## Implementation :

Implementation refers to the process of converting a new or revised system design into operational one. Conversion, implementation review and software maintenance are various aspects of system implementation.

The various types of implementation are:

1. Implementation of a computer system to replace a manual system: Here the task involved are converting files, training users, creating accurate file etc.
2. Implementation of a new computer system to replace an existing one: This implementation must be properly planned and is one of the most important conversion.
3. Implementation of a modified application to replace an existing one: This type of conversion is very easy to handle.

Purpose of System Implementation

1. To convert final physical system specifications into working and reliable software
2. To document work that has been done
3. To provide help for current and future users

Six major activities:

- Coding
- Testing
- Installation
- Documentation
- Training
- Support

Conversion:

Conversion refers to changing from one design to another system. The main objective of conversion is to put tested system into operation while holding costs, risks, and personal irritation to minimum. The various tasks involved in conversion are:

i. Creating computer compatible files.

ii. Training the operating staffs.

iii. Installing terminals and hardware.

# Maintrenance

System Maintenance means restoring something to its original conditions.. System maintenance conforms the system to its original requirements and enhancement adds to system capability by incorporating new requirements.

System Maintenance / Enhancement ● Thus, maintenance changes the existing system, enhancement adds features to the existing system, and development

replaces the existing system. It is an important part of system development that includes the activities which corrects errors in system design and implementation, updates the documents, and tests the data.

Maintenance Types System maintenance can be classified into three types ● Corrective Maintenance − Enables user to carry out the repairing and correcting leftover problems. ● Adaptive Maintenance − Enables user to replace the functions of the programs. ● Perfective Maintenance − Enables user to modify or enhance the programs according to the users' requirements and changing needs.

After a new system has been introduced, it enters the maintenance phase. The system is in production and is being used by the organization. While the system is no longer actively being developed, changes need to be made when bugs are found or new features are requested. During the maintenance phase, IT management must ensure that the system continues to stay aligned with business priorities and continues to run well. Documentation is key at this stage and others, so that there is information to refer to on the changes made. Feedback is important as well in order to drive future development.

## Activities in the Implementation Stage

- Describe the implementation stage for a project.
- Include a discussion of the six major activities for the implementation stage as described within the text: (1) coding, (2) testing, (3) installation, (4) documentation, (5) training and (6) support. The discussion of these six activities should describe specifically how each activity would be planned for the individual project situation.
- Discuss the benefits of using defined and repeatable processes for accomplishing these activities for the implementation stage.

Exploration – identifying the need for change, learning about possible evidence-based programs and practices (EBPs) that may provide solutions, learning about what it takes to implement the program or practice effectively, developing an Implementation Team to support the work as it progresses through the Stages, growing collaborators and champions, assessing and creating readiness for change, developing communication processes to support the work, and deciding to proceed (or not) Installation – securing and developing the support needed to put a new program or practice into place as intended, developing feedback loops between the practice and leadership level in order to streamline communication, and gathering feedback on how new practices are being implemented Initial Implementation – the first use of an EBP by practitioners and others who have just learned how to use the program or practice. Initial implementation is about trying out those new skills and practices, and getting better in implementation. In this stage, we are gathering data to check in on how implementation is going, and developing improvement strategies based on the data.

Implementation supports are refined based on data. For example, we might find that a new skill educators are using as part of social and emotional development could be further strengthened by additional coaching from an expert; so we would think about how to embed these strategies into ongoing coaching opportunities, and how we would gather data on if the coaching is leading to the improved use of these skills. Full Implementation – the skillful use of an EBP that is well-integrated into the repertoire of practitioners and routinely and effectively supported by successive program and local administrations

# Documenting the System

In one sense, every information systems development project is unique and will generate its own unique documentation. In another sense, though, system development projects are probably more alike than they are different.

Each project shares a similar systems development life cycle, which dictates that certain activities be undertaken and that each of those activities be documented. Specific documentation will vary depending on the life cycle you are following, and the format and content of the documentation may be mandated by the organization you work for. Start developing documentation elements early, as the information needed is captured.

***Types of documentation in system:***

We can simplify the situation by dividing the types of documentation into two basic types,

- system documentation and
- user documentation.

**SYSTEM DOCUMENTATION**

records detailed information about a system's design specifications, its internal workings, and its functionality. System documentation can be further divided into internal and external documentation.

**INTERNAL DOCUMENTATION**

is part of the program source code or is generated at compile time. External documentation includes the outcome of all of the structured diagramming techniques, such as data-flow and entity-relationship diagrams.

**USER DOCUMENTATION**

is written or other visual information about how an application system works and how to use it. Although not part of the code itself, external documentation can provide useful information to the primary users of system documentation—maintenance programmers.

For example, data-flow diagrams provide a good overview of a system's structure. In the past, external documentation was typically discarded after implementation, primarily because it was considered too costly to keep up to date but today's integrated development environment makes it possible to maintain and update external documentation as long as desired.

Whereas system documentation is intended primarily for maintenance programmers, user documentation is intended mainly for users. An organization should have definitive standards on system documentation.
These standards may include the outline for the project dictionary and specific pieces of documentation within it. Standards for user documentation are not as explicit

# User Documentation

User documentation consists of written or other visual information about an application system, how it works, and how to use it. The documentation lists the item necessary to perform the task the user inquired about. The user controls how much of the help is shown.

# Importance of documentation for the developer

Each function in a piece of software solves a specific problem. Before you try to solve any problem, you should have a good understanding of exactly what the problem is. It makes no sense just to start writing and then, afterwards, look at what you have come up with to see whether it solves any useful problem!

Inexperienced computer programmers imagine that they can keep all problem descriptions in their heads. Experience has shown that they can't. Three issues come up.

1. When writing a function definition without written documentation, you only have a rough idea of what it is supposed to do. While you write, the idea morphs in your head. A simple interruption can cause the idea to lose what focus it has. You start thinking about the program as a whole instead of thinking of just the

function that you are working on, and the function starts to take on responsibilities that it should have nothing to do with.

2. Suppose that you test the function and find that it does not work. So you need to fix it. But during the process of fixing it, you have nothing but your memory telling you want the function is supposedto do. It is difficult to keep that in your head along with the details of how the function is supposed to work, and the process of fixing a function definition takes the function further away from its original intent.

3. Later, when you need to use that function, you have forgotten just what it does. Unwilling to reverse-engineer it, you make a guess based on what you remember. You often forget important details, and your software does not work because of it. You are faced with laborious debugging to find out what is going on.

# Importance of documentation for the maintainer

You might have heard of "self-documenting code". The idea is for functions to be written in a readable form so that, to find out what a function does, you just read the function's definition. For very small pieces of software that can be achieved. But imagine a larger piece of software, say with about 1000 functions. Such software is built up function upon function; one function typically uses a few others that are defined in the same collection of 1000 functions, with the exception of the bottom-level functions that only use the library.

Suppose that the software has no internal documentation, and relies on "self-documenting code". Now you want to understand what a particular function does. But it uses 3 other undocumented functions, so you need to understand what they do first. Each of those uses 2 undocumented functions, so you must read their definitions too. It goes on and on. You find yourself reading thousands of lines of code to understand a single function whose body is only ten lines long.

The only way that anyone can work with undocumented software is to reverse-engineer the whole thing and add documentation that should have been written by the developer. Most of the time, that is too difficult. Undocumented software is often just thrown away as unmaintainable.