# Software maintenance and reverse engineering

# صيانة البرمجيات والهندسة العكسية

د. عدي المعايطه

# Chapter 1: Introduction to the basic concepts

# What is software maintenance?

- The discipline concerned with changes related to a software system after delivery is traditionally known as **software maintenance**.

- There is an increasing reliance on software systems and it is ever more important that such systems do the job they are intended to do, and do it well.

- it is vital that systems are useful. If they fail to be useful, they will not be accepted by users and will not be used.

# Important terms

- **Evolution** - a process of continuous change from a lower, simpler, or worse to a higher, more complex, or better state.
- **Maintainability** - the ease with which maintenance can be carried out.
- **Maintenance** - the act of keeping an entity in an existing state of repair, efficiency, or validity; to preserve from failure or decline.
- **Software** - the programs, documentation and operating procedures by which computers can be made useful to man.

# Important terms continue

- **Software maintenance** - modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment

# Software

- **software** comprises not only programs - source and object code - but also documentation of any facet of the program, such as requirements analysis, specification, design, system and user manuals, and the procedures used to set up and operate the software system.

# Measuring maintainability

- The **maintainability** of a software system is something that is notoriously difficult to quantify.
- Certain aspects of systems can be measured. For example, there are several different ways of measuring complexity. Specific features such as interoperability or adherence to standards are also significant.
- However, there is no simple overall **'maintainability factor'** that can be calculated.
- recognizing the features and traits that make a system easy to maintain is one of the major attributes of a good software maintenance engineer.

Table 1.1 Components of a software system

| Software Components | Examples | | |
|---|---|---|---|
| Program | 1 Source code<br>2 Object code | | |
| Documentation | 1 Analysis / specification: | (a)<br>(b)<br>(c) | Formal specification<br>Context diagram<br>Data flow diagrams |
| | 2 Design: | (a)<br>(b) | Flowcharts<br>Entity-relationship charts |
| | 3 Implementation: | (a)<br>(b) | Source code listings<br>Cross-reference listings |
| | 4 Testing: | (a)<br>(b) | Test data<br>Test results |
| Operating procedures | 1 Instructions to set up and use the software system<br>2 Instructions on how to react to system failures | | |

# How New Development and Maintenance Activities Differ

- Although maintenance could be regarded as a continuation of new development there is a fundamental difference between the two activities.
- **New development** is, within certain constraints, done on a green field site.
- **Maintenance** must work within the parameters and constraints of an existing system.

