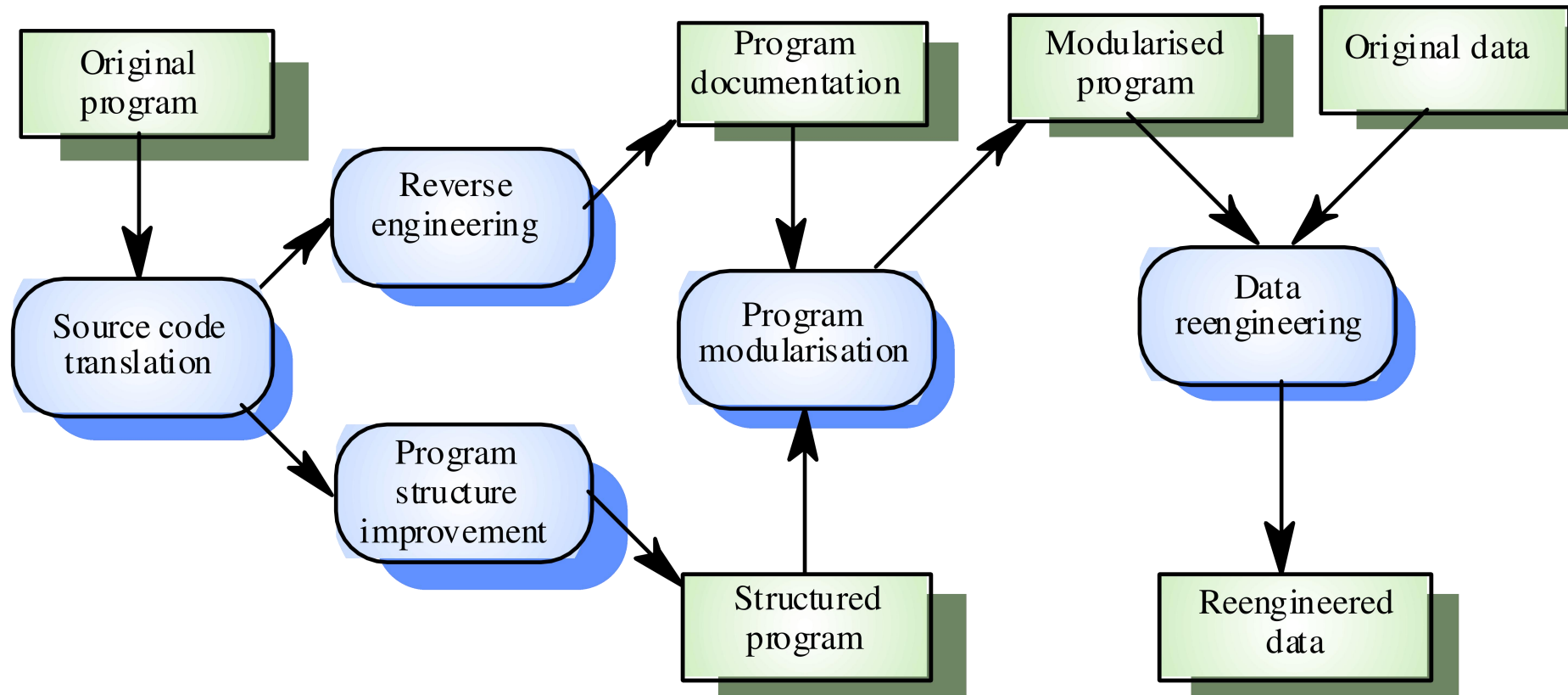
The quality of the software to be re-engineered

