ICT-5405

PROJECT MANAGEMENT AND QUALITY ASSURANCE

06 Software Quality Fundamentals

Introduction

- **Software** simply as a set of instructions that make up a program. These instructions are also called the software's **source code**. A set of programs forms an application or a software component of a system with hardware components.
- An **information system** is the interaction between the software application and the information technology (IT) infrastructure of the organization. It is the information system or the system (e.g., digital camera) that clients use.

Definition of Software

1. All or part of the programs, procedures, rules, and associated documentation of an information processing system.
2. Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

ISO 24765 [ISO 17a]

"SOFTWARE" in more detail

- **Programs:** the instructions that have been translated into source code, which have been specified, designed, reviewed, unit tested, and accepted by the clients;
- **Procedures:** the user procedures and other processes that have been described (before and after automation), studied, and optimized;
- **Associated documentation:** all types of documentation that is useful to customers, software users, developers, auditors, and maintainers. Documentation enables different members of a team to better communicate, review, test, and maintain software. Documentation is defined and produced throughout the key stages of the software life cycle;

Firmware

- Combination of a hardware device and computer instructions or computer data that reside as read-only software on the hardware device.

Software Defects Terminology

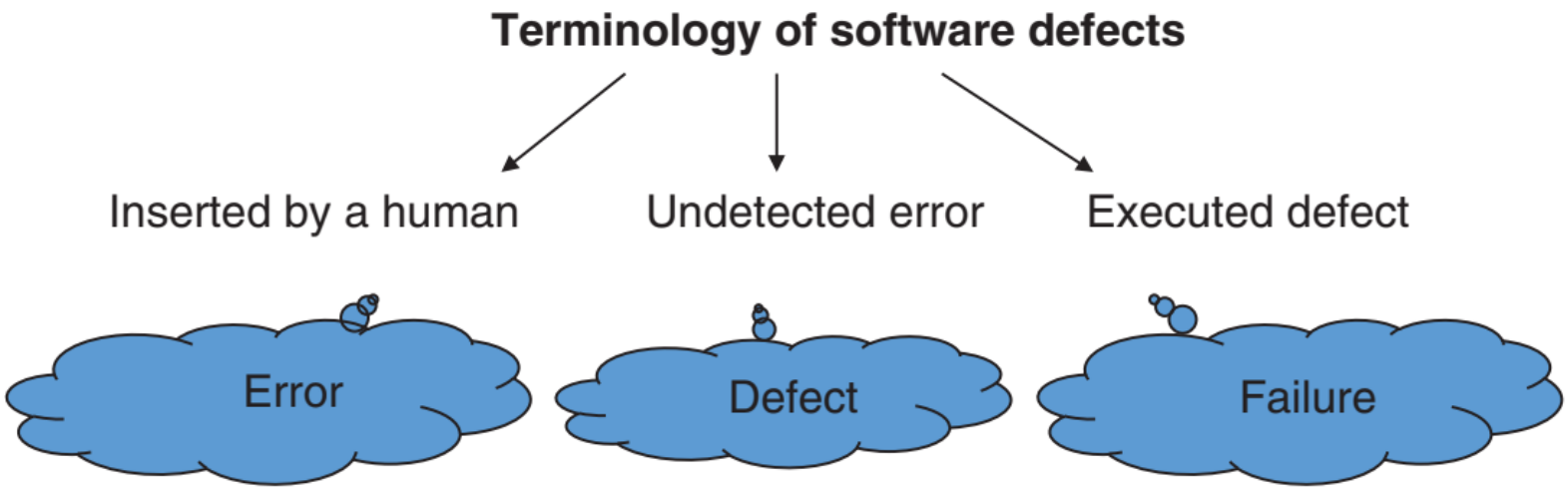
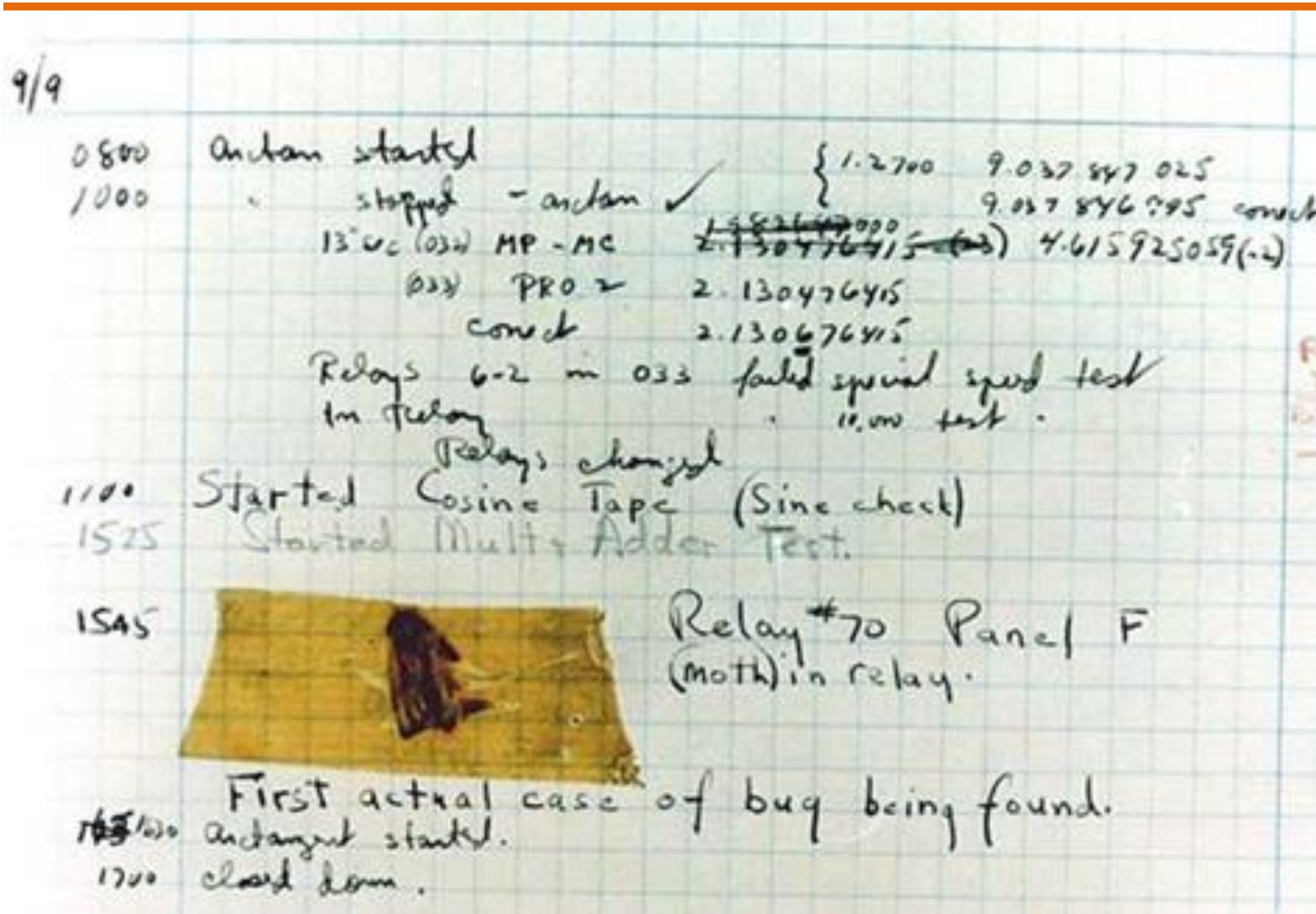


