Software Maintenance*

Trung Hung VO

Abstract

We introduce the concepts and terminology that form an underlying basis to understanding the role and scope of software maintenance and activities required to provide cost-effective support to software. Activities are performed during the pre-delivery stage, as well as during the post-delivery stage.

1 Introduction

In software engineering, software maintenance is the process of enhancing and optimizing deployed software (software release), as well as remedying defects. Software maintenance is one of the phases in the software development process, and follows deployment of the software into the field. The software maintenance phase involves changes to the software in order to correct defects and deficiencies found during field usage as well as the addition of new functionality to improve the software's usability and applicability.

Software maintenance involves a number of specific techniques. One technique is static slicing, which is used to identify all the program code that can modify some variable. It is generally useful in refactoring program code and was specifically useful in assuring Y2K compliance.

The software maintenance phase is an explicit part of the waterfall model of the software development process which was developed during the structured programming movement of computer programming. The other major model, the spiral model developed during the object oriented movement of software engineering makes no explicit mention of a maintenance phase. Nevertheless, this activity is notable, considering the fact that two-thirds of a software system's lifetime cost involves maintenance.

*Version 1.1: Jul 8, 2007 4:05 am GMT-5
†http://creativecommons.org/licenses/by/2.0/

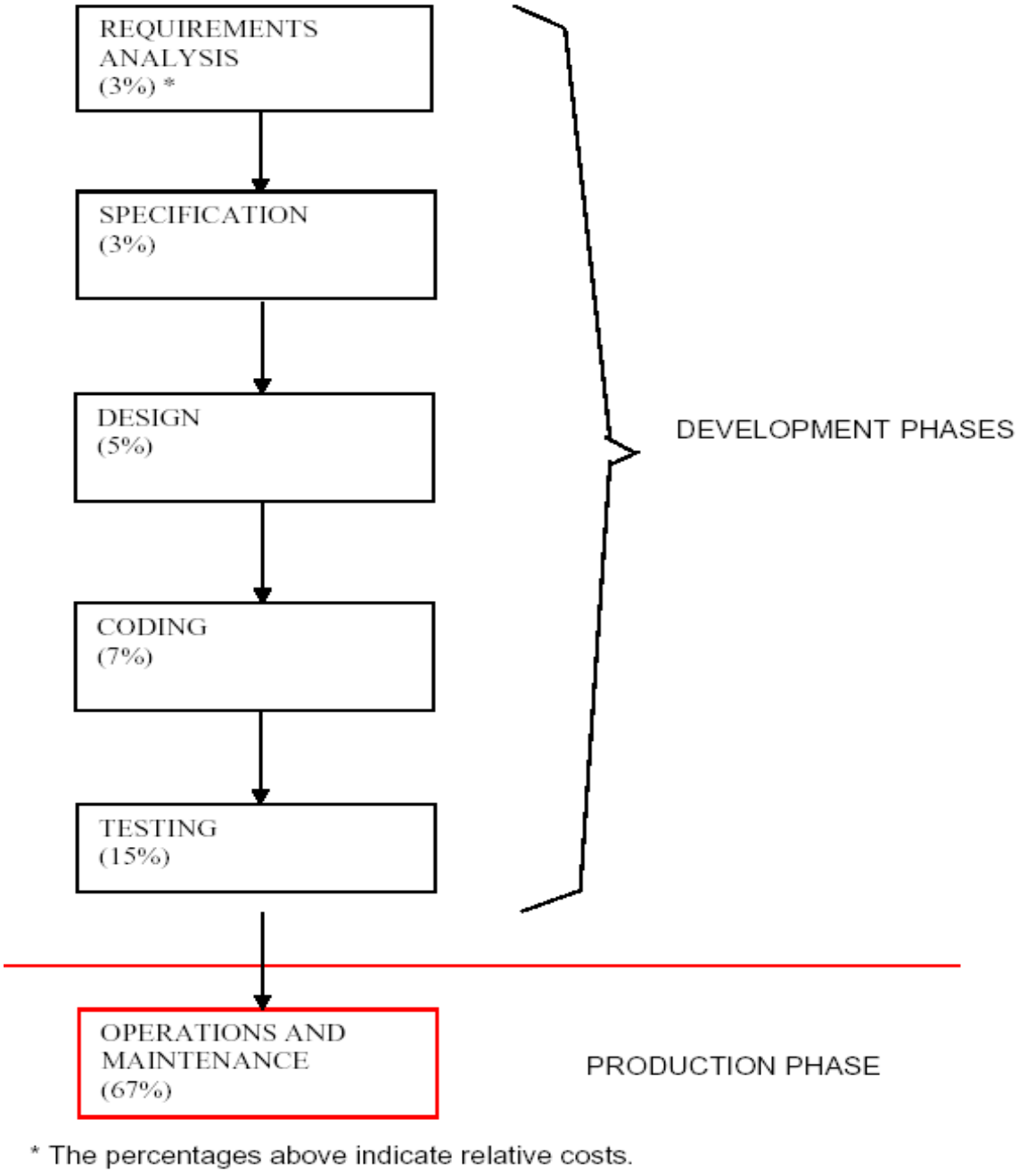


Figure 1

In a formal software development environment, the developing organization or team will have some mechanisms to document and track defects and deficiencies. Software just like most other products, is typically released with a known set of defects and deficiencies. The software is released with the issues

because the development organization decides the utility and value of the software at a particular level of quality outweighs the impact of the known defects and deficiencies.