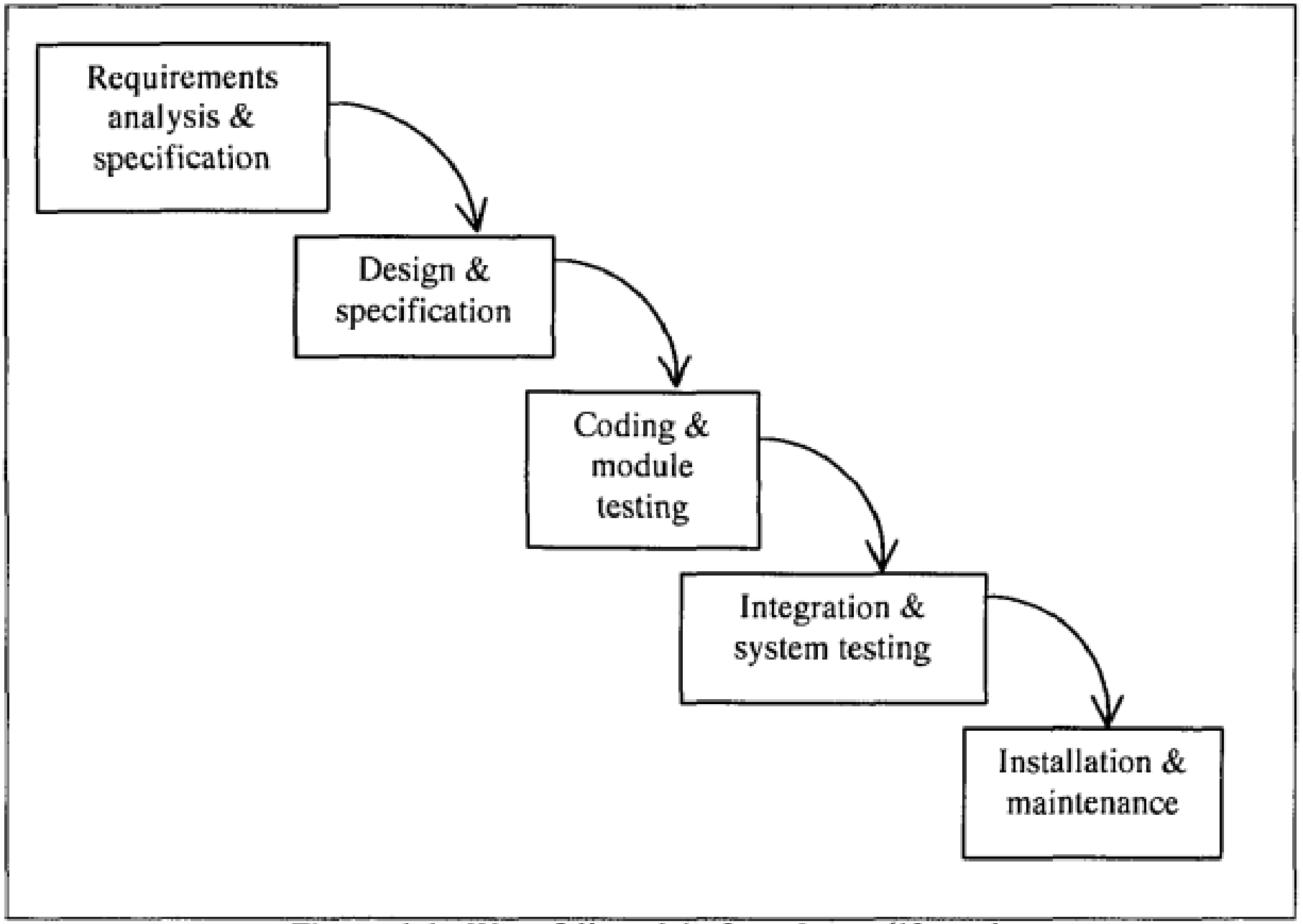


**Figure 1.1** Waterfall model of a software life cycle

# Impact analysis

- Before undertaking any system development or maintenance work, an **impact analysis** should be carried out to determine the ramifications of the new or modified system upon the environment into which it is to be introduced.
- The impact of introducing a specific feature into a system will be very different if it is done as a maintenance activity, as opposed to a development activity.
- It is the constraints that the existing system imposes on maintenance that give rise to this difference.

# Using impact analysis information

- This information is then used to:
- (i) work out how the change can be accommodated.
- (ii) predict the potential **ripple effect** of the change.
- (iii) determine the skills and knowledge required to do the job.

# Difference between new development and software maintenance

- To explain the difference between new development and software maintenance, Jones provides an interesting analogy where he likens the addition of functional requirements to a live system, to the addition of a new room to an existing building:

"*The architect and the builders must take care not to weaken the existing structure when additions are made. Although the costs of the new room usually will be lower than the costs of constructing an entirely new building, the costs per square foot may be much higher because of the need to remove existing walls, reroute plumbing and electrical circuits and take special care to avoid disrupting the current site* "

# 1.6 Why software maintenance is needed

- There are a number of factors that provide the motivation for maintenance:

1. **To provide continuity of service: Systems need to keep running.**
   - For example, software controlling aero planes in flight or train signaling systems cannot be allowed just to stop if an error occurs.
   - Unexpected failure of software can be life threatening. Many facets of daily life are now managed by computer.
   - There can be severe consequences to system failure such as serious inconvenience or significant financial implications.
   - Maintenance activities aimed at keeping a system operational include bug-fixing, recovering from failure, and accommodating changes in the operating system and hardware.

