Software Evolution and Maintenance

Chapter 1
Software Evolution and Maintenance Concepts

- Evolution versus maintenance
- Software Evolution
- Types of Software Maintenance
- Evolution & Maintenance Models
- Configuration Management
- Reengineering
- Legacy Systems
- Refactoring
- Software Reuse

Another flaw in the human character is that everybody wants to build and nobody wants to do maintenance.
—Kurt Vonnegut, Jr

Software Evolution and Maintenance Concepts

- ♦ The concept of ***software maintenance*** means preventing software from failing to deliver the intended functionalities by means of bug fixing.
- ♦ The concept of ***software evolution*** means a continual change from a lesser, simpler, or worse state to a higher or better state.
- ➢ Bennett and Xu made further distinctions between the two as follows:
- ♦ All support activities carried out *after* delivery of software are put under the category of *maintenance*.
- ♦ All activities carried out to effect changes in requirements are put under the category of *evolution*.

Software Evolution and Maintenance Concepts

Software maintenance comprises all activities associated with the process of changing software for the purposes of:

- ♦ fixing bugs; and/or
- ♦ improving the design of the system
  - ♦ future changes to the system are less expensive.

Evolution of software systems means creating new but related designs from existing ones. The objectives include:

- ♦ supporting *new functionalities*
- ♦ making the *system perform better*
- ♦ making the *system run on a different OS*

Software Evolution: SPE Taxonomy

- The abbreviation SPE refers to S (Specified), P (Problem), and E (Evolving) programs.

- **S-type programs:** S-type programs have the following characteristics:
  - All the nonfunctional and functional program properties that are important to its stakeholders are *formally* and *completely* defined.
  - Correctness of the program with respect to its formal specification is the *only* criterion of the acceptability of the solution to its stakeholders.
- **P-type programs:** With many real problems, the system outputs are accurate to a constrained level of precision. The concept of correctness is difficult to define in those programs.

Software Evolution: SPE Taxonomy

- **E-type programs:** An E-type program is one that is embedded in the real world and it changes as the world does.
- These programs mechanize a human or society activity, make simplifying assumptions, and interface with the external world by requiring or providing services.
- An E-type system is to be regularly adapted to:

- (i) stay true to its domain of application
- (ii) remain compatible with its executing environment
- (iii) meet the goals and expectations of its stakeholders

Software Evolution

- Fundamental work in the field of software evolution was done by Lehman and his collaborators.
  - Based on empirical studies, Lehman and his collaborators formulated some observations and they introduced them as ***laws of evolution***.
  - The “laws” themselves have “evolved” from *three* in 1974 to *eight* by 1997.
- Those laws are the results of studies of the evolution of large-scale proprietary or closed source software (CSS) systems.
- Lehman and his colleagues have postulated eight “laws” over 20 years starting from the mid-1970s to explain some key observations about the evolution of E-type software systems