The known issues are normally documented in a letter of operational considerations or release notes so that the users of the software will be able to work around the known issues and will know when the use of the software would be inappropriate for particular tasks.

With the release of the software, other, undocumented defects and deficiencies will be discovered by the users of the software. As these issues are reported into the development organization, they will be entered into the defect tracking system.

The people involved in the software maintenance phase are expected to work on these known issues, address them, and prepare for a new release of the software, known as a maintenance release, which will address the documented issues.

2 Software Maintenance Fundamentals

This section introduces the concepts and terminology that form an underlying basis to understanding the role and scope of software maintenance. The topics provide definitions and emphasize why there is a need for maintenance. Categories of software maintenance are critical to understanding its underlying meaning.

2.1 Definitions and Terminology

Software maintenance is defined as the 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. The standard also addresses maintenance activities prior to delivery of the software product, but only in an information appendix of the standard.

The IEEE/EIA 12207 standard for software life cycle processes essentially depicts maintenance as one of the primary life cycle processes, and describes maintenance as the process of a software product undergoing "modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity".

2.2 Nature of Maintenance

Software maintenance sustains the software product throughout its operational life cycle. Modification requests are logged and tracked, the impact of proposed changes is determined, code and other software artifacts are modified, testing is conducted, and a new version of the software product is released. Also, training and daily support are provided to users. Pfleeger states that "maintenance has a broader scope, with more to track and control" than development.

A maintainer is defined by IEEE/EIA 12207 as an organization which performs maintenance activities, sometimes refer to individuals who perform those activities, contrasting them with the developers.

IEEE/EIA 12207 identifies the primary activities of software maintenance as: process implementation; problem and modification analysis; modification implementation; maintenance review/acceptance; migration; and retirement.

Maintainers can learn from the developer's knowledge of the software. Contact with the developers and early involvement by the maintainer helps reduce the maintenance effort. In some instances, the software engineer cannot be reached or has moved on to other tasks, which creates an additional challenge for the maintainers. Maintenance must take the products of the development, code, or documentation, for example, and support them immediately and evolve/maintain them progressively over the software life cycle.

2.3 Need for Maintenance

Maintenance is needed to ensure that the software continues to satisfy user requirements. Maintenance is applicable to software developed using any software life cycle model (for example, spiral). The system changes due to corrective and non-corrective software actions. Maintenance must be performed in order to:

- Correct faults
- Improve the design
- Implement enhancements
- Interface with other systems
- Adapt programs so that different hardware, software, system features, and telecommunications facilities can be used
- Migrate legacy software
- Retire software

The maintainer's activities comprise four key characteristics, according to Pfleeger:

- Maintaining control over the software's day-to-day functions
- Maintaining control over software modification
- Perfecting existing functions
- Preventing software performance from degrading to unacceptable levels

2.4 Majority of Maintenance Costs

Maintenance consumes a major share of software life cycle financial resources. A common perception of software maintenance is that it merely fixes faults. However, studies and surveys over the years have indicated that the majority, over 80%, of the software maintenance effort is used for non-corrective actions. Jones describes the way in which software maintenance managers often group enhancements and corrections together in their management reports. This inclusion of enhancement requests with problem reports contributes to some of the misconceptions regarding the high cost of corrections. Understanding the categories of software maintenance helps to understand the structure of software maintenance costs. Also, understanding the factors that influence the maintainability of a system can help to contain costs. Pfleeger presents some of the technical and non-technical factors affecting software maintenance costs, as follows:

- Application type
- Software novelty
- Software maintenance staff availability
- Software life span
- Hardware characteristics
- Quality of software design, construction, documentation and testing

2.5 Evolution of Software

Lehman first addressed software maintenance and evolution of systems in 1969. Over a period of twenty years, his research led to the formulation of eight "Laws of Evolution". Key findings include the fact that maintenance is evolutionary developments, and that maintenance decisions are aided by understanding what happens to systems (and software) over time. Others state that maintenance is continued development, except that there is an extra input (or constraint)–existing large software is never complete and continues to evolve. As it evolves, it grows more complex unless some action is taken to reduce this complexity.

Since software demonstrates regular behavior and trends, these can be measured. Attempts to develop predictive models to estimate maintenance effort have been made, and, as a result, useful management tools have been developed.