# Why software maintenance is needed continue

**2. To support mandatory upgrades:**

- This type of change would be necessary because of such things as amendments to government regulations e.g. changes in tax laws will necessitate modifications in the software used by tax offices.
- Additionally, the need to maintain a competitive edge over rival products will trigger this kind of change.

# Why software maintenance is needed continue

**3. To support user requests for improvements:**

- On the whole, the better a system is, the more it will be used and the more the users will request enhancements in functionality.
- There may also be requirements for better performance and customization to local working patterns.

# Why software maintenance is needed continue

**4. To facilitate future maintenance work:**

- It does not take long to learn that shortcuts at the software development stage are very costly in the long run.
- It is often financially and commercially justifiable to initiate change solely to make future maintenance easier.
- This would involve such things as code and database restructuring, and updating of documentation.

# Change never ends!

- **If a system is used, it is never finished because it will always need to evolve to meet the requirements of the changing world in which it operates.**

# 1.7 Maintaining Systems Effectively

- In order to maintain systems effectively, a good grounding in the relevant theory is essential and certain skills must be learnt.
- Software maintenance is a key discipline, because it is the means by which systems remain operational and cope efficiently in a world ever more reliant on software systems.
- Maintenance activities are far-reaching. The maintenance practitioner needs to understand the past and appreciate future impact. As a maintenance engineer, you need to know whether you are maintaining a system that must operate for the next five minutes or the next five decades.

# 1.8 Categorizing Software Change

- **Software change may be classified under the following categories**

1. Modification initiated by defects in the software.
2. Change driven by the need to accommodate modifications in the environment of the software system.
3. Change undertaken to expand the existing requirements of a system.
4. Change undertaken to prevent malfunctions.

- The categorizing of software change is vital in understanding when and how to make changes. how to assign resources and how to priorities requests for change.

# Maintainability

Dr. Adi Maaita

# Maintainability

- **Maintainability** focuses on the ease with which a software system can be maintained.
- Maintenance of a software system takes place as changes are made to it after the software is in operation.
- Maintenance is necessary to preserve the value of the software over time.
- Change is constant in the real world. Sometimes it is expected and can be planned for, while other times it is not. Either way, it is inevitable that software systems will experience change.
- With the knowledge that change is unavoidable, it is important to build maintainable systems.

# Maintainability continue

- Decades ago, the greater part of a software project's costs went into software development.
- However, over the years there has been a shift in the cost ratio from development to maintenance.
- Code that is easy to maintain allows maintenance work to be completed more quickly, and in turn will help to keep maintenance costs down. When a developer is writing code, he or she has to take into consideration not just the end user of the software, but also those who will be maintaining it.
- Even if the original developer ends up also being responsible for the maintenance of a particular piece of code, consider the fact that the developer could leave the organization.
- Further, a developer may have a need to revisit their own code after some time passes, only to have forgotten the intricacies of it.
- In some cases, a developer may not even remember at first that they were the original developer of the code! Maintainable code benefits whoever needs to maintain it, even if it is the original developer.
- Maintainability also affects how easily a software system can be reverse-engineered.
- There may be a need for a software system to be reverse-engineered, possibly so that it can be migrated to a newer technology.
- An architecture that exhibits a high level of modifiability will be easier to understand and reason about, making it easier to reverse-engineer.