Figure 1.2 Terminology recommended for describing software problems.

Bug

- The word "**BUG**" to refer to failures in the systems that they have developed. A bug refers to a flaw, error, or unexpected behavior in a software application that causes it to produce incorrect or unintended results.
- The first documented case of a **computer bug** involved a moth trapped in a relay of Mark II computer at Harvard University in 1947.
- Grace hopper, the computer operator, pasted the insect into the laboratory log specifying it as the "**First actual case of a bug being found**"


9/9
0800 Arctan started
1000 stopped - arctan ✓
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult+ Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Failure

- A failure (synonymous with a crash or breakdown) is the execution (or manifestation) of a fault in the operating environment.
- A failure is defined as the termination of the ability of a component to fully or partially perform a function that it was designed to carry out.
- The origin of a failure lies with a defect hidden, that is, not detected by tests or reviews, in the system currently in operation. As long as the system in production does not execute a faulty instruction or process faulty data, it will run normally. Therefore, it is possible that a system contains defects that have not yet been executed.

Defects and Error

- **Defects** (synonym of faults) are human errors that were not detected during software development, quality assurance (QA), or testing.
- An **error** can be found in the documentation, the software source code instructions, the logical execution of the code, or anywhere else in the life cycle of the system.

Error, Defect, and Failure

Error

- A human action that produces an incorrect result (ISO 24765)

Defect

- A problem (synonym of fault) which, if not corrected, could cause an application to either fail or to produce incorrect results. (ISO 24765) [ISO 17a].
- An imperfection or deficiency in a software or system component that can result in the component not performing its function, e.g. an incorrect data definition or source code instruction. A defect, if executed, can cause the failure of a software or system component (ISTQB 2011 [IST 11]).

Failure

- The termination of the ability of a product to perform a required function or its inability to perform within previously specified limits (ISO 25010 [ISO 11i]).

Case of Errors, Defects, and Failures

- **Case 1:** A local pharmacy added a software requirement to its cash register to prevent sales of more than $75 to customers owing more than $200 on their pharmacy credit card. The programmer did not fully understand the specification and created a sales limit of $500 within the program. This defect never caused a failure since no client could purchase more than $500 worth of items given that the pharmacy credit card had a limit of $400.

Case of Errors, Defects, and Failures

- **Case 2:** In 2009, a loyalty program was introduced to the clients of American Signature, a large furniture supplier. The specifications described the following business rules: a customer who makes a monthly purchase that is **higher** than the average amount of monthly purchases for all customers will be considered a Preferred Customer. The Preferred Customer will be identified when making a purchase, and will be immediately given a gift or major discount once a month.
- The defect introduced into the system involved only taking into account the average amount of current purchases and not the customer's monthly history. At the time of the software failure, the cash register was identifying far too many Preferred Clients, resulting in a loss for the company.

Case of Errors, Defects, and Failures

- **Case 3:** Peter tested Patrick's program when Patrick was away. He found a defect in the calculation for a retirement savings plan designed to apply the new tax-exemption law for this type of investment. He traced the error back to the project specification and informed the analyst. In this case, the test activity correctly identified the defect and the source of the error.

Life Cycle

Evolution of a system, product, service, project, or other human-made entity from conception through retirement.

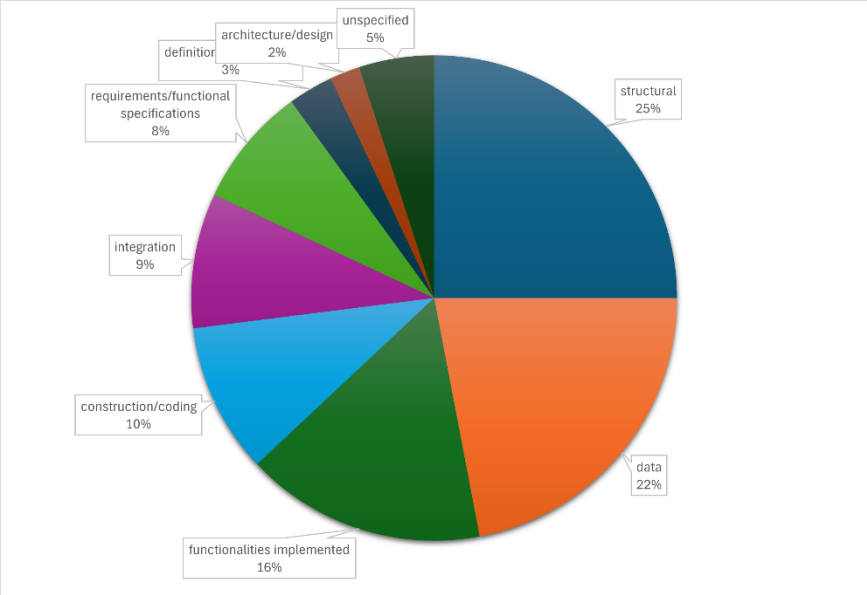
Development Life Cycle

- Software life cycle process that contains the activities of requirements analysis, design, coding, integration, testing, installation, and support for acceptance of software products.

Origin of errors

- Beizer (1990) [BEI 90] provides a study that has combined the result of several other studies to provide us with an indication of the origin of errors. The following is a summarized list of this study's results [BEI 90].