2.6 Categories of Maintenance

Maintenance consists of four parts:

- Corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered problems. It deals with fixing bugs in the code.
- Adaptive maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment. It deals with adapting the software to new environments.
- Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability. It deals with updating the software according to changes in user requirements.
- Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults. It deals with updating documentation and making the software more maintainable.

All changes to the system can be characterized by these four types of maintenance. Corrective maintenance is 'traditional maintenance' while the other types are considered as 'software evolution'.

3 Key Issues in Software Maintenance

A number of key issues must be dealt with to ensure the effective maintenance of software. It is important to understand that software maintenance provides unique technical and management challenges for software engineers. Trying to find a fault in software containing 500K lines of code that the software engineer did not develop is a good example. Similarly, competing with software developers for resources is a constant battle. Planning for a future release, while coding the next release and sending out emergency patches for the current release, also creates a challenge. The following section presents some of the technical and management issues related to software maintenance. They have been grouped under the following topic headings:

- Technical issues
- Management issues
- Cost estimation and Measures

3.1 Technical Issues

3.1.1 Limited understanding

Limited understanding refers to how quickly a software engineer can understand where to make a change or a correction in software which this individual did not develop. Research indicates that some 40% to 60% of the maintenance effort is devoted to understanding the software to be modified. Thus, the topic of software comprehension is of great interest to software engineers. Comprehension is more difficult in text-oriented representation, in source code, for example, where it is often difficult to trace the evolution of software through its releases/versions if changes are not documented and when the developers are not available to explain it, which is often the case. Thus, software engineers may initially have a limited understanding of the software, and much has to be done to remedy this.

3.1.2 Testing

The cost of repeating full testing on a major piece of software can be significant in terms of time and money. Regression testing, the selective retesting of a software or component to verify that the modifications have not caused unintended effects, is important to maintenance. As well, finding time to test is often difficult. There is also the challenge of coordinating tests when different members of the maintenance team are working on different problems at the same time. When software performs critical functions, it may be impossible to bring it offline to test.

3.1.3 Impact analysis

Impact analysis describes how to conduct, cost effectively, a complete analysis of the impact of a change in existing software. Maintainers must possess an intimate knowledge of the software's structure and content. They use that knowledge to perform impact analysis, which identifies all systems and software products affected by a software change request and develops an estimate of the resources needed to accomplish the change. Additionally, the risk of making the change is determined. The change request, sometimes called a modification request (MR) and often called a problem report (PR), must first be analyzed and translated into software terms. It is performed after a change request enters the software configuration management process. Arthur states that the objectives of impact analysis are:

- Determination of the scope of a change in order to plan and implement work
- Development of accurate estimates of resources needed to perform the work
- Analysis of the cost/benefits of the requested change
- Communication to others of the complexity of a given change

The severity of a problem is often used to decide how and when a problem will be fixed. The software engineer then identifies the affected components. Several potential solutions are provided and then a recommendation is made as to the best course of action.

Software designed with maintainability in mind greatly facilitates impact analysis.

3.1.4 Maintainability

How does one promote and follow up on maintainability issues during development? The IEEE [IEEE610.12-90] defines maintainability as the ease with which software can be maintained, enhanced, adapted, or corrected to satisfy specified requirements. ISO/IEC defines maintainability as one of the quality characteristics (ISO9126-01).

Maintainability sub-characteristics must be specified, reviewed, and controlled during the software development activities in order to reduce maintenance costs. If this is done successfully, the maintainability of the software will improve. This is often difficult to achieve because the maintainability sub-characteristics are not an important focus during the software development process. The developers are preoccupied with many other things and often disregard the maintainer's requirements. This in turn can, and often does, result in a lack of system documentation, which is a leading cause of difficulties in program comprehension and impact analysis. It has also been observed that the presence of systematic and mature processes, techniques, and tools helps to enhance the maintainability of a system.

3.2 Management Issues

3.2.1 Alignment with organizational objectives

Organizational objectives describe how to demonstrate the return on investment of software maintenance activities. Bennett states that "initial software development is usually project-based, with a defined time scale and budget. The main emphasis is to deliver on time and within budget to meet user needs. In contrast, software maintenance often has the objective of extending the life of software for as long as possible. In addition, it may be driven by the need to meet user demand for software updates and enhancements. In both cases, the return on investment is much less clear, so that the view at senior management level is often of a major activity consuming significant resources with no clear quantifiable benefit for the organization."

3.2.2 Staffing

Staffing refers to how to attract and keep software maintenance staff. Maintenance is often not viewed as glamorous work. Deklava provides a list of staffing-related problems based on survey data. As a result, software maintenance personnel are frequently viewed as "second-class citizens" and morale therefore suffers.

3.2.3 Process

Software process is a set of activities, methods, practices, and transformations which people use to develop and maintain software and the associated products. At the process level, software maintenance activities share much in common with software development (for example, software configuration management is a crucial activity in both). Maintenance also requires several activities which are not found in software development. These activities present challenges to management.

