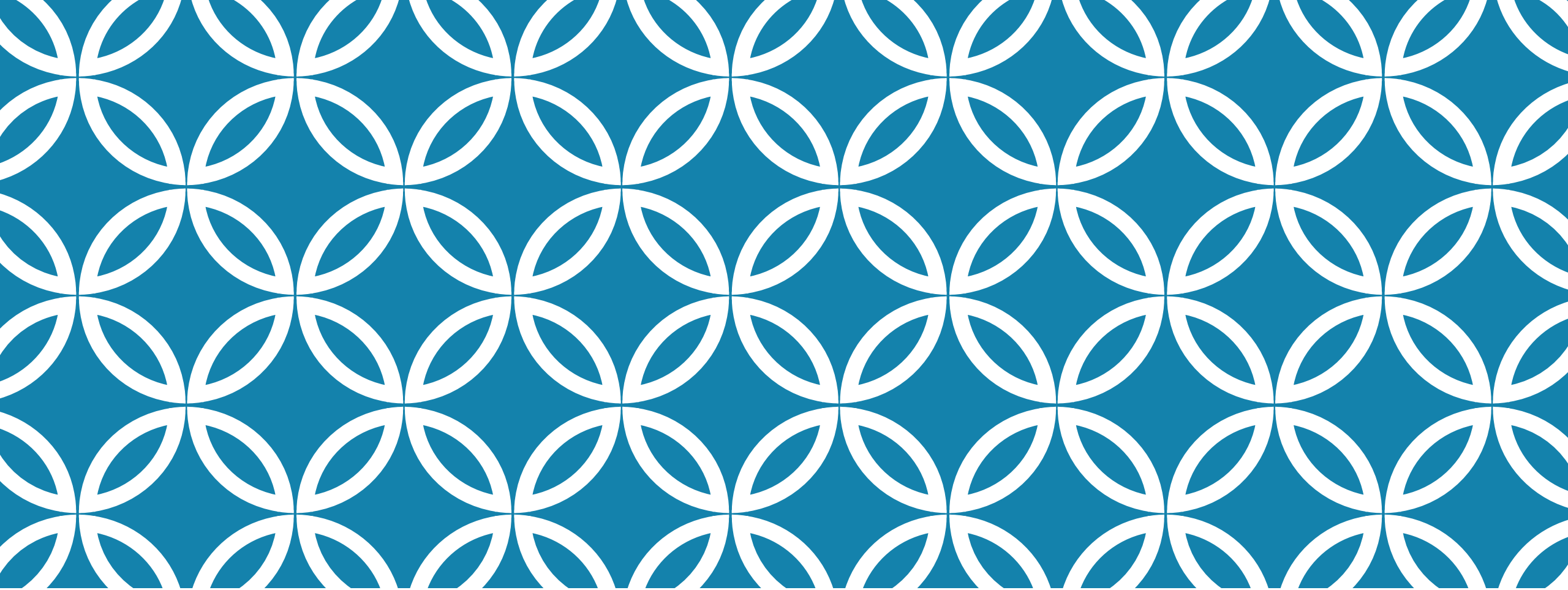


# IDENTIFY TEST REQUIREMENTS

CHAPTER 01

# WHAT IS SOFTWARE?

- Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

- The IEEE definition of software, which is almost identical to the ISO definition
(ISO, 1997, Sec. 3.11 and ISO/IEC 9000-3 Sec. 3.14), lists the following four components of software:
  - Computer programs (the “code”)
  - Procedures
  - Documentation
  - Data necessary for operating the software system.

# WHAT IS SOFTWARE?

- Computer programs (the “code”) are needed because, obviously, they activate the computer to perform the required applications.
- Procedures are required, to define the order and schedule in which the programs are performed, the method employed, and the person responsible for performing the activities that are necessary for applying the software.
- Various types of documentation are needed for developers, users and maintenance personnel.
- Data including parameters, codes and name lists that adapt the software to the needs of the specific user are necessary for operating the software.

- "We've used the Panol HR software in our Human Resources Department for about three years and we have never had a software failure."
- "I started to use Panol HR two months ago; we had so many failures that we are considering replacing the software package."
- "We have been using the same software package for almost four years. We were very satisfied throughout the period until the last few months, when we suddenly faced several severe failures.