# Types of software maintenance

- Software maintenance is performed for a variety of reasons, such as correcting defects, improving quality in some way, or meeting new requirements. As a result, software maintenance work can be categorized into the following different types of software maintenance:
- Corrective
- Perfective
- Adaptive
- Preventive maintenance

# Corrective maintenance

- **Corrective maintenance** is the work involved in analyzing and fixing defects in the software. Although it isn't the only kind of maintenance, it is the type that people associate most with maintenance work. Defects might be found internally, or by users in production. The severity and priority of defects vary depending on the nature of the bug.
- Severity represents the level of impact the bug has on the operation of the software. Organizations have various classification systems for severity, but categories such as *critical*, *high*, *medium*, and *low* are common examples.
- The priority of a defect is the order in which it will be fixed. Typically, the higher the priority, the quicker it will be fixed. Like severity, organizations may have different classification systems for priority, but categories such as *high*, *medium*, and *low* are common. Another common classification system is P0, P1, P2, P3, and P4, with P0 being the highest priority. P0 defects are critical and are considered *blockers*. A release will be put on hold until all P0 defects are fixed.
- Maintainability can be measured by the time it takes to analyze and fix a particular defect. Higher levels of maintainability allow these tasks to be completed in a shorter amount of time.

# Perfective maintenance

- **Perfective maintenance** is necessary when the software needs to implement new or updated requirements. These types of changes are mostly focused on the functionality of the software. An example of perfective maintenance is a new enhancement to the software system.
- Software that has a higher level of maintainability will allow for perfective changes to be made with less effort, and therefore at a lower total cost.

# Adaptive maintenance

- **Adaptive maintenance** is defined as the work required to adapt a software system to changes in the software environment. Examples of this may be to adapt the software system for a new **operating system** (**OS**), a new version of the same OS, or to use a new **database management system** (**DBMS**).
- The duration of time it takes to adapt a software system to changes in the environment is a measure of the maintainability of the software.

# Preventive maintenance

- The goal of **preventive maintenance** tasks is to prevent problems in the future, by increasing quality. This may include improving quality attributes of the software system, such as increasing maintainability and reliability.
- Preventive maintenance may take place to make maintenance easier in the future. An example of this is refactoring a software component to make it less complex.

# Modifiability

- **Modifiability** is one aspect of maintainability. Modifiability is the ease with which changes can be made to the software without introducing defects or reducing quality. It is an important quality attribute because there are a number of reasons why software needs to be changed.
- Some software can remain useful in production for years, or even decades. Inevitably, that code will need to be modified for the different types of maintenance that were described previously. The time required from when a change is specified until it is deployed is an indication of the modifiability of the system.
- In today's world, agile software development methodologies are the most common. These software projects embrace change. In addition to new functionality, each iteration of the project can involve changes to existing code. Improving modifiability is not just beneficial for maintenance but also for the entire development of the software.

# Extensibility and flexibility

- **Extensibility** and **flexibility** are additional characteristics related to maintainability. The level of extensibility reflects how easy it is to extend or enhance the software system. Software systems that are designed to be extensible take future growth into consideration by anticipating the need to add new functionality.
- Flexibility is similar but mostly focuses on how easy it is to change a capability so that it can be used in a way that wasn't originally designed. Both extensibility and flexibility are characteristics that dictate the level of ease with which someone can perform perfective maintenance.

# Scope of modifications

- Not all changes to software are equal, and the scope of a particular modification is a factor in how difficult the change will be to implement. The larger and the more complex the modification, the greater the effort to complete it.
- In addition to size, if the changes require architecture level changes, that will increase the level and scale of effort involved. Some components and their interactions may need to be refactored extensively for large changes.

# Chapter 2: The maintenance framework

# 2.1 Introduction

- Software maintenance is not an activity carried out in a vacuum. It affects and interacts with the environment within which it is carried out.
- It is changes and interactions in the surrounding environment that bring about the need for change.
- Understanding the framework, and the relationship between the factors comprising this framework allows prediction of problem areas and the ability to avoid them.

# 2.2 Definitions

- **Environment** - the totality of conditions and influences which act from outside upon an entity.
- **Environmental factor** - an agent which acts upon the entity from without and influences its form or operation.
- **Framework** - a set of ideas, conditions, or assumptions that determine how something will be approached, perceived, or understood.
- **Information gap** - this is the discrepancy between the body of knowledge that system users and system maintainers possess and the body of knowledge that each needs to have in order to satisfy a request for change.
- **Maintenance challenge** - the need to keep systems running. Historically the challenge has been to keep mechanical systems operational after physical wear and tear on components. Software is not subject to physical wear and tear, but to influences less easy to identify and address. Thus, the maintenance challenge when applied to software is a far more complex task than when applied to mechanical systems.
- **Maintenance personnel** - the individuals involved in maintaining a software product.