3.2.4 Organizational aspects of maintenance

Organizational aspects describe how to identify which organization and/or function will be responsible for the maintenance of software. The team that develops the software is not necessarily assigned to maintain the software once it is operational.

In deciding where the software maintenance function will be located, software engineering organizations may, for example, stay with the original developer or go to a separate team (or maintainer). Often, the maintainer option is chosen to ensure that the software runs properly and evolves to satisfy changing user needs. Since there are many pros and cons to each of these options, the decision should be made on a case-by-case basis. What is important is the delegation or assignment of the maintenance responsibility to a single group or person, regardless of the organization's structure.

3.2.5 Outsourcing

Outsourcing of maintenance is becoming a major industry. Large corporations are outsourcing entire portfolios of software systems, including software maintenance. More often, the outsourcing option is selected for less mission-critical software, as companies are unwilling to lose control of the software used in their core business. Carey reports that some will outsource only if they can find ways of maintaining strategic control. However, control measures are hard to find. One of the major challenges for the outsourcers is to determine the scope of the maintenance services required and the contractual details. McCracken states that 50% of outsourcers provide services without any clear service-level agreement. Outsourcing companies typically spend a number of months assessing the software before they will enter into a contractual relationship. Another challenge identified is the transition of the software to the outsourcer.

3.3 Maintenance Cost Estimation

Software engineers must understand the different categories of software maintenance, discussed above, in order to address the question of estimating the cost of software maintenance. For planning purposes, estimating costs is an important aspect of software maintenance.

3.3.1 Cost estimation

It was mentioned in "Impact Analysis", that impact analysis identifies all systems and software products affected by a software change request and develops an estimate of the resources needed to accomplish that change.

Maintenance cost estimates are affected by many technical and non-technical factors. ISO/IEC14764 states that "the two most popular approaches to estimating resources for software maintenance are the use of parametric models and the use of experience". Most often, a combination of these is used.

3.3.2 Parametric models

Some work has been undertaken in applying parametric cost modeling to software maintenance. Significance is that data from past projects are needed in order to use the models. Jones discusses all aspects of estimating costs, including function points (IEEE14143.1-00), and provides a detailed chapter on maintenance estimation.

3.3.3 Experience

Experience, in the form of expert judgment (using the Delphi technique, for example), analogies, and a work breakdown structure, are several approaches which should be used to augment data from parametric models. Clearly the best approach to maintenance estimation is to combine empirical data and experience. These data should be provided as a result of a measurement program.

3.4 Software Maintenance Measurement

Grady and Caswell discuss establishing a corporate-wide software measurement program, in which software maintenance measurement forms and data collection are described. The Practical Software and Systems Measurement (PSM) project describes an issue-driven measurement process that is used by many organizations and is quite practical.

There are software measures that are common to all endeavors, the following categories of which the Software Engineering Institute (SEI) has identified: size; effort; schedule; and quality. These measures constitute a good starting point for the maintainer.

3.4.1 Specific Measures

Abran presents internal benchmarking techniques to compare different internal maintenance organizations. The maintainer must determine which measures are appropriate for the organization in question. IEEE1219-98 suggests measures which are more specific to software maintenance measurement programs. That list includes a number of measures for each of the four sub-characteristics of maintainability:

- Analyzability: Measures of the maintainer's effort or resources expended in trying to diagnose deficiencies or causes of failure, or in identifying parts to be modified
- Changeability: Measures of the maintainer's effort associated with implementing a specified modification
- Stability: Measures of the unexpected behavior of software, including that encountered during testing
- Testability: Measures of the maintainer's and users' effort in trying to test the modified software

Certain measures of the maintainability of software can be obtained using available commercial tools.

4 Maintenance Process

The Maintenance Process subarea provides references and standards used to implement the software maintenance process. The Maintenance Activities topic differentiates maintenance from development and shows its relationship to other software engineering activities.

4.1 Maintenance Processes

Maintenance processes provide needed activities and detailed inputs/outputs to those activities, and are described in software maintenance standards IEEE 1219 and ISO/IEC 14764.

The maintenance process model described in the Standard for Software Maintenance (IEEE1219) starts with the software maintenance effort during the post-delivery stage and discusses items such as planning for maintenance.