The tool support available for re-engineering

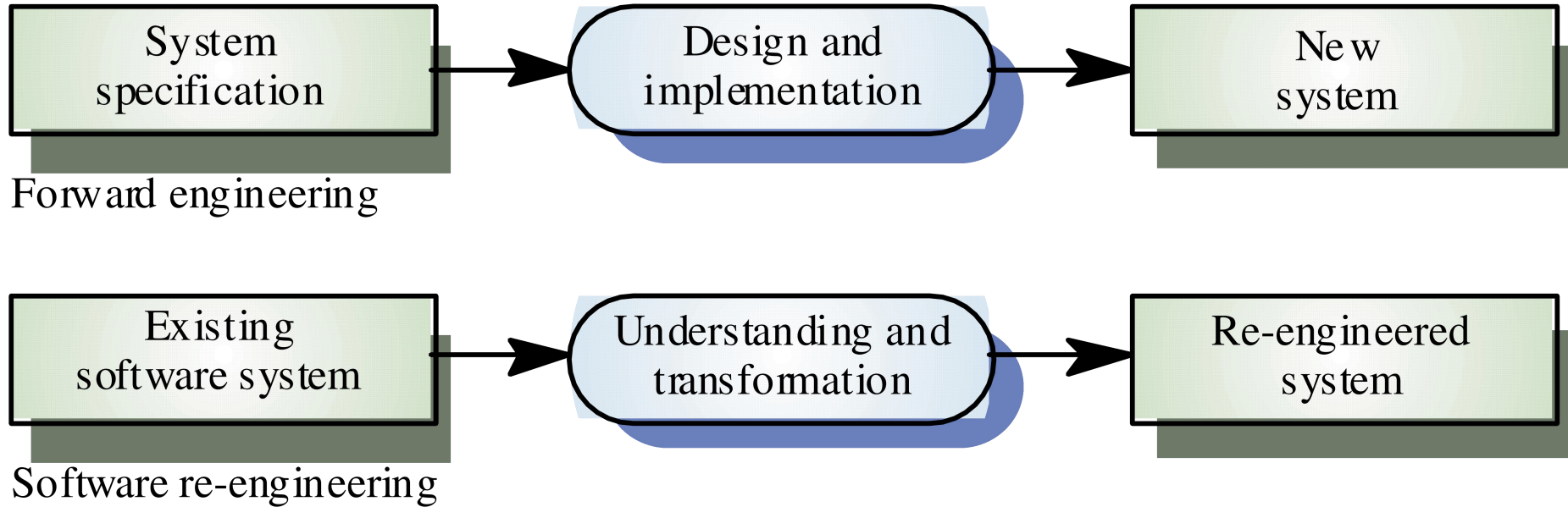
The extent of the data conversion which is required

The availability of expert staff for re-engineering

Reengineering Process



Forward Engineering vs Re-engineering



Forward Engineering

Applies software engineering principles, concepts, and methods to recreate an existing application
The recreated application often extends the capability of the older application by integrating new user and technology requirements

Re-Engineering Approaches

Automated program
restructuring

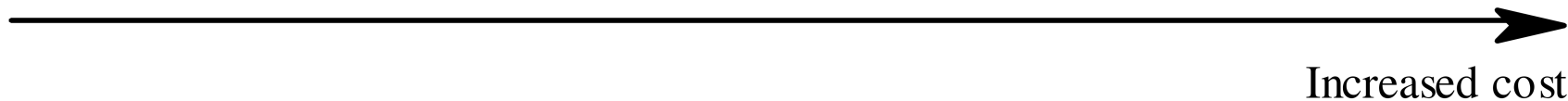
Program and data
restructuring



Automated source
code conversion

Automated restructuring
with manual changes

Restructuring plus
architectural changes



Source code translation

Convert code to a new language. e.g. FORTRAN to C

May be necessary because of **hardware platform update, staff skill shortages, or organisational policy changes**

Only realistic if an automatic translator is

process ...

Reverse engineering

In many legacy systems, the only reliable information about the system is the source code.

Reverse engineering reconstructs requirements, design models, test cases and user documentation consistent with the current state of the source code.

Reverse engineering encompasses several activities: program comprehension, software visualization, static and dynamic slicing, etc.

Reverse engineering is often the initial activity in a reengineering project.

Generates structure charts, entity

process ...

Reverse engineering Activities

- Understanding process
 - source code is analyzed at varying levels of detail
 - System, program, component, pattern, statement
 - to understand procedural abstractions and overall functionality
- Understanding data
 - internal data structures
 - database structure
- User interfaces
 - what are the basic actions (e.g. key strokes or mouse operations) processed by the interface?
 - what is a compact description of the system's behavioral response to these actions?
 - what concept of equivalence of interfaces is

process ...

Program structure improvement

Restructure automatically for understandability;
Maintenance tends to corrupt the structure of a program.

It becomes harder and harder to understand
The program may be automatically restructured to remove unconditional branches
Conditions may be simplified to make them more readable

Example 1:

-- Complex condition

if not (A > B **and** (C < D **or not** (E > F)))...

-- Simplified condition

if (A <= B **and** (C>= D **or** E > F)...

process ...

Program structure improvement (cont'd)

Example 2: Spaghetti Logic

```
Start:  Get (Time-on, Time-off, Time, Setting, Temp, Switch)
        if Switch = off goto off
        if Switch = on goto on
        goto Cntrlr
off:    if Heating-status = on goto Sw-off
        goto loop
on:     if Heating-status = off goto Sw-on
        goto loop
Cntrlr: if Time = Time-on goto on
        if Time = Time-off goto off
        if Time < Time-on goto Start
        if Time > Time-off goto Start
        if Temp > Setting then goto off
        if Temp < Setting then goto on
Sw-off: Heating-status := off
        goto Switch
Sw-on:  Heating-status := on
Switch: Switch-heating
loop:   goto Start
```


process ...