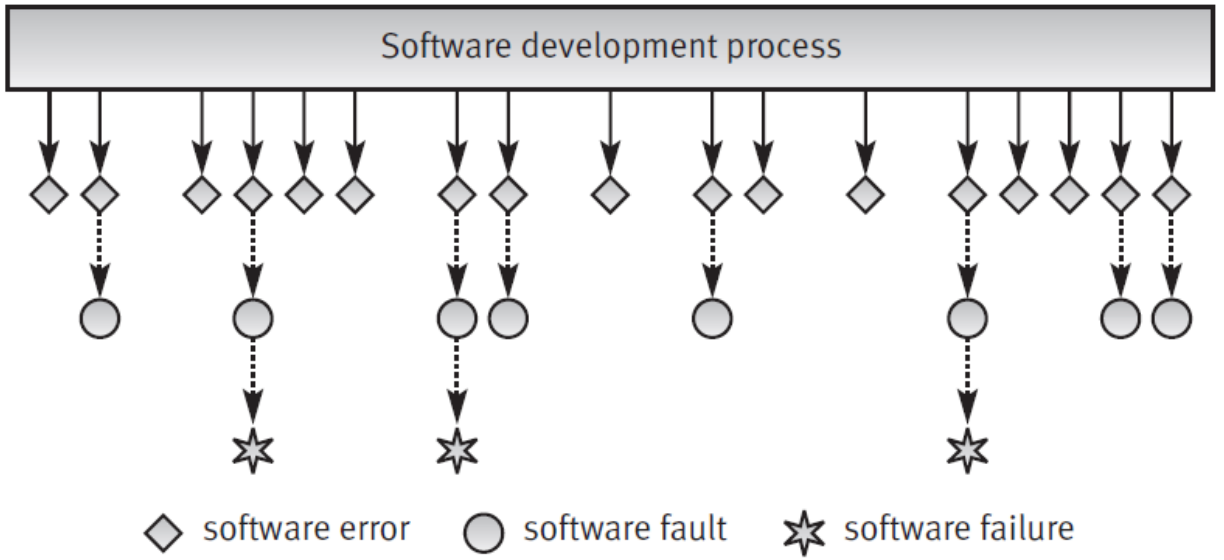
**Can a software package that successfully served an organization for a long period "suddenly" change its nature (quality) and become "bugged"?**

# THE ORIGIN OF SOFTWARE FAILURES

- The origin of software failures lies in a software error made by a programmer. An error can be a grammatical error in one or more of the code lines, or a logical error in carrying out one or more of the client's requirements. However, not all software errors become software faults
- However, not all software errors become software faults. In other words, in some cases, the software error can cause improper functioning of the software in general or in a specific application. In many other cases, erroneous code lines will not affect the functionality of the software as a whole; in a part of these cases, the fault will be corrected or "neutralized" by subsequent code lines.
- We are interested mainly in the software failures that disrupt our use of the software. This requires us to examine the relationship between software faults and software failures. Do all software faults end with software failures? Not necessarily: a software fault becomes a software failure only when it is "activated".

# CASE STUDY

*"Pharm-Plus", a software package developed for the operations required of a pharmacy chain, included several software faults, such as the following: (a) The chain introduced a software requirement to avoid the current sale of goods to customers whose total debts will exceed $200 upon completion of the current sale. Unfortunately, the programmer erroneously put the limit at $500, a clear software fault. However, a software failure never occurred as the chain's pharmacies do not offer credit to their customers, that is, sales are cash sales or credit card sales.*



# 09 CLASSIFICATION OF THE CAUSES OF SOFTWARE ERRORS

- As software errors are the cause of poor software quality, it is important to investigate the causes of these errors in order to prevent them.
- A software error can be “code error”, a “procedure error”, a “documentation error”, or a “software data error”.
- It should be emphasized that the causes of all these errors are human, made by systems analysts, programmers, software testers, documentation experts, managers and sometimes clients and their representatives.
- Even in rare cases where software errors may be caused by the development environment (interpreters, wizards, automatic software generators, etc.), it is reasonable to claim that it is human error that caused the failure of the development environment tool.

# 09 CLASSIFICATION OF THE CAUSES OF SOFTWARE ERRORS

1. Faulty requirements definition
2. Client-developer communication failures
3. Deliberate deviations from software requirements
4. Logical design errors
5. Coding errors
6. Non-compliance with documentation and coding instructions
7. Shortcomings of the testing process
8. Procedure errors
9. Documentation errors

# 1. FAULTY DEFINITION OF REQUIREMENTS

The faulty definition of requirements, usually prepared by the client, is one of the main causes of software errors. The most common errors of this type are:

- Erroneous definition of requirements.
- Absence of vital requirements.
- Incomplete definition of requirements.
- Inclusion of unnecessary requirements

# 2. CLIENT-DEVELOPER COMMUNICATION FAILURES

Misunderstandings resulting from defective client–developer communication are additional causes for the errors that prevail in the early stages of the development process:

- Misunderstanding of the client's instructions as stated in the requirement document.
- Misunderstanding of the client's requirements changes presented to the developer in written form during the development period.
- Misunderstanding of the client's requirements changes presented orally to the developer during the development period.
- Misunderstanding of the client's responses to the design problems presented by the developer.
- Lack of attention to client messages referring to requirements changes and to client responses to questions raised by the developer on the part of the developer.