# Definitions continue

- **Maintenance process** - any activity carried out, or action taken, either by a machine or maintenance personnel during software maintenance.
- **Operating environment** - all software and hardware systems that influence or act upon a software product in any way.
- **Organizational environment** - all non-software- or non-hardware related environmental factors.
- **Safety-critical** - a system where failure could result in death, injury or illness, major economic loss, environmental or property damage.
- **Safety-related** - a system where failure could significantly increase the risk of injury or damage.
- **Software maintenance framework** - the context and environment in which software maintenance activities are carried out.

# 2.3 A Software Maintenance Framework

- The requirement for software systems to evolve in order to accommodate changing user needs contributes to the high maintenance costs.
- There are other factors which contribute indirectly by hindering maintenance activities.
- A Software Maintenance Framework (SMF)1 will be used to discuss some of these factors.
- The elements of this framework are the user requirements, organisational and operational environments, maintenance process, software product, and the maintenance personnel (Table 2.1).
- To understand the sources of the software maintenance challenge, you need an understanding of these components, their characteristics and the effect of their interactions.

**Table 2.1** Components of a software maintenance framework

| Component | Feature |
|---|---|
| 1. User requirements | ▪ Requests for additional functionality, error correction and improving maintainability<br>▪ Request for non-programming-related support |
| 2. Organisational environment | ▪ Change in policies<br>▪ Competition in the market place |
| 3. Operational environment | ▪ Hardware innovations<br>▪ Software innovations |
| 4. Maintenance process | ▪ Capturing requirements<br>▪ Creativity and undocumented assumptions<br>▪ Variation in programming practice<br>▪ Paradigm shift<br>▪ 'Dead' paradigms for 'living' systems<br>▪ Error detection and correction |
| 5. Software product | ▪ Maturity and difficulty of application domain<br>▪ Quality of documentation<br>▪ Malleability of programs<br>▪ Complexity of programs<br>▪ Program structure<br>▪ Inherent quality |
| 6. Maintenance personnel | ▪ Staff turnover<br>▪ Domain expertise |

# 2.3.1 Components of the Framework

- User
- Environment
- Maintenance Process
- Software Product
- Maintenance Personnel

# User

- The user in this context refers to individuals who use the system, regardless of their involvement in its development or maintenance.
- There are several reasons why there could be requests for modification of a system after it becomes operational.
- The implementation of such modifications may necessitate:

  (i) 'progressive' work to refine existing functions or to introduce new features; and

  (ii) 'anti-regressive' work to make programs well structured, better documented, more understandable and capable of further development.
- Regardless of the degree of success of a system, it has a propensity to evolve (while it remains operational) in order to support users' changing needs.

# Environment

- Essentially, the environments affecting software systems are the operating environment and the organizational environment.
- Typical environmental factors within these are business rules, government regulations, work patterns, software and hardware operating platforms.

**Operating environment**

- Examples of factors within the operating environment are innovations in hardware and software platforms:
- **Hardware innovations:** The hardware platform on which a software system runs may be subject to change during the lifetime of the software. Such a change tends to affect the software in a number of ways. For example, when a processor is upgraded, compilers that previously produced machine code for that processor may need to be modified.
- **Software innovations:** Like hardware, changes in the host software may warrant a corresponding modification in the software product. Operating systems, database management systems and compilers are examples of host software systems whose modification may affect other software products.

