CHAPTER 2

MAINTENANCE SUPPORT PROCESSES

- Maintenance Planning
- Evolution and Maintenance Testing
- Configuration Management
- Problem Management
- Maintenance Supporting Tools

Prepared by Mintesinot A.

Maintenance Support Processes

- A process that is used, when required, by an operational process is said to be an **operational support process**.
- In most IS/IT organizations, developers, maintainers, and operations share the responsibility for these processes.

Maintenance Support Processes

- *Support processes typically include:*

- ➢A documentation process
- ➢A software configuration and version management process and tools
- ➢A product quality assurance process
- ➢The verification and validation processes
- ➢The problem resolution process, which is often shared with operations
- ➢The reviews and audits processes

Maintenance Planning

- Software maintenance, being a large portion of IT costs, needs to be part of the strategic IT plan.
- Maintenance planning ensures the creation and updating of plans that describe the current and planned software maintenance activities
- Software maintenance planning is typically elaborated from three perspectives: organizational, tactical, and operational.

Maintenance Planning

- At the organizational level, planning goals are strategic, general, and corporate, and include partners, the SLA, contracts, and licensing.
- At the tactical level, the planning goal is to identify and plan the activities of predelivery and transition for new software and also the yearly plans associated with a specific customer account.
- Finally, at the ***operational level***, the goals are specific:

i. To conduct impact analysis on a given maintenance request,

ii. To plan the recovery/failure tests,

iii. To plan software versions/releases (e.g., assign each request to a future version/release of the software), and

iv. To plan software upgrades (especially the COTS) for the many platforms.

Maintenance Planning

Goals of Maintenance Planning KPA(key process area)

i. Ensure a proactive collection and documentation of customer/user, development, and computer operation requests (in the short term, medium term …) to conduct detailed planning.

ii. Identify, communicate, and obtain consensus on priorities of requests currently in queue and assigned.

iii. Identify the required controls for each type of maintenance service and application software under maintenance.

iv. Plan the recovery/failure tests for all software in maintenance.

v. Identify version management plans (also called release plans) for all software maintained.

Maintenance Planning

vi. Prepare software upgrade plans.

vii. Establish a rationale for the allocation of new requests to maintenance engineers and for the capacity of maintenance.

viii. Inform the stakeholders of maintenance work performed, waiting in queue, and in progress based on agreed priorities.

ix. Inform stakeholders on the status of budgets and on the use of resources.

Maintenance Planning

Expected Results from the KPA

After the successful implementation of the exemplary practices of this KPA, the following can be observed:

- A sound forecast is available on needed resources in software maintenance (strategic plan, tactical plan, and operational plan).
- SLA, agreement, license, and contract renewals are planned.
- The failure/recovery tests are planned for each software in maintenance.
- Requests are allocated to future versions of the software.
- Software upgrades are planned.
- Pre-delivery and transition services are planned.
- There is an optimal allocation of maintenance resources to work items.
- Software maintenance has a plan, and this plan is communicated and approved.
- The stakeholders are informed about the plans.

Maintenance Planning

- Planning for maintenance may include:
  - Determining the maintenance effort
  - Determining the current maintenance process
  - Quantifying the maintenance effort
  - Projecting maintenance requirements
  - Developing a maintenance plan.
- IEEE P1058/D2.1 and IEEE Std 1058a-1998 may also be used for guidance in maintenance planning.

Determine Maintenance Effort

- The first step in the maintenance planning process is an analysis of current service levels and capabilities.
- This includes an analysis of the existing maintenance portfolio and the state of each system within that portfolio.
- At the system level, each system should be examined to determine the following:
  - Age since being placed in production
  - Number and type of changes during life
  - Usefulness of the system
  - Types and number of requests received for changes
  - Quality and timeliness of documentation
  - Any existing performance statistics (CPU, disk I/O, etc.).

Determine Maintenance Effort

- The reviews of the maintenance staff and the maintenance procedures are also necessary to determine the overall maintenance effort.
- The analysis at this stage is simply to gather those measures needed to determine the following:
    - The number of maintainers, their job descriptions, and their actual jobs
    - The experience level of the maintenance staff, both industry-wide and for the particular application
    - The rate of turnover and possible reasons for leaving
    - Current written maintenance methods at the systems and program level
    - Actual methods used by programming staff
    - Tools used to support the maintenance process and how they are used.