Error Origin	Percentage
structural	25%
data	22%
functionalities implemented	16%
construction/coding	10%
integration	9%
requirements/functional specifications	8%
definition/running tests	3%
architecture/design	2%
unspecified	5%



How many errors in software

- McConnell (2004) [MCC 04] suggested that this number varied based on the quality and maturity of the software engineering processes as well as the training and competency of the developers.
- The more mature the processes are, the fewer errors are introduced into the development life cycle of the software.
- Humphrey (2008) [HUM 08] also collected data from many developers. He found that a developer involuntarily creates about 100 defects for each 1000 lines of source code written.
- In addition, he noted large variations for a group of 800 experienced developers, that is, from less than 50 defects to more than 250 defects injected per 1000 lines of code.
- At Rolls-Royce, the manufacturer of airplane engines, the variation published is from 0.5 to 18 defects per 1000 lines of source code [NOL 15].
- The use of proven processes, competent and well-trained developers, and the reuse of already proven software components can considerably reduce the number of errors of a software

Causes of errors

- Errors are the main cause of poor software quality. It is important to look for the cause of error and identify ways in which to prevent these errors in the future.
- The causes are almost always human mistakes made by clients, analysts, designers, software engineers, testers, or users. SQA will need to develop a classification of the causes of software error by category which can be used by everyone involved in the software engineering process.

Causes of errors

Here are eight popular error-cause categories:

1. problems with defining requirements;
2. maintaining effective communication between client and developer;
3. deviations from specifications;
4. architecture and design errors;
5. coding errors (including test code);
6. non-compliance with current processes/procedures;
7. inadequate reviews and tests;
8. documentation errors.

Problems with Defining Requirements

- Defining software requirements is now considered a specialty, which means a business analyst or a software engineer specialized in requirements.
- There are a certain number of problems related to the clear, correct, and concise writing of requirements so that they can be converted into specifications that can be directly used by colleagues, such as architects, designers, programmers, and testers. It must also be understood that there are a certain number of activities that must be mastered when eliciting requirements:
  - – identifying the stakeholders (i.e., key players) who must participate in the requirements elicitation;
  - – managing meetings;
  - – interview techniques that can identify differences between wishes, expectations, and actual needs;
  - – clear and concise documentation of functional requirements, performance
  - requirements, obligations, and properties of future systems;
  - – applying systematic techniques for requirement elicitation;
  - – managing priorities and changes (e.g., changes to requirements).

Good quality requirement

- A requirement is said to be of good quality when it meets the following characteristics:
  - correct;
  - complete;
  - clear for each stakeholder group (e.g., the client, the system architect,testers, and those who will maintain the system);
  - unambiguous, that is, same interpretation of the requirement from all stake-holders;
  - concise (simple, precise);
  - consistent;
  - feasible (realistic, possible);
  - necessary (responds to a client's need);
  - independent of the design;
  - independent of the implementation technique;
  - verifiable and testable;
  - can be traced back to a business need;
  - unique.

Maintaining Effective Communications

- Errors can also occur in intermediary products due to involuntary misunderstandings between software personnel and clients and users from the outset of the software project. Software developers and software engineers must use simple, non-technical language and try to take into account the user's reality. They must be aware of all signs of lack of communication, on both sides. Examples of these situations are:
  - poor understanding of the client's instructions;
  - the client wants immediate results;
  - the client or the user does not take the time to read the documentation sent to him;
  - poor understanding of the changes requested from the developers during design;
  - the analyst stops accepting changes during the requirements definition and design phase, given that for certain projects 25% of specifications will have changed before the end of the project.

Maintaining Effective Communications

To minimize errors:

– take notes at each meeting and distribute the minutes to the entire project team;

– review the documents produced;

-be consistent with your use of terms and develop a glossary of terms to be shared with all stakeholders;

– inform clients of the cost of changing specifications;

– choose a development approach that allows you to accept changes along the way;

– number each requirement and implement a change management process

Deviations from Specifications

- This situation occurs when the developer incorrectly interprets a requirement and develops the software based on his own understanding. This situation creates errors that unfortunately may only be caught later in the development cycle or during the use of the software.

  Other types of deviations are:

  – reusing existing code without making adequate adjustments to meet new requirements;

  -deciding to drop part of the requirements due to budget or time pressures;

  – initiatives and improvements introduced by developers without verifying with clients.

Architecture and Design Errors

Errors can be inserted in the software when designers (system and data architects) translate user requirements into technical specifications. The typical design errors are:

- an incomplete overview of the software to be developed;
- unclear role for each software architecture component (responsibility, communication);
- unspecified primary data and data processing classes;
- a design that does not use the correct algorithms to meet requirements;
- incorrect business or technical process sequence;
- poor design of business or process rule criteria;
- a design that does not trace back to requirements;
- omission of transaction statuses that correctly represent the client's process;
- failure to process errors and illegal operations, which enables the software to process cases that would not exist in the client's sector of business—up to 80% of program code is estimated to process exceptions or errors.