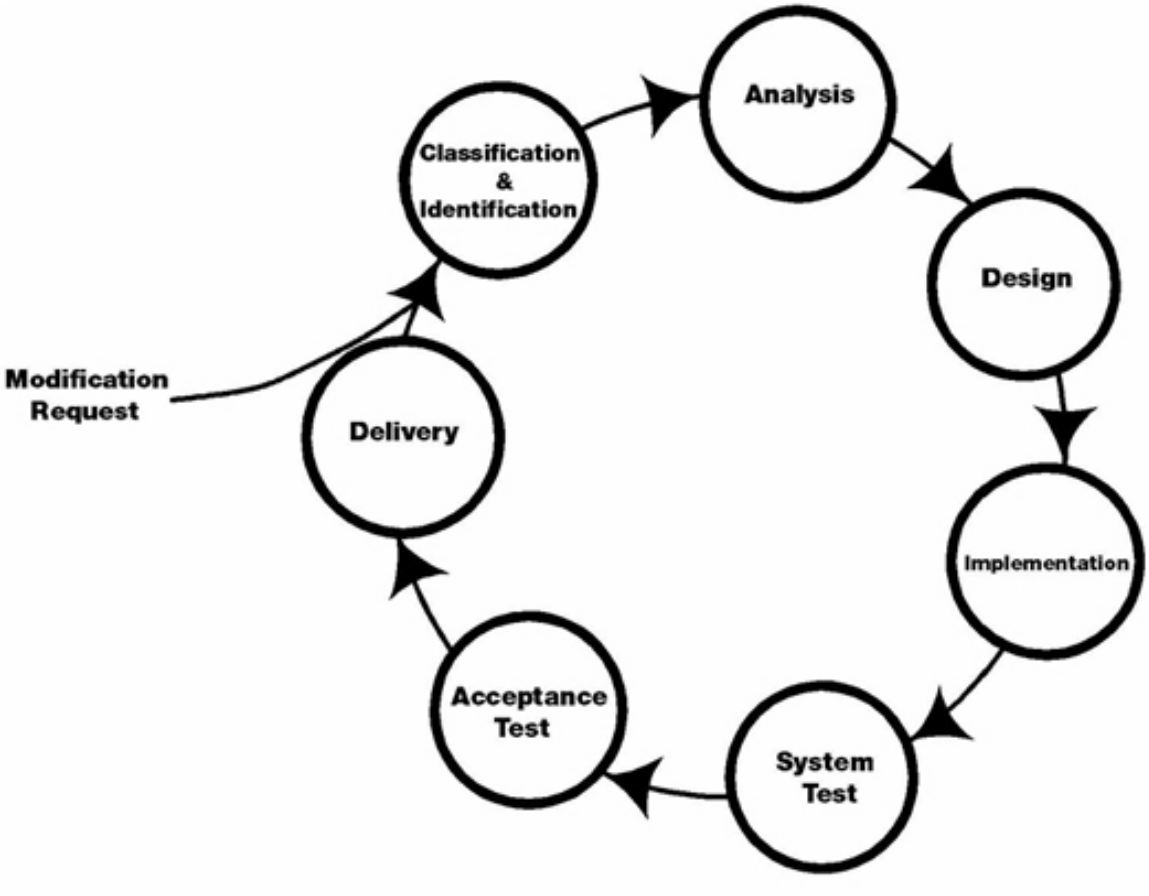


Figure 2

ISO/IEC 14764 is an elaboration of the IEEE/EIA 12207.0-96 maintenance process. The activities of the ISO/IEC maintenance process are similar to those of the IEEE, except that they are aggregated a little differently.