Determine Current Maintenance Process

- The maintenance process is a natural outgrowth of many of the baseline measures.
- Once those measures have been collected, the actual process needs to be determined.
- In some organizations, the process is tailored to the type of maintenance being performed and can be divided in several different ways.
- This can include different processes for *corrections vs. enhancements*, *small changes vs. large changes*, etc.
- It is helpful to classify the maintenance approaches used before defining the processes.
- Each process will then be described by a series of events.
- In general, the flow of work is described from *receipt of a request to its implementation and delivery*.

Quantify Maintenance Effort

- ➢Each step in the process needs to be described numerically in terms of volume or time.
- ➢These numbers can then be used as a basis to determine the actual performance of the maintenance organization.

Projecting Maintenance Requirements

- At this stage, the maintenance process needs to be coupled to the business environment.
- A review of future expectations should be completed and may include the following:
  1. Expected external or regulatory changes to the system
  2. Expected internal changes to support new requirements
  3. Wish-list of new functions and features
  4. Expected upgrades for performance, adaptability, connectivity, etc.
  5. New lines of business that need to be supported
  6. New technologies that need to be incorporated.

  ➢ These need to be quantified (or sized) to determine the future maintenance load for the organization.

Develop Maintenance Plan

- The information collected will provide a basis for a new maintenance plan.
- The plan should cover the following four main areas:
  - Maintenance Process
  - Organization
  - Resource allocations
  - Performance tracking.
- Each of these issues are addressed and embedded in the final maintenance plan.
- The actual ***process*** should be described in terms of
  - its ***scope***,
  - the ***sequence*** of the process
  - the ***control*** of the process.

Develop Maintenance Plan

Process scope

- The plan needs to define the boundaries of the maintenance process.
- The process begins at some point (receipt of the request) and will end with some action (delivery and sign-off).
- In addition, the difference between maintenance and development should be addressed at this point.
- Is an enhancement considered to be a new development, or maintenance?
- At what point does a newly developed system enter the maintenance process?
- Another issue that should be defined within the scope is whether and how the maintenance process will be categorized.
- Will there be differences between reporting and other types of maintenance?
- Will adaptations and enhancements be considered within the same process or will they be handled differently?

Develop Maintenance Plan

Process sequence

- The overall flow of work needs to be described.
- This should include the following:

i. Entry into automated SCM and project management systems;

ii. Descriptions of each process step and their interfaces;

iii. The data flow between processes.

Process control

- Each step in the process should be controlled and measured.
- Expected levels of performance should be defined.
- The control mechanisms should be automated, if possible.
- The control process should follow the standards set forth in this document.

Develop Maintenance Plan

Organization

- Staff size can be estimated from the current work load and estimates of future needs.
- This estimate may also be based on the expected productivity of each step in the process.

Resource allocation

- An analysis of the hardware & software most appropriate to support the organization’s needs.
- The development, maintenance, and target platforms should be defined and differences between the environments should be described.
- Tool sets that enhance productivity should be identified and provided.
- The tools should be accessible to all who need them, with sufficient training.

Develop Maintenance Plan

Tracking

- Once the process is in place, it should be tracked and evaluated to judge its effectiveness.
- If each step in the process has measurement criteria, it should be a straightforward process to collect measurements and evaluate performance over time.

Implementation of plan

- Implementing a maintenance plan is accomplished in the same way that any organizational change is performed.
- It is important to have as much technical, professional, and managerial input as possible when the plan is being developed.

Evolution And Maintenance Testing

Testing: is the examination of a software system in the context of a given specification set.

Types of code Tests

i. Black Box and White Box Testing

- Black Box: we don't see inside it, we just see what goes in and what comes out.
- White Box: we 'see inside the box' and look at the detail of the code.

ii. Structured Testing

- An aim of structured testing is to maximize the number of errors found by the test cases used and to avoid redundant test cases.

iii. Integration Testing

- Testing a program starting with tests of its elements and then combining them to test larger elements is known as integration testing.

iv. Regression Testing

Evolution And Maintenance Testing

Maintenance Testing

- Maintenance Testing is done on the already deployed software.
- The deployed software needs to be enhanced, changed or migrated to other.
- The Testing done during this enhancement, change and migration cycle is known as ***maintenance testing***.