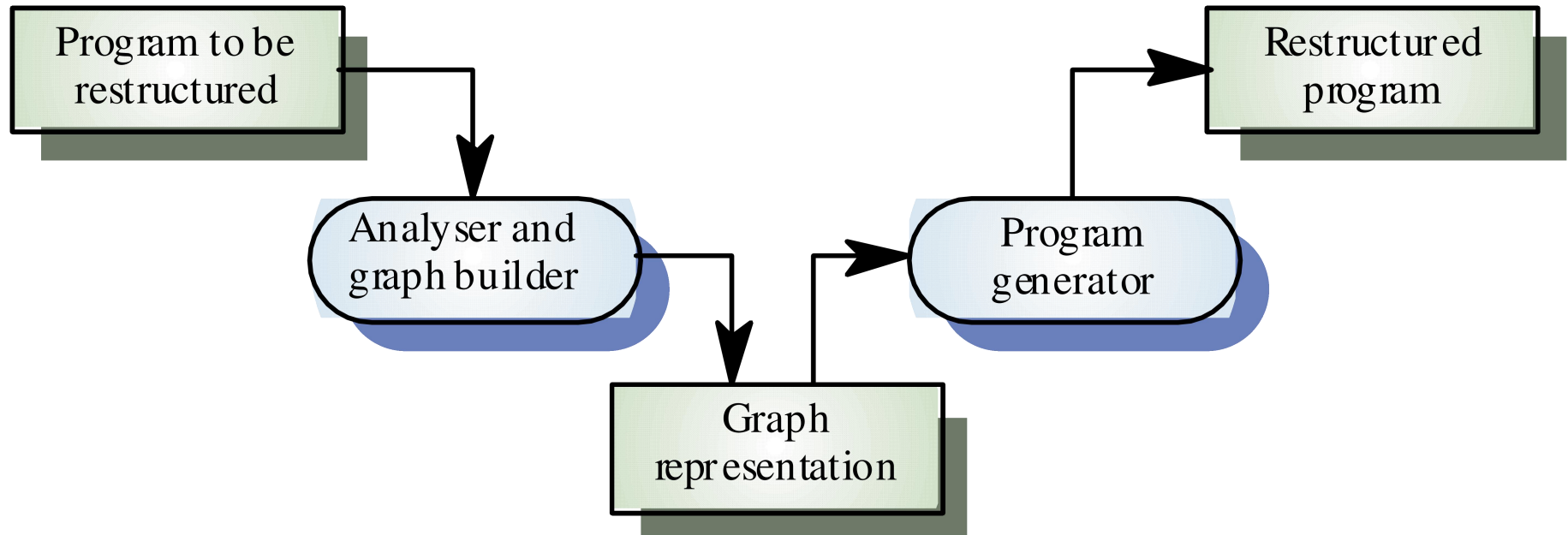
Program structure improvement (cont'd)

Example 2: Spaghetti Logic (Improved one)

```
loop
  -- The Get statement finds values for the given variables from the system's
  -- environment.
  Get (Time-on, Time-off, Time, Setting, Temp, Switch) ;
  case Switch of
    when On => if Heating-status = off then
      Switch-heating ; Heating-status := on ;
    end if ;
    when Off => if Heating-status = on then
      Switch-heating ; Heating-status := off ;
    end if;
    when Controlled =>
      if Time >= Time-on and Time <= Time-off then
        if Temp > Setting and Heating-status = on then
          Switch-heating; Heating-status = off;
        elsif Temp < Setting and Heating-status = off then
          Switch-heating; Heating-status := on ;
        end if;
      end if ;
    end case ;
  end loop ;
```

process ...

Automatic Program Restructuring



Problems with re-structuring are: Loss of comments, Loss of documentation and Heavy computational demands

Restructuring doesn't help with poor modularisation where related components are dispersed throughout the code

The understandability of data-driven programs may

process ...

Program modularization

Related parts of the program are grouped together and, where appropriate, redundancy is removed.

In some cases, this stage may involve architectural transformation

Different types of module may be created during the program modularisation process.

- Data abstractions

- Hardware modules

- Functional modules

- Process support modules

Program modularisation is usually carried

process ...

Data reengineering

The storage, organisation and format of the data processed by legacy programs may have to evolve to reflect changes to the software.

The process of analysing and re-organising the data structures and, sometimes, the data values in a system to make it more understandable is called data re-engineering

In principle, data re-engineering should not be necessary if the functionality of a system is unchanged

In practice, however, there are a number of reasons why you may have to modify the data as well as the programs in a legacy system

Refactoring

Refactoring is the process of making improvements to a program to slow down degradation through change.

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.

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.

‘Bad smells’ in program code decided when to refactor

- 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

Refactoring...

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