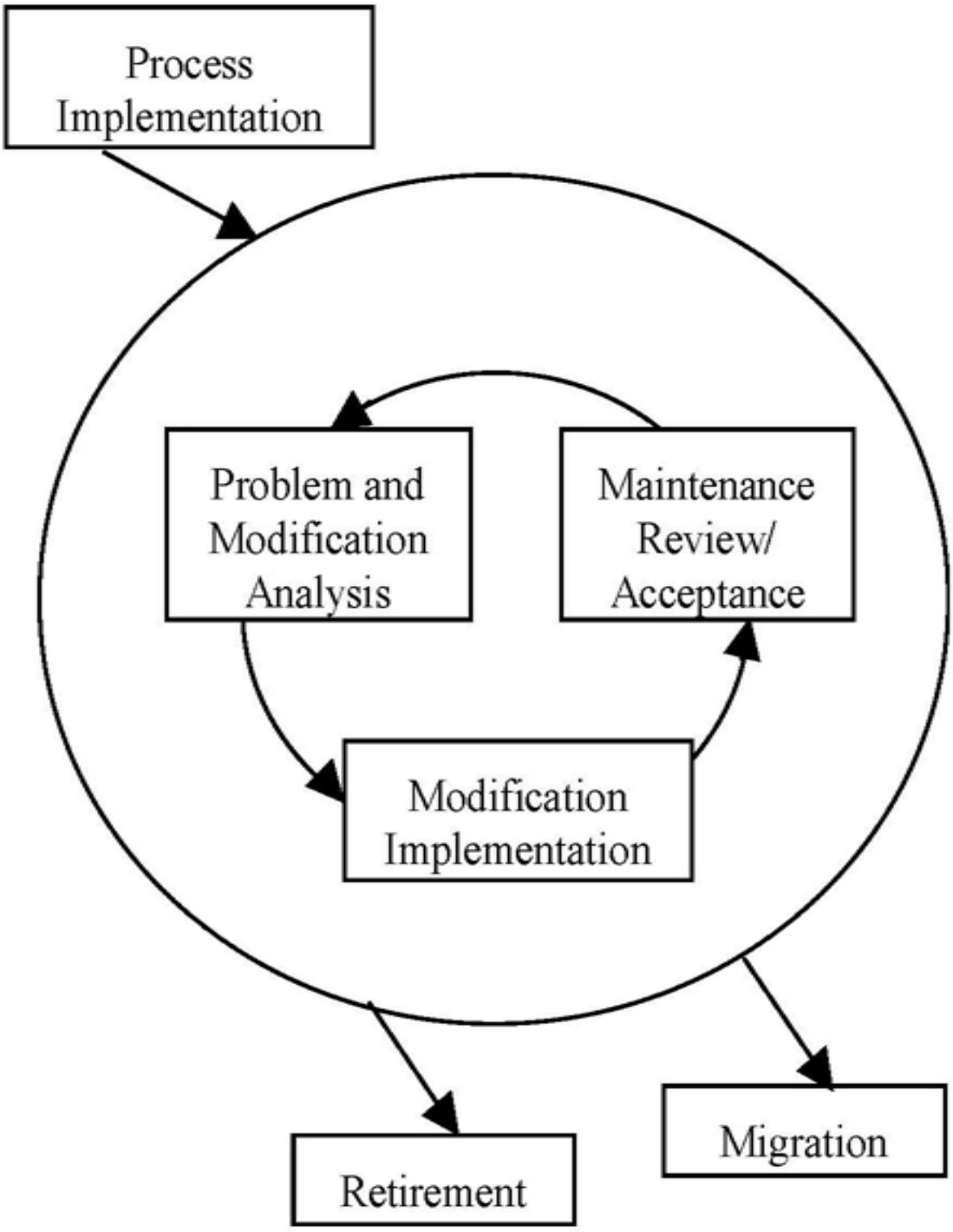

Process Implementation
Problem and Modification Analysis
Maintenance Review/ Acceptance
Modification Implementation
Retirement
Migration

Figure 3

Each of the ISO/IEC 14764 primary software maintenance activities is further broken down into tasks, as follows.

- Process Implementation
- Problem and Modification Analysis
- Modification Implementation
- Maintenance Review/Acceptance
- Migration
- Software Retirement

4.2 Maintenance Activities

As already noted, many maintenance activities are similar to those of software development. Maintainers perform analysis, design, coding, testing, and documentation. They must track requirements in their activities just as is done in development, and update documentation as baselines change. ISO/IEC14764 recommends that, when a maintainer refers to a similar development process, he must adapt it to meet his specific need. However, for software maintenance, some activities involve processes unique to software maintenance.

4.2.1 Unique activities

There are a number of processes, activities, and practices that are unique to software maintenance, for example:

- Transition: a controlled and coordinated sequence of activities during which software is transferred progressively from the developer to the maintainer.
- Modification Request Acceptance/Rejection: modification request work over a certain size/effort/complexity may be rejected by maintainers and rerouted to a developer.
- Modification Request and Problem Report Help Desk: an end-user support function that triggers the assessment, prioritization, and costing of modification requests.
- Impact Analysis.
- Software Support: help and advice to users concerning a request for information (for example, business rules, validation, data meaning and ad-hoc requests/reports).
- Service Level Agreements (SLAs) and specialized (domain-specific) maintenance contracts which are the responsibility of the maintainers.

4.2.2 Supporting activities

Maintainers may also perform supporting activities, such as software maintenance planning, software configuration management, verification and validation, software quality assurance, reviews, audits, and user training.

4.2.3 Maintenance planning activity

An important activity for software maintenance is planning, and maintainers must address the issues associated with a number of planning perspectives:

- Business planning (organizational level)
- Maintenance planning (transition level)
- Release/version planning (software level)
- Individual software change request planning (request level)

At the individual request level, planning is carried out during the impact analysis. The release/version planning activity requires that the maintainer:

- Collect the dates of availability of individual requests
- Agree with users on the content of subsequent releases/versions
- Identify potential conflicts and develop alternatives
- Assess the risk of a given release and develop a back-out plan in case problems should arise
- Inform all the stakeholders

Whereas software development projects can typically last from some months to a few of years, the maintenance phase usually lasts for many years. Making estimates of resources is a key element of maintenance planning. Those resources should be included in the developers' project planning budgets. Software maintenance planning should begin with the decision to develop a new system and should consider quality objectives (IEEE1061-98). A concept document should be developed, followed by a maintenance plan.

The concept document for maintenance should address:

- The scope of the software maintenance
- Adaptation of the software maintenance process
- Identification of the software maintenance organization
- An estimate of software maintenance costs

The next step is to develop a corresponding software maintenance plan. This plan should be prepared during software development, and should specify how users will request software modifications or report problems. Software maintenance planning is addressed in IEEE 1219 and ISO/IEC 14764. ISO/IEC14764 provides guidelines for a maintenance plan.