- Usually maintenance testing is consisting of two parts:

Types Of Maintenance Testing

- While performing maintenance testing, testers should consider two types of maintenance testing.

i. Confirmation Testing: Testing the modified functionality

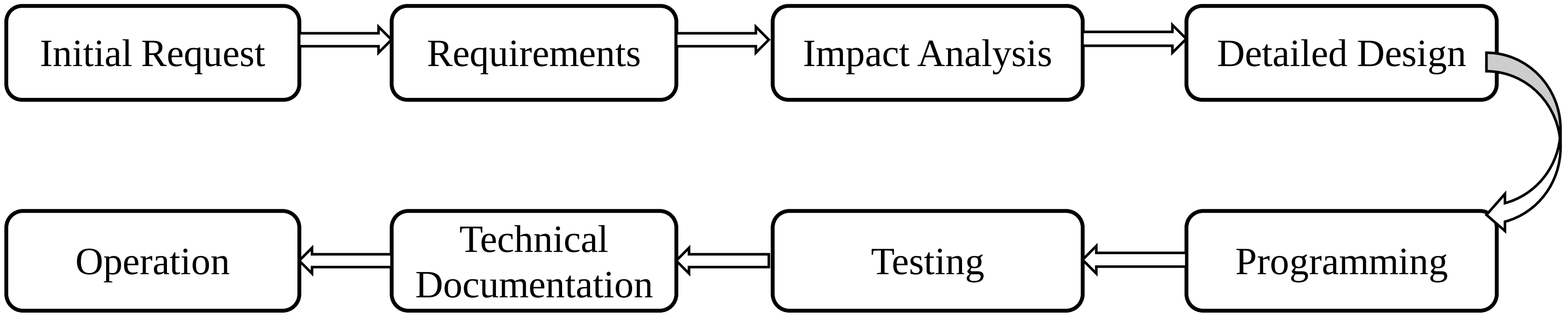
ii. Regression Testing: Testing the existing functionality

Configuration Management(cm)

- ✓ Software systems are constantly changing during development and use.
- ✓ **Software Configuration Management** encompasses the disciplines and techniques of *initiating*, *evaluating* and *controlling* ***change to software products*** during and after the software engineering process.
- ✓ Configuration management consists of determining the configuration and description of products and intermediate work products at a precise time.
- ✓ The means by which the process of software development and evolution is controlled is called **configuration management.**
- ✓ Its objective is to ensure systematic control over configuration changes and to maintain integrity and traceability throughout the life cycle of a request that will modify the existing software.

Configuration Management(cm)

- Configuration management's first objective is to establish a link between the different intermediate products throughout a change.
- The maintainer is thus able to follow a change's progress from