Software Evolution

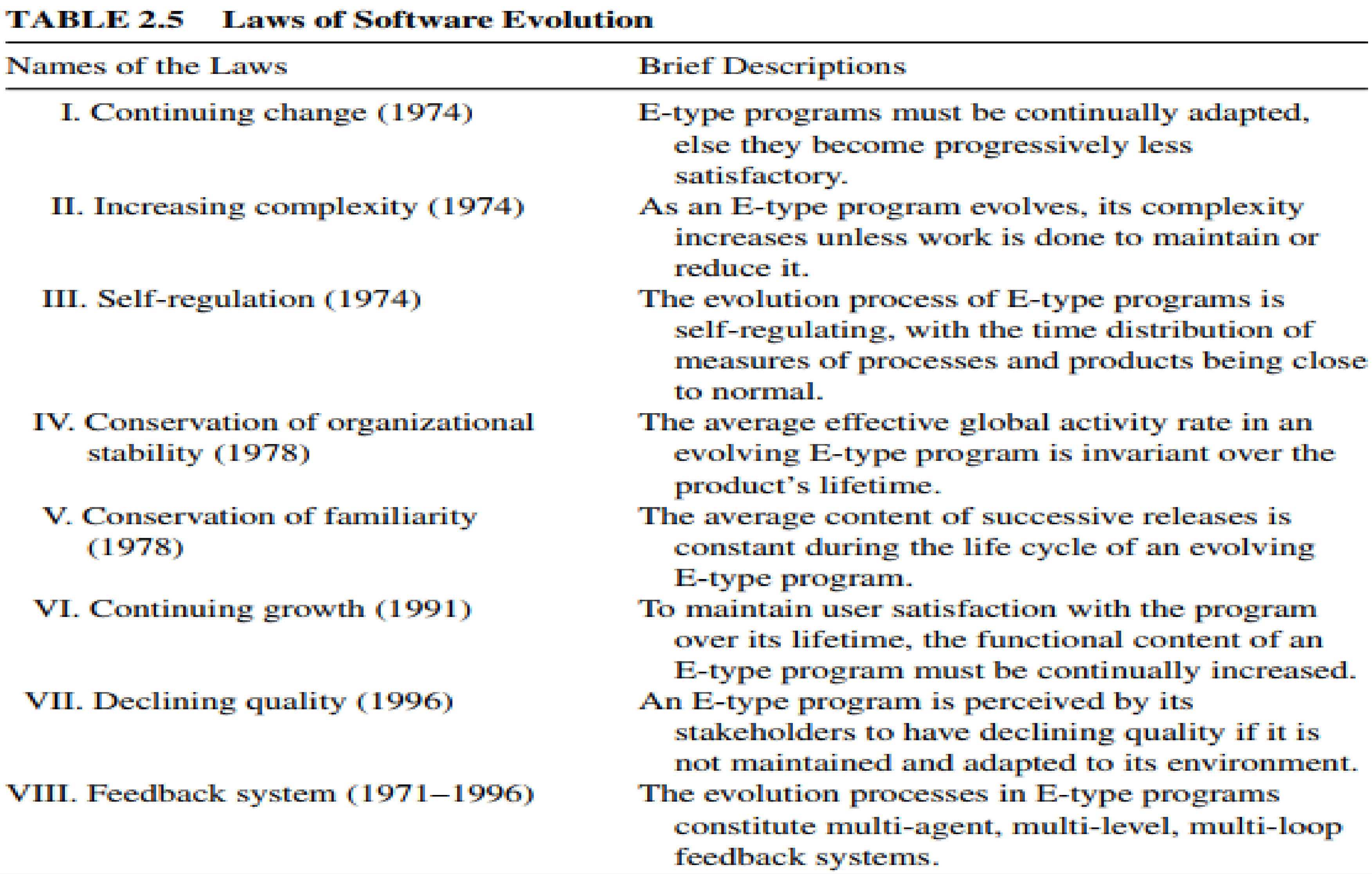


TABLE 2.5 Laws of Software Evolution

Names of the Laws	Brief Descriptions
I. Continuing change (1974)	E-type programs must be continually adapted, else they become progressively less satisfactory.
II. Increasing complexity (1974)	As an E-type program evolves, its complexity increases unless work is done to maintain or reduce it.
III. Self-regulation (1974)	The evolution process of E-type programs is self-regulating, with the time distribution of measures of processes and products being close to normal.
IV. Conservation of organizational stability (1978)	The average effective global activity rate in an evolving E-type program is invariant over the product's lifetime.
V. Conservation of familiarity (1978)	The average content of successive releases is constant during the life cycle of an evolving E-type program.
VI. Continuing growth (1991)	To maintain user satisfaction with the program over its lifetime, the functional content of an E-type program must be continually increased.
VII. Declining quality (1996)	An E-type program is perceived by its stakeholders to have declining quality if it is not maintained and adapted to its environment.
VIII. Feedback system (1971–1996)	The evolution processes in E-type programs constitute multi-agent, multi-level, multi-loop feedback systems.