# 3. DELIBERATE DEVIATIONS FROM SOFTWARE REQUIREMENTS

In several circumstances, developers may deliberately deviate from the documented requirements, actions that often cause software errors. The errors in these cases are byproducts of the changes. The most common situations of deliberate deviation are:

- The developer reuses software modules taken from an earlier project without sufficient analysis of the changes and adaptations needed to correctly fulfill all the new requirements.
- Due to time or budget pressures, the developer decides to omit part of the required functions in an attempt to cope with these pressures.
- Developer-initiated, unapproved improvements to the software, introduced without the client's approval, frequently disregard requirements that seem minor to the developer. Such "minor" changes may, eventually, cause software errors.

# 4. LOGICAL DESIGN ERRORS

Software errors can enter the system when the professionals who design the system – systems architects, software engineers, analysts, etc. – formulate the software requirements. Typical errors include:

- Definitions that represent software requirements by means of erroneous algorithms.
- Process definitions that contain sequencing errors.
- Erroneous definition of boundary conditions. For example, the client's requirements stated that a special discount will be granted to customers who make purchases more than three times in the same month. The analyst erroneously defined the software process to state that the discount would be granted to those who make purchases three times or more in the same month.
- Omission of required software system states.
- Omission of definitions concerning reactions to illegal operation of the software system.

# 5. CODING ERRORS

- A broad range of reasons cause programmers to make coding errors. These include misunderstanding the design documentation, linguistic errors in the programming languages, errors in the application of CASE and other development tools, errors in data selection, and so forth.

# 6. NON-COMPLIANCE WITH DOCUMENTATION AND CODING INSTRUCTIONS

- Almost every development unit has its own documentation and coding standards that define the content, order and format of the documents, and the code created by team members.
- The test team will find it more difficult to test the module; consequently, their efficiency is expected to be lower, leaving more errors undetected. Moreover, team members required to correct the detected errors can be expected to encounter greater difficulties when doing so. They may leave some errors only partially corrected, and even introduce new errors as a result of their incomplete grasp of the other team members' work

# 7. SHORTCOMINGS OF THE TESTING PROCESS

Shortcomings of the testing process affect the error rate by leaving a greater number of errors undetected or uncorrected. These shortcomings result from the following causes:

- Incomplete test plans leave untreated portions of the software or the application functions and states of the system.
- Failures to document and report detected errors and faults.
- Failure to promptly correct detected software faults as a result of inappropriate indications of the reasons for the fault.
- Incomplete correction of detected errors due to negligence or time pressures.

# 8. PROCEDURE ERRORS

- Procedures direct the user with respect to the activities required at each step of the process. They are of special importance in complex software systems where the processing is conducted in several steps, each of which may feed a variety of types of data and allow for examination of the intermediate results.

# 9. DOCUMENTATION ERRORS

- The documentation errors that trouble the development and maintenance teams are errors in the design documents and in the documentation integrated into the body of the software. These errors can cause additional errors in further stages of development and during maintenance.

.

# SOFTWARE QUALITY – DEFINITION

*Software quality – IEEE definition*

Software quality is:

1. The degree to which a system, component, or process meets specified requirements.
2. 2. The degree to which a system, component, or process meets customer or user needs or expectations.

# SOFTWARE QUALITY ASSURANCE– DEFINITION

*Software quality assurance – The IEEE definition*

Software quality assurance is:

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
2. 2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

# SOFTWARE QUALITY ASSURANCE VS. SOFTWARE QUALITY CONTROL

Two phrases are constantly repeated within the context of software quality: "**Quality control**" and "**Quality assurance**". Are they synonymous? How are they related?

These two terms represent separate and distinct concepts:

- **Quality control** is defined as "a set of activities designed to evaluate the quality of a developed or manufactured product" in other words, activities whose main objective is the withholding of any product that does not qualify.
- The main objective of **Quality assurance** is to minimize the cost of guaranteeing quality by a variety of activities performed throughout the development and manufacturing processes/stages

# SOFTWARE QUALITY ASSURANCE VS. SOFTWARE QUALITY CONTROL

(1) Quality control and quality assurance activities serve different objectives.

(2) Quality control activities are only a part of the total range of quality assurance activities.

# THE OBJECTIVES OF SQA ACTIVITIES

***Software development (process-oriented):***

1. Assuring an acceptable level of confidence that the software will conform to functional technical requirements.
2. Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements.
3. Initiating and managing of activities for the improvement and greater efficiency of software development and SQA activities. This means improving the prospects that the functional and managerial requirements will be achieved while reducing the costs of carrying out the software development and SQA activities.

***Software maintenance (product-oriented):***

1. Assuring with an acceptable level of confidence that the software maintenance activities will conform to the functional technical requirements.
2. Assuring with an acceptable level of confidence that the software maintenance activities will conform to managerial scheduling and budgetary requirements.
3. Initiating and managing activities to improve and increase the efficiency of software maintenance and SQA activities. This involves improving the prospects of achieving functional and managerial requirements while reducing costs.

# END