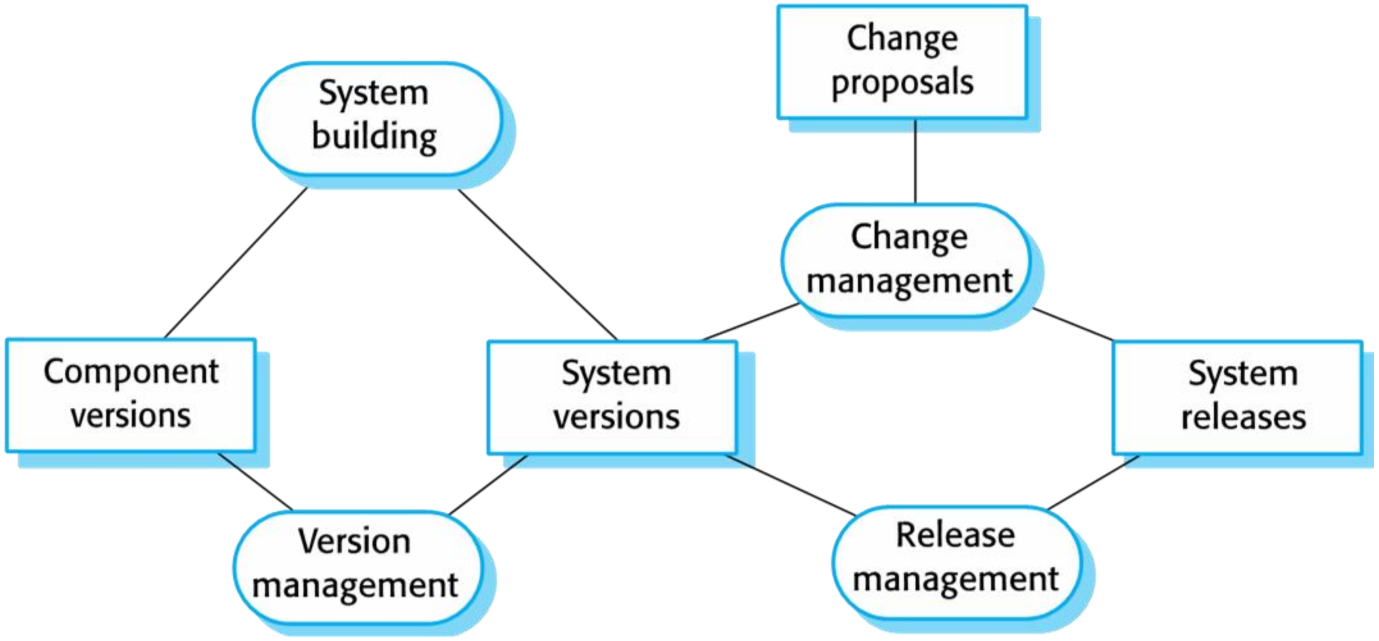
Configuration Management

- ✓ This first aspect of configuration is called traceability.
- ✓ Another aspect of configuration management is the use of support software to reserve components while the software is being worked on.
- ✓ CM ensures that employees cannot change a component without being aware of changes made by a colleague.
- ✓ CM is essential for team projects to control changes made by different developers
- ✓ Developers and maintainers make images of software at every critical step using the same tools.
- ✓ For example
  - ✓ They would make a copy of the complete configuration at each major step in the testing phase.
  - ✓ This technique allows them to isolate an image of all software components at any given time.
  - ✓ They can always revert to this state should the work being done become corrupted.
  - ✓ It can be imagined that backup copies are, in fact, images of the software's configuration.

Configuration Management Activities

- *Version management*
  - Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.
- *System building*
  - The process of assembling program components, data and libraries, then compiling these to create an executable system.
- *Change management*
  - Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.
- *Release management*
  - Preparing software for external release and keeping track of the system versions that have been released for customer use.

Configuration Management Activities



Version Management

- Version management (VM) is the process of keeping track of different versions of software components or configuration items and the systems in which these components are used.
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Therefore version management can be thought of as the process of managing ***codelines*** and ***<u>baselines</u>***.