Software Maintenance-Development

Software Maintenance Vs. Development

Development	Maintenance
development is requirements driven	maintenance is event driven
Begins with designing and implementing a system to deliver functional and nonfunctional requirements.	A maintenance task is scheduled in response to an event.
	Events : • change request • needs to fix a set of bugs

Types of Software Maintenance

- There are defects in delivered software applications, because defect removal and quality control processes are not perfect.
  - Therefore, maintenance is needed to repair those defects in released software.
  - E. Burton Swanson initially defined three categories of software maintenance activities, namely, corrective, adaptive, and perfective.
- Those definitions were later incorporated into the standard software engineering–software life cycle processes–Maintenance and introduced a fourth category called preventive maintenance.
  - The reader may note that some researchers and developers view preventive maintenance as a subset of perfective maintenance.

Types of Software Maintenance

- In the following subsections, we explain maintenance activities from three viewpoints:
  - ➢Intention-based classification of software maintenance activities;
  - ➢Activity-based classification of software maintenance activities
  - ➢Evidence-based classification of software maintenance activities.

Types of Maintenance …

Intention-based classification of software maintenance activities

- ✓ In this classification, one categorizes maintenance activities into four groups based on what we intend to achieve with those activities.
- ✓ Based on the Standard for Software Engineering–Software Maintenance, ISO/IEC 14764 [3], the four categories of maintenance activities are corrective, adaptive, perfective, and preventive.

I. **Corrective maintenance:** the purpose of corrective maintenance is to correct failures: processing failures and performance failures.

- A program producing a wrong output is an example of processing failure.
- A program not being able to meet real-time requirements is an example of performance failure.

  Example: correcting a program that aborts or produces incorrect results.
- corrective maintenance is a ***reactive*** process, which means that corrective maintenance is performed after detecting defects with the system.

Types of Maintenance …

Intention-based …

II. Adaptive maintenance

- The purpose of adaptive maintenance is to enable the system to adapt to changes in its data environment or processing environment.
- modifies the software to properly interface with a changing or changed environment

Some generic examples are:

1. changing the system to support new hardware configuration;
2. converting the system from batch to online operation;
3. changing the system to be compatible with other applications.
4. an application software on a smartphone can be enhanced to support WiFi-based communication in addition to its present 3G cellular communication

Types of Maintenance …

Intention-based …

III. Perfective maintenance

- The purpose of perfective maintenance is to make a variety of improvements, namely: user experience, processing efficiency, and maintainability.
- For example:
  1. the program outputs can be made more readable for better user experience
  2. the program can be modified to make it faster, thereby increasing the processing efficiency
  3. the program can be restructured to improve its readability, so increasing its maintainability.
- In general, activities for perfective maintenance include:
  - restructuring of the code
  - creating and updating documentations
  - tuning the system to improve performance

Types of Maintenance …

Intention-based …

IV. **Preventive maintenance**

- The purpose of preventive maintenance is to prevent problems from occurring by modifying software products.
- For example
  - good programming styles can reduce the impact of change, thereby reducing the number of failure.
- Preventive maintenance is very often performed on safety critical and high available software systems
- software rejuvenation is a preventive maintenance measure to prevent, or at least postpone, the occurrences of failures due to continuously running the software system.
- *Software rejuvenation* is a proactive fault management technique aimed at cleaning up the system internal state to prevent the occurrence of more severe crash in the future

Types of Maintenance…

Activity-Based Classification of Software Maintenance

✓ In Kitchenham et al. organize maintenance modification activities based on the maintenance activity.

I. Corrections

Activities are designed to fix defects in the system, where a defect is a discrepancy between the expected behavior and the actual behavior of the system.

II. Enhancements

Activities are designed to effect changes to the system. The changes to the system do not necessarily modify the behavior of the system.

- enhancement activities that modify some of the existing requirements implemented
- enhancement activities that add new system requirements
- enhancement activities that modify the implementation without changing the requirements implemented by the system.

Types of Maintenance…

Evidence-Based Classification of Software Maintenance

The intention-based classification of maintenance activities was further refined by Chapin et al.