# Environment continue

## Organizational Environment

- Examples of the entities of the organizational environment are policies and imposed factors of business and taxation, and also competition in the marketplace:
- **Change in policies:** Many information systems have business rules and taxation policies incorporated into their program code. A business rule refers to the procedures used by an organization in its day-to-day operation. A change in the business rule or taxation policy leads to a corresponding modification of the programs affected. For example, changes to Value Added Tax (VAT) rules necessitate modification of programs that use VAT rules in their computations.
- **Competition in the marketplace:** Organizations producing similar software products are usually in competition. From a commercial point of view, organizations strive towards having a competitive edge over their rivals (by securing a significant proportion of the market for that product). This can imply carrying out substantial modifications so as to maintain the status of the product - reflected in the level of customer satisfaction - or to increase the existing 'client base'.

# Maintenance Process

- The maintenance process itself is a major player in the software maintenance framework.
- Significant factors are:
- **Capturing change requirements**: This is the process of finding out exactly what changes are required. It poses a lot of problemsVariation in programming practice
- **Variation in programming practice**: This refers to differences in approach used for writing and maintaining programs.
- **Paradigm shift:** This refers to an alteration in the way we develop and maintain software
- **'Dead' paradigms for 'living' systems:** Many 'living systems' are developed using 'dead paradigms’
- **Error detection and correction:** 'Error-free' software is non-existent.

# Software Product

- Aspects of a software product that contribute to the maintenance challenge include:
- **Maturity and difficulty of the application domain:** The requirements of applications that have been widely used and well understood are less likely to undergo substantial modification on installation than those that are still in their infancy.
- **Quality of the documentation:** The lack of up-to-date systems' documentation is one of the major problems that software maintainers face.
- **Malleability of the programs:** The malleable or 'soft' nature of software products makes them more vulnerable to undesirable modification than hardware items.

# Maintenance Personnel

- It should never be forgotten that people are involved at all stages of the maintenance process and as such are components within the maintenance framework. Maintenance personnel include maintenance managers, analysts, designers, programmers and testers. The personnel aspects that affect maintenance activities include the following:
- Staff turnover
- Domain expertise
- Working practices

# 2.3.2 Relations Between the Maintenance Factors

- It is a change in, or interaction between, the factors discussed above that causes software products to evolve and hence causes maintenance problems to arise.
- Three major types of relation and interaction that can be identified are product/environment, product/user and product/maintenance personnel (Figure 2.2).
- **Relation between product and environment:** A software product does not exist in a vacuum, rather it can be seen as an entity which is hosted by its organizational and operational environments.
- A software product inherits changes in the elements of these environments - taxation policies, software innovations, etc

# Relations Between the Maintenance Factors

continue

- **Relation between product and user:** One of the objectives of a software product is to serve the needs of its users.
- The needs of the users change all the time. In order for the system to stay useful and acceptable it has to change to accommodate these changing requirements.