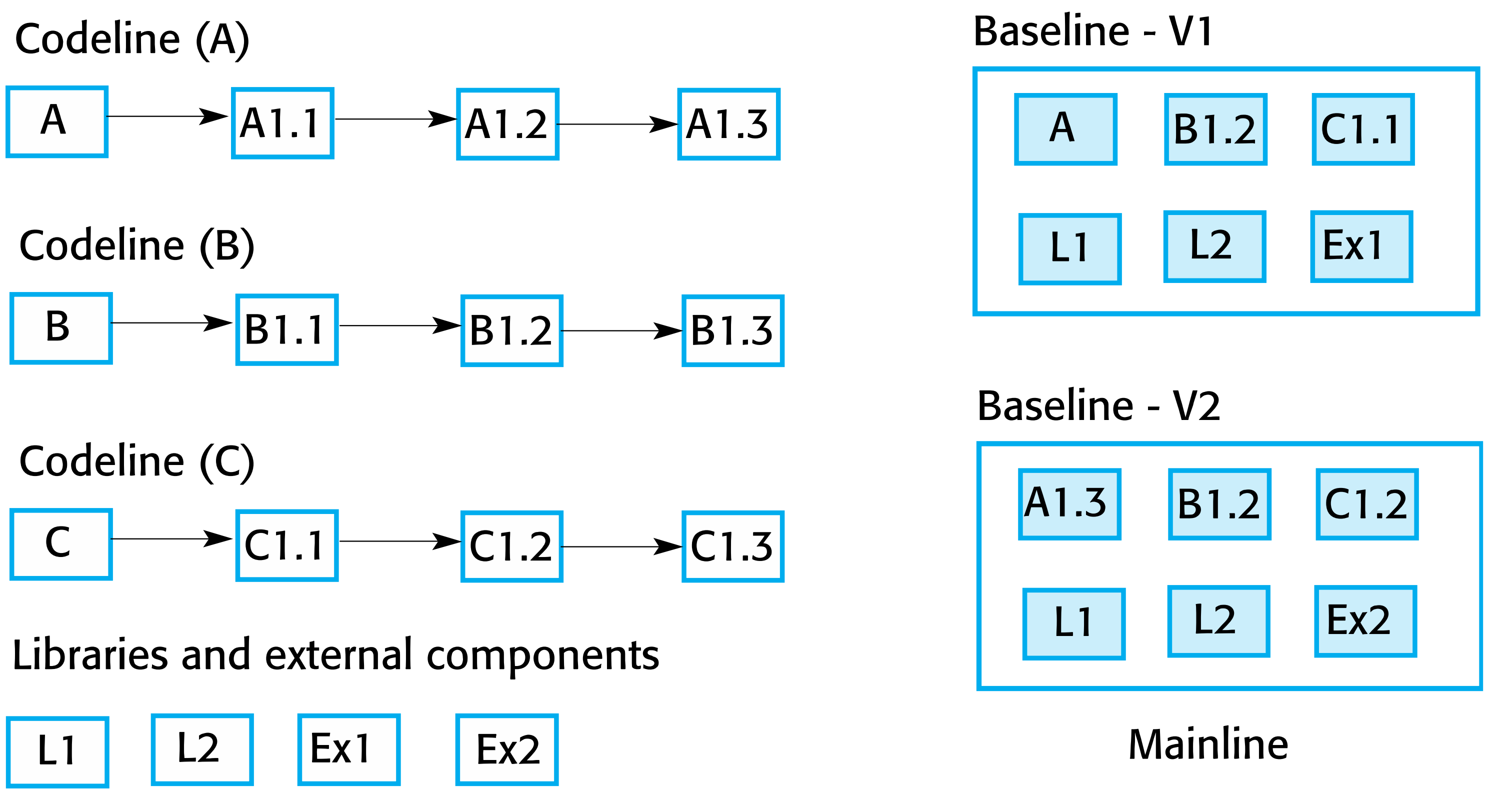
Codelines And Baselines

- A codeline is a sequence of versions of source code with later versions in the sequence derived from earlier versions.
- Codelines normally apply to components of systems so that there are different versions of each component.
- A baseline is a definition of a specific system.
- The baseline therefore specifies the component versions that are included in the system plus a specification of the libraries used, configuration files, etc.

Baselines

- Baselines may be specified using a configuration language, which allows you to define what components are included in a version of a particular system.
- Baselines are important because you often have to recreate a specific version of a complete system.
  - For example, a product line may be instantiated so that there are individual system versions for different customers. You may have to recreate the version delivered to a specific customer if, for example, that customer reports bugs in their system that have to be repaired.

Codelines and Baselines



System Building

- System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.
- System building tools and version management tools must communicate as the build process involves checking out component versions from the repository managed by the version management system.
- The configuration description used to identify a baseline is also used by the system building tool.

Change Management

- **Change management** is the process of evaluating the impact of a requirement or design change on the system, analyzing the effects of a proposed change in terms of the system foundation architecture, performance, costs, and schedule criteria.
- The general change management process:
  - The change is requested
  - The change request is assessed against requirements and project constraints
  - Following the assessment, the change request is accepted or rejected
  - If it is accepted, the change is assigned to a developer & implemented
  - The implemented change is audited.

Configuration Item

Configuration Item: An aggregation of hardware, software, or both, designated for configuration management and treated as a single entity in the configuration management process.

- Software configuration items are not only source files but all types of documents.
- In some projects, not only software but also hardware configuration items (CPUs, bus speed frequencies) need to be put under control!

Configuration Item

- Not every entity needs to be under configuration management control all the time
- Two Issues:
  - **What:** Selection of Configuration Items
    - What should be under configuration control?
  - **When:** When do you start to place entities under configuration control?
- Choices for the Project Manager:
  - Starting with Configuration Items too early introduces bureaucracy
  - Starting with Configuration Items too late introduces chaos.

Configuration Item

- Selecting the right configuration items is a skill that takes practice
  - Very similar to object modeling
  - Use techniques similar to object modeling for finding configuration items!
    - Find the configuration items
    - Find relationships between configuration items.

Possible Selection Of Configuration Items

- Problem Statement
- Software Project Management Plan (SPMP)
- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Project Agreement
- Object Design Document (ODD)
- Dynamic Model
- Object model
- Functional Model
- Unit tests
- Integration test strategy
- Source code
- API Specification
- Input data and data bases
- Test plan
- Test data
- Support software (part of the product)
- Support software (not part of the product)
- User manual
- Administrator manual

Release Management

- System releases include executable code, data files, configuration files and documentation.
- *Release management* involves making decisions on system release dates, preparing all information for distribution and documenting each system release.