Coding Errors

- inappropriate choice of programming language and conventions;
- not addressing how to manage complexity from the onset;
- poor understanding/interpretation of design documents;
- incoherent abstractions;
- loop and condition errors;
- data processing errors;
- processing sequence errors;
- lack of or poor validation of data upon input;
- poor design of business rule criteria;
- omission of transaction statuses that are required to truly represent the client process;
- failure to process errors and illegal operations, which enables the software to process cases that would not exist in the client's sector of business;
- poor assignment or processing of the data type;
- error in loop or interfering with the loop index;
- lack of skills in dealing with extremely complex nestings;
- integer division problem;
- poor initialization of a variable or pointer;
- source code that does not trace back to design;
- confusion regarding an alias for global data (global variable passed on to a subprogram).

Non-Compliance with Current Processes/Procedures

- When members of the software team need to coordinate their work, they will have difficulty understanding and testing poorly documented or undocumented software.
- The person who replaces or maintains the software will only have the source code as a reference.
- SQA will find a large number of non-conformities (with respect to the internal methodology) regarding this software.
- The test team will have problems developing test plans and scenarios, primarily because the specifications are not available.

Inadequate Reviews and Tests

All kinds of issues can crop up when reviewing and testing software:

- reviews only cover a very small part of the software's intermediate deliverables;
- reviews do not identify all errors found in the documentation and software code;
- the list of recommendations stemming from reviews is not implemented or followed up on adequately;
- incomplete test plans do not adequately cover the entire set of functions of the software, leaving parts untested;
- the project plan has not left much time to perform reviews or tests. In some cases, this step is shortened because it is wedged between coding and the final delivery. Delays in the early steps of the project do not always mean the delivery date will be extended, to the detriment of proper testing;
- the testing process does not correctly report the errors or defects found;
- The defects found are corrected, but are not subject to adequate regression testing (i.e., retesting the complete corrected software) thereafter.

Documentation Errors

- It has been recognized that obsolete or incomplete documentation for software being used in an organization is a common problem.
- Few development teams enjoy spending time preparing and reviewing documentation.

Figure 1.5 describes the reliability curve for computer hardware as a function of time. This curve is called a U-shaped or bathtub curve. It represents the reliability of a piece of equipment, such as a car, throughout its life cycle.

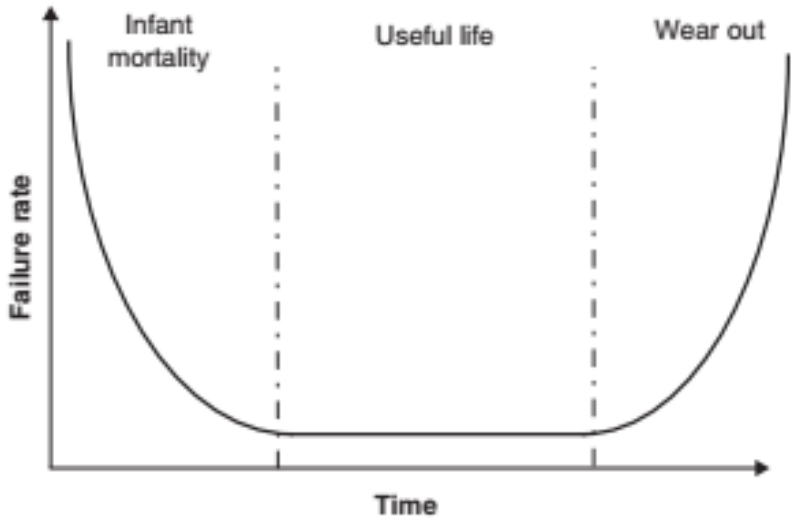


Figure 1.5

- Professor April worked in the Middle East between 1998 and 2003 in a large telecommunications company. When he arrived, he noticed that the original documentation for critical application software for the telephone company had not been updated in over 10 years.
- There was no software quality assurance function in the information technology division of that company.

Software Quality

- Conformance to established software requirements; the capability of a software product to satisfy stated and implied needs when used under specified conditions (ISO 25010 [ISO 11i])
- The degree to which a software product meets established requirements; however, quality depends upon the degree to which those established requirements accurately represent stakeholder needs, wants, and expectations [Institute of Electrical and Electronics Engineers (IEEE 730)] [IEE 14].

- These two points of view force the software engineer to establish the kind of agreement that must describe client's requirements and attempt to faithfully reflect his needs, wants, and expectations.
- Of course, there is a practical element to the functional characteristics that need to be described, but also implicit characteristics, which are expected of any professionally developed piece of software.