Finally, at the highest level, the maintenance organization will have to conduct business planning activities (budgetary, financial, and human resources) just like all the other divisions of the organization.

4.2.4 Software configuration management

The IEEE Standard for Software Maintenance, IEEE 1219, describes software configuration management as a critical element of the maintenance process. Software configuration management procedures should provide for the verification, validation, and audit of each step required to identify, authorize, implement, and release the software product.

It is not sufficient to simply track Modification Requests or Problem Reports. The software product and any changes made to it must be controlled. This control is established by implementing and enforcing an approved software configuration management (SCM) process. SCM for software maintenance is different from SCM for software development in the number of small changes that must be controlled on operational software. The SCM process is implemented by developing and following a configuration management plan and operating procedures. Maintainers participate in Configuration Control Boards to determine the content of the next release/version.

4.2.5 Software quality

It is not sufficient, either, to simply hope that increased quality will result from the maintenance of software. It must be planned and processes implemented to support the maintenance process. The activities and techniques for Software Quality Assurance (SQA), V&V, reviews, and audits must be selected in concert with all the other processes to achieve the desired level of quality. It is also recommended that the maintainer adapt the software development processes, techniques and deliverables, for instance testing documentation, and test results.

5 Techniques for Maintenance

This subarea introduces some of the generally accepted techniques used in software maintenance.

5.1 Program Comprehension

Programmers spend considerable time in reading and understanding programs in order to implement changes. Code browsers are key tools for program comprehension. Clear and concise documentation can aid in program comprehension.

5.2 Reengineering

Reengineering is defined as the examination and alteration of software to reconstitute it in a new form, and includes the subsequent implementation of the new form. Dorfman and Thayer state that reengineering is the most radical (and expensive) form of alteration. Others believe that reengineering can be used for minor changes. It is often not undertaken to improve maintainability, but to replace aging legacy software. Arnold provides a comprehensive compendium of topics, for example: concepts, tools and techniques, case studies, and risks and benefits associated with reengineering.

5.3 Reverse engineering

Reverse engineering is the process of analyzing software to identify the software's components and their interrelationships and to create representations of the software in another form or at higher levels of abstraction. Reverse engineering is passive; it does not change the software, or result in new software. Reverse engineering efforts produce call graphs and control flow graphs from source code. One type of reverse engineering is redocumentation. Another type is design recovery. Refactoring is program transformation which reorganizes a program without changing its behavior, and is a form of reverse engineering that seeks to improve program structure.

Finally, data reverse engineering has gained in importance over the last few years where logical schemas are recovered from physical databases.

6 Tools

6.1 Introduction

A software maintenance tool is an artifact that supports a software maintainer in performing a task. The use of tools for software maintenance simplifies the tasks and increases efficiency and productivity.

There are several criteria for selecting the right tool for the task. These criteria are capability, features, cost/benefit, platform, programming language, ease of use, openness

of architecture, stability of vendor, and organizational culture.

Capability decides whether the tool is capable of fulfilling the task. Once it has been decided that a method can benefit from being automated, then the features of the tool need to be considered for the job.

The tool must be analyzed for the benefits it brings against its cost. The benefit indicators of a tool are quality, productivity, responsiveness, and cost reduction. The environment that the tool runs on is called the platform. The language of the source code is called the programming language. It's important to select a tool that supports a language that is an industry standard.

The tool should have a similar feel to the ones that the users are already familiar with. The tool should have the ability to be integrated with different vendors' tools. This will help when a tool will need to run with other tools. The openness of the architecture plays an important role when the maintenance problem is complex. Therefore, it is not always sufficient to use only one tool. There may need to be multiple tools running together.

It is also important to consider the vendor's credibility. The vendor should be capable of supporting the tool in the future. If the vendor is not stable, the vendor could run out of business and not be able to support the tool. Another important factor is the culture of the organization. Every culture has its own work pattern. Therefore, it is important to take into consideration whether the tool is going to be accepted by the target users.

The chosen tools must support program understanding and reverse engineering, testing, configuration management, and documentation.

Selecting a tool that promotes understanding is very important in the implementation of change since a large amount of time is used to study and understand programs.

Tools for reverse engineering also accomplish the same goal. The tools mainly consist of visualization tools, which assist the programmer in drawing a model of the system.

Examples of program understanding and reverse engineering tools include the program slicer static analyzer, dynamic analyzer, cross-referencer and dependency analyzer.

Slicing is the mechanical process of marking all the sections of a program text that may influence the value of a variable at a given point in the program. Program slicing helps the programmers select and view only the parts of the program that are affected by the changes. Static analyzer is used in analyzing the different parts if the program such as modules, procedures, variables, data elements, objects and classes. A static analyzer allows general viewing of the program text and generates summaries of contents and usage of selected elements in the program text, such as variables or objects.