Release Management

- A software release can be categorized as *safe*, *risky*, or *unsafe* according to the condition described as follows.
- Let *m* be the mean of the incremental growth *mi* of the system in going from release *i* to release *i* + 1 and *s* be the standard deviation of the incremental growth.
  - The release is ***safe*** if the content of the *i*th desired release (say, *mi*) is less than or equal to *m*.
  - The release is said to be ***risky*** if the content of the desired release is greater than *m* but less than *m* + 2*s*.
  - Finally, the release is ***unsafe*** if the content of the desired release is close to or greater than *m* + 2*s*.

Release Management

- Based on the aforementioned concepts of safety, concrete activities for release management are as follows:
  - Ensure that the release is *safe*.
  - When the release is not *safe*, then distribute the growth across several releases to make individual releases safe.
  - If excessive functional increments are unavoidable, plan for follow-on clean-up releases with a focus on fixing defects and updating documentation.
  - Follow established software engineering principles, namely, information hiding to minimize spread of changes between system elements.
  - By allocating resources, put emphasis on antiregressive work, namely, restructuring, eliminating dead code, and reengineering.
  - Consider the alternation of enhancement and extension with clean-up and restructuring releases.

Problem Management

- A mechanized problem reporting process is used that ensures efficient communications for quick resolution of failures and other support requests.
- A specific user request, sometimes called a "ticket," will typically circulate between the help desk, software maintenance, and operations in order to isolate a problem.
- The IT department provides a large number of IT services to its customers, which are mainly departments from the sibling organization.
- To manage the communication with customers regarding those services, the department has implemented *helpdesk management* and *problem management processes*.

Problem Management

- The implementation of these processes has been based on the Information Technology Infrastructure Library (ITIL).
- The ITIL process Helpdesk Management is used to guarantee the continuity of services, while the ITIL process Problem Management is used to improve the level of service in the future.
- **Helpdesk Management** deals with *incidents*, whereas **Problem Management** is concerned with solving the *problems* that cause these incidents.

Problem Management Process

The problem management process has many steps, and each is vitally important to the success of the process and the quality of service delivered.

1. **Detect the problem:** A problem is raised either through
   - ✓ Appreciation from the service desk
   - ✓ Proactive evaluation of incident patterns and alerts from event management
   - ✓ Continual service improvement processes.

2. **Log the problem:** Problems are logged in a problem record. A problem record is a compilation of every problem in an organization. Pertinent problem data, such as the time and date of occurrence, the related incident, the symptoms, previous troubleshooting steps…

3. **Categorize the problem:** Incident [and problem] categorization involves assigning a main and secondary category to the issue. This step is beneficial in several ways.
   - ✓ It allows the service desk to sort and model incidents that occur regularly.
   - ✓ The modeling allows for automatic assignment of prioritization.
   - ✓ The third and most important benefit is the ability to gather and report on service desk data.

Problem Management Process

4. **Prioritize the problem:** is determined by its impact on users and on the business and its urgency. Urgency is how quickly the organization requires a resolution to the problem.

5. **Investigating and diagnosing the problem:** involves analyzing the incidents that lead to the problem report as well as further testing that may not be possible at the service desk level, such as advanced log analysis.

6. **Identify a workaround for the problem:** A workaround enables the service desk to restore services to users while the problem is being resolved. A problem can take anywhere from an hour to months to resolve, therefore a workaround is vital.

Problem Management Process

7. **Raise a known error record:** It’s good practice to record a known error in both an incident knowledge base and what ITIL calls a known error database (KEDB). Documenting the workaround allows the service desk to resolve incidents quickly and avoid further problems being raised on the same issue.

8. **Resolve the problem:** Resolution resolves the underlying cause of a set of incidents and prevents those incidents from recurring. Some resolutions may require the change management board, as they may affect service levels.

9. **Close the problem:** This step should only occur after the problem has been raised, categorized, prioritized, identified, diagnosed, and resolved.

10. **Review the problem:** During the review, the problem management team evaluates the problem documentation and identifies what happened and why.
    - ✓ Lessons learned, such as process bottlenecks, what went wrong, and what helped should be discussed.