- The client's need for software (or more generally any kind of system) may be defined at four levels:
- True needs
- Expressed needs
- Specified needs
- Achieved needs

Quality Assurance

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) a set of activities designed to evaluate the process by which products are developed or manufactured;
3) the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality.

Perspective of QA

- This perspective of QA, in terms of software development, involves the following elements:
- – the need to plan the quality aspects of a product or service;
- – systematic activities that tell us, throughout the software life cycle, that certain corrections are required;
- – the quality system is a complete system that must, in the context of quality management, allow for the setting up of a quality policy and continuous improvement;
- – QA techniques that demonstrate the level of quality reached so as to instill confidence in users; and lastly,
- – demonstrate that the quality requirements defined for the project, for the change or by the software department have been met.

Business Model

- A business model describes the rationale of how an organization creates, delivers, and captures value (economic, social, or other forms of value).
- The essence of a business model is that it defines the manner by which the business enterprise delivers value to customers, entices customers to pay for value, and converts those payments to profit.

Knowledge of the business models and organizational culture will help the reader to [IBE 02]:

- – evaluate the effectiveness of new practices for an organization or specific project;
- – learn software practices from other fields or cultures;
- – understand the context that promotes collaboration with members of other cultures;
- – more easily integrate into a new job within another culture.

Business Model Descriptions

The five main business models in the software industry are [IBE 03]:

- – Custom systems written on contract: The organization makes profits by selling tailored software development services for clients (e.g., Accenture, TATA, and Infosys).
- – Custom software written in-house: The organization develops software to improve organizational efficiency (e.g., your current internal IT organization).
- – Commercial software: The company makes profits by developing and selling software to other organizations (e.g., Oracle and SAP).
- – Mass-market software: The company makes profits by developing and selling software to consumers (e.g., Microsoft and Adobe).
- – Commercial and mass-market firmware: The company makes profits by selling software in embedded hardware and systems (e.g., digital cameras, automobile braking system, and airplane engines).

Attributes or factors of business model

1. **Criticality:** The potential to cause harm to the user or prejudice the interests of the purchaser varies depending on the type of product. Some software may kill a person if it shuts down; other software programs may result in major money losses for many people; others will make a user waste time.
2. **Uncertainty of users' wants and needs:** The requirements for software that implements a familiar process in an organization are better known than the requirements for a consumer product that is so new that the end-users do not even know what they want.
3. **Range of environments:** Software written for use in a specific organization only has to be compatible with its own computer environment, whereas software sold to a mass market must work in a wide range of environments.
4. **Cost of fixing errors:** Distributing corrections for certain software applications (e.g., embedded software of an automobile) is usually far more costly than fixing a website.

Attributes or factors of business model

5. **Regulations:** Regulatory bodies and contractual clauses may require the use of software practices other than those that would normally be adopted. Certain situations require process audits to check whether a process was followed at the time of producing the software.
6. **Project size:** Projects that take several years and require hundreds of developers are common in certain organizations, whereas in other organizations, shorter projects developed by a single team are more typical.

Attributes or factors of business model

7. **Communication:** There are a certain number of factors, in addition to project scope, that can increase the quantity of person-to-person communication or make communications more difficult. Certain factors seem to occur more often within certain cultures, whereas others happen at random:
   - ❑Concurrent developer–developer communication: Communication with other people on the same project is affected by the way in which the work is distributed. In certain organizations, senior engineers design the software and junior staff carries out the coding and unit tests (instead of having the same person carrying out the design, coding, and unit tests for a given component). This practice increases the quantity of communications between developers.
   - ❑Developer–maintainer communication: Maintenance and enhancements require communication with the developers. Communication with developers is greatly facilitated when they work in the same area.
   - ❑Communication between managers and developers: Progress reports must be sent to upper management. However, the quantity of information and form of communication that managers believe they need may vary substantially.

Attributes or factors of business model

8. **Organization's culture:** The organization has a culture that defines how people work. There are four types of organizational cultures:
   - **Control culture**: control cultures, such as IBM and GE, are motivated by the need for power and security.
   - **Skill culture:** A culture of skill is defined by the need to make full use of one's skills: Microsoft is a good example.
   - **Collaborative culture:** A collaborative culture, as illustrated by Hewlett- Packard, is motivated by a need to belong.
   - **Thriving culture:** A thriving culture is motivated by self-actualization, and can be seen in start-up organizations.