A dynamic analyzer could be used to analyze the program while it is executing. A data flow analyzer is a static analysis tool that allows the maintainer to track all possible data flow and control flow paths in the program. It allows analysis of the program to better outline the underlying logic of the program. It also helps display the relationship between components of the system. A cross-referencer produces information on the usage of a program. This tool helps the user focus on the parts that are affected by the change.

A dependency analyzer assists the maintainer to analyze and understand the interrelationships between entities in a program. Such a tool provides capabilities to set up and query the database of the dependencies in a program. It also provides graphical representations of the dependencies. Testing is the most time consuming and demanding task in software maintenance.

Therefore, it could benefit the most from tools. A test simulator tool helps the maintainer

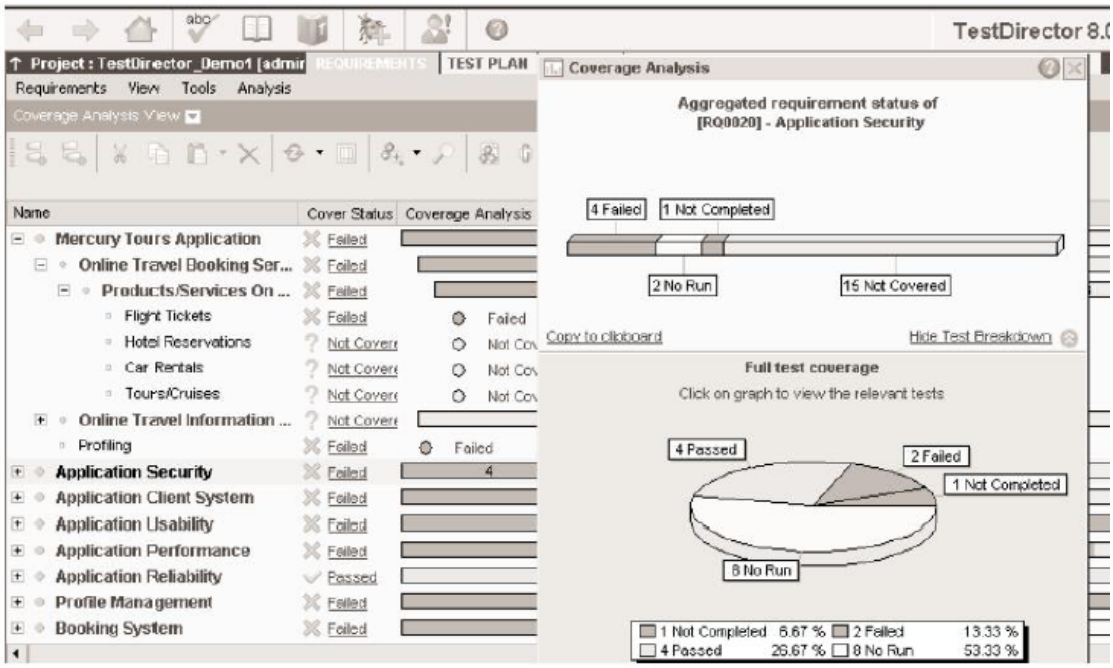
to test the effects of the change in a controlled environment before implementing the change on the actual system. A test case generator produces test data that is used to test the functionality of the modified system, while a test path generator helps the maintainer to find all the data flow and control flow paths affected by the changes.

Configuration management benefits from automated tools. Configuration management and version control tools help store the objects that form the software system. A source control system is used to keep a history of the files so that versions can be tracked and the programmer can keep track of the file changes.

6.2 Commercially available products

There are numerous products on the market available for software maintenance. One type

of product is bug tracking tools, which play an important role in maintenance. Bugzilla by the Mozilla Foundation is an example of such a tool. Other bug tracking products are Test Director by Mercury Interactive, Silk Radar by Segue Software, SQA Manager by Rational software, and QA director by Compuware.



TestDirector's Requirements Manager links test cases to testing requirements, ensuring traceability.

Figure 4

ProTeus III Expert CMMS by Eagle Technology, Inc. is a maintenance software package that lets users schedule preventative maintenance, generate automatic work orders, document equipment maintenance history, track assets and inventory, track personnel, create purchase orders, and generate reports. Microsoft Visual Source Safe is a source control system tool that is used by configuration management.

Products that are specific to programming languages are CCFinder and JAAT which

is specifically designed for JAVA programs. CCFinder identifies code clones in JAVA program. JAAT executes alias analysis for JAVA programs. For C++ programs, there is a tool called OCL query-based debugger which is a tool to debug C++ programs using queries formulated in the object constraint language.

6.3 Summary of tools

The task of software maintenance has become so vital and complex that automated support is required to do it effectively. The use of tools simplifies tasks, increase efficiency and productivity. There are numerous tools available on the market for maintenance.