Maintenance Supporting Tools

- **Tool** - implement or device used to carry out functions automatically or manually.
- **Software maintenance tool** - an artefact used to carry out automatically a function relevant to software change.
- Software maintenance tools are programs used by software engineers
  - ✓ To increase their productivity for gathering data
  - ✓ To detecting bugs and
  - ✓ To managing their software.
- Without these tools, it would be nearly impossible for these engineers to go through thousands of lines of code to find errors or determine why a particular server went down.

Maintenance Supporting Tools

Criteria for Selecting Tools

- There are several vendors developing and marketing a wide variety of tools that claim to support software maintenance.
- To acquiring a tool for software maintenance work, there are a number of factors that should be taken into consideration:

- ✓ ***Capability:*** The tool must be capable of supporting the task to be performed.
- ✓ ***Features:*** The features expected of any potential tool need to be considered. the importance of each of these features should be rated and the tool selected accordingly.
- ✓ ***Cost and benefits:*** The cost of introducing a tool needs to be weighed against the benefits. The benefits that the tool brings need to be evaluated in terms of indicators such as product quality, productivity, responsiveness, cost reduction.
- ✓ ***Platform:*** The platform refers to the specific hardware and software environments on which the tool runs.

Maintenance Supporting Tools

Criteria for Selecting Tools…

- ✓ ***Programming language:*** To be on the safe side, it is important to obtain a tool that supports a language that is already an industry standard.
- ✓ ***Ease of use:*** Usually, a tool that has a similar 'feel' to the tools that users are already familiar with tends to be accepted more easily than one which is radically different.
- ✓ ***Openness of architecture:*** The ability to integrate a tool with others from different vendors plays a major role in its extensibility and flexibility.
- ✓ ***Stability of vendor:*** It is important to consider the reputation of the vendor before acquiring a tool. it is essential to look into the background of any company being considered as a supplier of a tool. If the tool is one with an open architecture then this factor may not be so important.
- ✓ ***Organizational culture:*** to increase the chances of the tool being accepted by the target users, it is essential to take such culture and work patterns into consideration

Maintenance Supporting Tools

- In this section an attempt is made to classify maintenance tools based on the specific tasks that they support. In cases where a tool supports more than one task, this will be pointed out in the discussion.
- The categories of tasks for which tools will be discussed are:
    - I. Program understanding and reverse engineering
    - II. Testing
    - III. Configuration management
    - IV. Documentation and measurement.

Maintenance Supporting Tools

I. Tools for Comprehension and Reverse Engineering

- Program understanding involves having a general knowledge of **what** a program does and how it relates to its environment; identifying **where** in the system changes are to be effected; and knowing **how** the different components to be modified work.
- Reverse engineering goes a step further by enabling analysis and different representations of the system to promote that understanding.

a) Program Slicer:

- ✓ major problems with software maintenance is coping with the size of the program source code.
- ✓ It is important that a programmer can select and view only those parts of the program that are affected by a proposed change without being distracted by the irrelevant parts.
- ✓ The program slicer also displays data links and related characteristics to enable the programmer to track the effect of changes.

Maintenance Supporting Tools

Tools for Comprehension and Reverse Engineering…

b) Static Analyser

- A **static analyser** allows derivation different aspects of the program (such as modules, procedures, variables, data elements, objects and classes, and class hierarchy) through careful and deep examination of the program text.

c) Dynamic Analyser

- There is a need to control and analyze various aspects of the program when it is executing.
- The dynamic analyzer allows a maintainer to trace the execution path of the system while it is running - it acts as a tracer.

Maintenance Supporting Tools

Tools for Comprehension and Reverse Engineering…

d) ***Data Flow Analyser***: is a static analysis tool that allows the maintainer to track all possible data flow and control flow paths in the program and also to backtrack.

- This is particularly important when there is a need for impact analysis: studying the effect of a change on other parts of the system.

e) **Cross-Referencer:** is a tool that generates an index of the usage of a given program entity.

- For example, it can produce information on the declarations of a variable and all the sections in the program in which it has been set and used.

Maintenance Supporting Tools

Tools for Comprehension and Reverse Engineering…

f) **Dependency Analyzer:** helps the maintainer to analyze and understand the interrelationships between entities in a program.

- This tool is particularly useful in situations where logically related entities, such as variables, may be physically far apart in the program.

g) **Transformation Tool:** converts programs between different forms of representations, usually between text and graphics; for example, transforming code to visual form and vice versa.