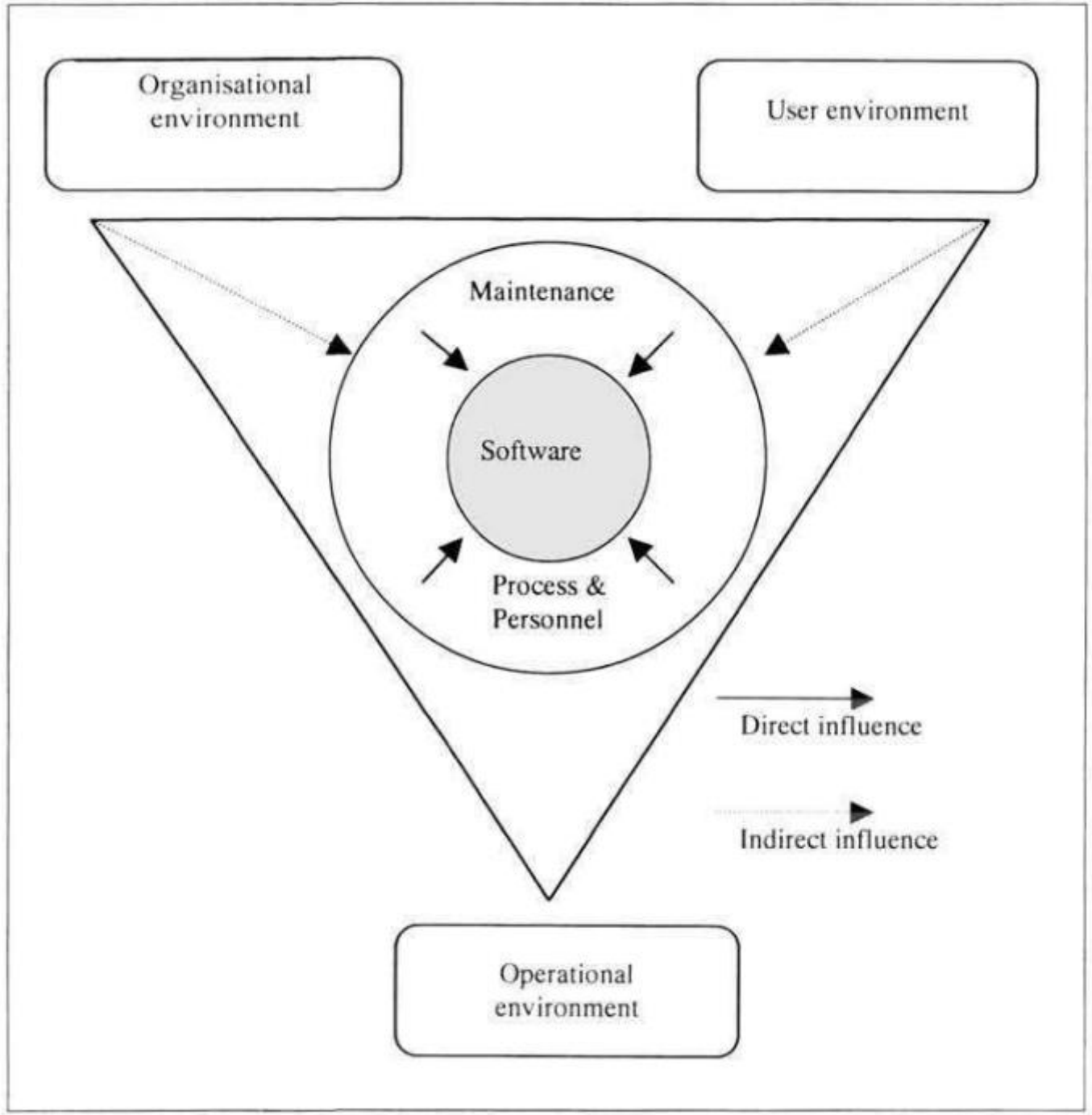


**Figure 2.2** Inter-relationship between maintenance factors

# Relations Between the Maintenance Factors

continue

- **Interaction between personnel and product:** The maintenance personnel who implement changes are themselves the conduit through which change is implemented.
- Changes in factors such as user requirements, the maintenance process, or the organizational and operational environments, will bring about the need for a change in the software product.
- However, the software product will not be affected until the maintenance personnel implement the changes. The type of maintenance process used and the nature of the maintenance personnel themselves, will affect the quality of the change.

# Relations Between the Maintenance Factors

continue

- The effects of the above relations and interactions are reflected in changes in some fundamental characteristics of the software product, such as its size and complexity.
- **The size** - measured in number of program modules or lines of code - tends to increase with an increase in the functionality being offered by the system.
- **The complexity** - measured in terms of the difficulty of understanding the program source code - tends to increase as the programs are modified.

# Summary

- The key points covered in this chapter are: Software maintenance is carried out within, and interacts with, the world around it.
- The software maintenance framework comprises the users, the operational and organizational environments, the maintenance process, the software product and the maintenance personnel.
- Interactions between the components of the software maintenance framework are the driving forces behind the need for software change.
- The major types of interaction are between the product and its environment, the product and its users, the product and the maintenance personnel.
- A good understanding of the context in which software maintenance is carried out is vital in effective implementation of software change.
- The maintenance framework gives the context in which maintenance activities are carried out. The next chapter goes on to explore in detail the fundamentals of software change.