Maintenance Supporting Tools

II. Tools to Support Testing

- Testing is one of the most expensive and demanding tasks in software development and maintenance and can benefit greatly from automated support.

a) **Simulator:** Using a test **simulator,** a controlled environment is set up for the testing to take place.

- The set-up and components of the environment will depend on the type of application being tested.
- For instance, a simulator for a real-time system needs to provide a priority-based, event-driven, multi-tasking environment together with interprocess communication through message queues and shared memory

Advantage: the maintainer can try out the effect of a change before implementing the change on the actual operational system.

Disadvantage: the results and observations may be misleading since some of the constraints in the real environment may not be reflected in the controlled environment.

Maintenance Supporting Tools

Tools to Support Testing…

b) **Test Case Generator:** Sets of test data used to test the functionality of the system undergoing modification are produced.

- The test data can be obtained from the system as well as data files.
- The tool that assists in generating test data is called a **test data generator**.
- The tool usually requires definition of the criteria for generating the test cases.
- ***Example***

c) **Test Paths Generator**

- It is important to know all the potential data flow and control flow paths that may have been affected by a change. This information enables the maintainer to carry out the appropriate set of tests to ensure that a change has achieved the desired effect.
- **Example**

Maintenance Supporting Tools

III. Tools to Support Configuration Management

- Effective configuration management is not possible without the aid of some kind of automated support tools in most maintenance environments.

a) Source Code Control System

- Source Code Control System (SCCS) consists of various utility programs usually accessed by a front end such as the *sees* UNIX command.
- An associated SCCS history file may be created for each file and the SCCS programs will be applied to the relevant history files such that versions can be tracked and programmers can keep track of which files have been changed and when.
- ***Examples***
    - CFEngine Configuration Tool
    - Desktop Central

Maintenance Supporting Tools

b) Other Utilities

- Many of the familiar programs and utilities have their part to play.
- For example, *Is* and *dir* commands, and file manager tools give information on files and directory structure.
- *Find* tools and the *grep* command allow searches for specific patterns within files.
- The important thing is to be aware that such tools exist and to take advantage of whichever are the most appropriate in a particular situation.

Maintenance Supporting Tools

IV. Other Tasks

a) Documentation

- The importance of documentation for software maintenance cannot be overemphasised.
- Its importance is reflected in the observation that lack of documentation is considered to be one of the major problems that software maintainers face.
- There is a wide variety of documentation tools which include hypertext based tools, data flow and control chart generators, requirements tracers, and CASE tools.

b) Complexity Assessment

- Usually in maintenance projects, it is essential to assess the complexity of a system before effecting any change to it.
- A **complexity quantifier** is a tool used to measure the complexity of a program. Such complexity measures are usually based on factors such as the underlying algorithm of a program or its structure.

Maintenance Supporting Tools

- The CASE tools aid software maintenance activities.
- The vision for CASE is an interrelated set of tools supporting all aspects of software development and maintenance [ISO/IEC DTR 14471].
- This interrelated collection of CASE tools should be brought together in the form of a ***Software Engineering Environment*** (SEE) to support the methods, policies, guidelines, and standards that support software maintenance activities.
- A ***Software Test Environment*** (STE) should also be provided for the maintainer so that the modified software product can be tested in a non-operational environment.
- The SEE provides the tools to initially develop and modify the software products. The STE provides the test environment. The STE should be used to test the modified software products in a non-operational environment.

Reference

- Alain April, Alain Abran (2008), Software Maintenance Management Evaluation and Continuous Improvement
- Priyadarshi Tripathy, Kshirasagar, 2015, Naik, Software evolution and maintenance : a practitioner's approach.
- Penny Grub, Armstrong A Takang, Software Maintenance Concepts and Practice, 2nd edition
- IEEE Std 1219-1998, IEEE Standard for Software Maintenance
- FRANK NIESSINK and HANS VAN VLIET(2000), Software Maintenance from a Service Perspective, Faculty of Sciences, division of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan Amsterdam, the Netherlands.
- Pierre Bourque, École de technologie supérieure(2014), Guide to the Software Engineering Body of Knowledge (SWEBOK) Version 3.0, A Project of the IEEE Computer Society.
- www.techwalla.com/articles/list-of-software-maintenance-tools(Retrieved March 20 2018)
- www.softwaretestingmentor.com/maintenance-testing/(Retrived March 20 2018)
- http://www.bmc.com/guides/itil-problem-management.html(Updated)