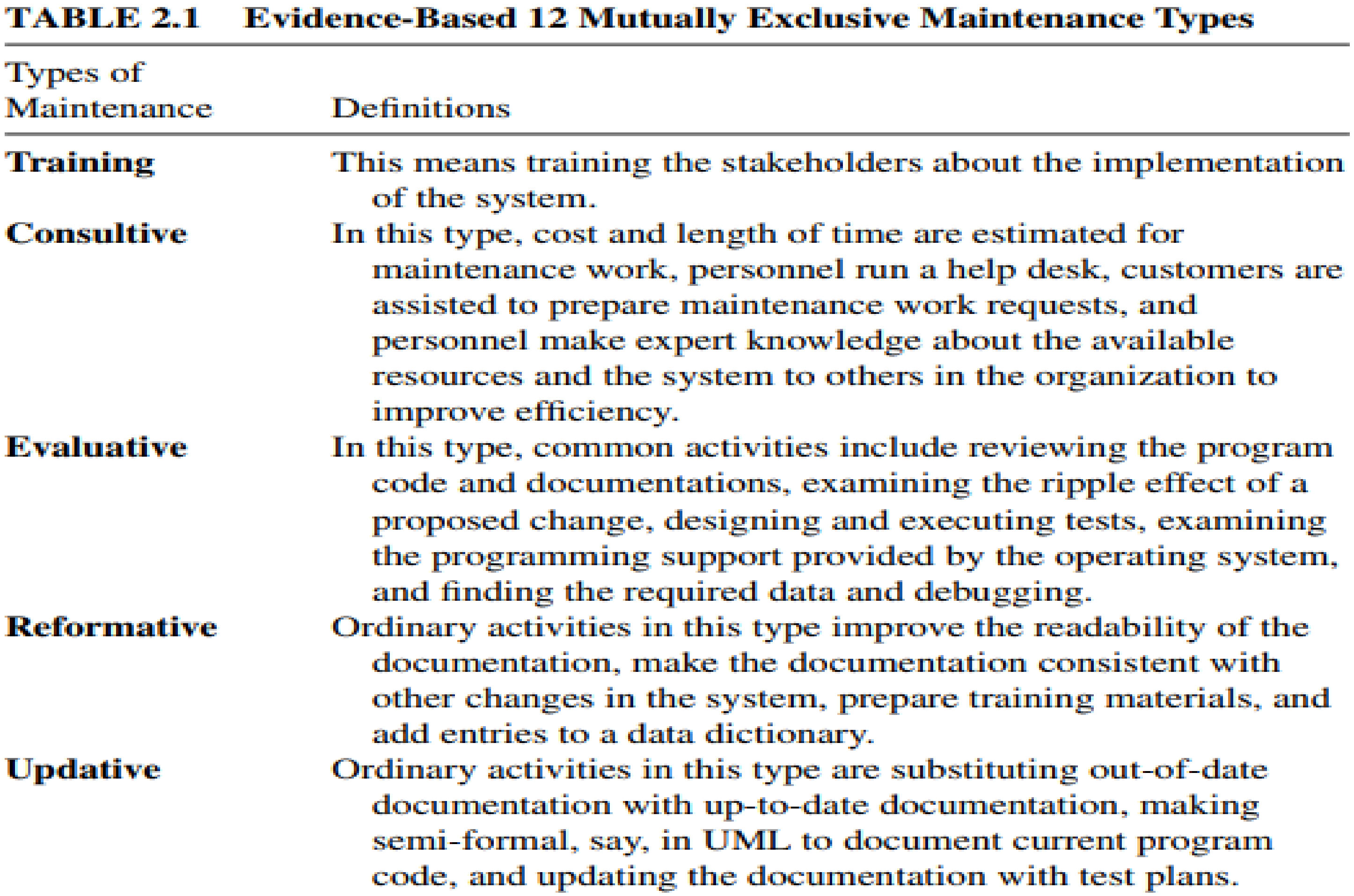
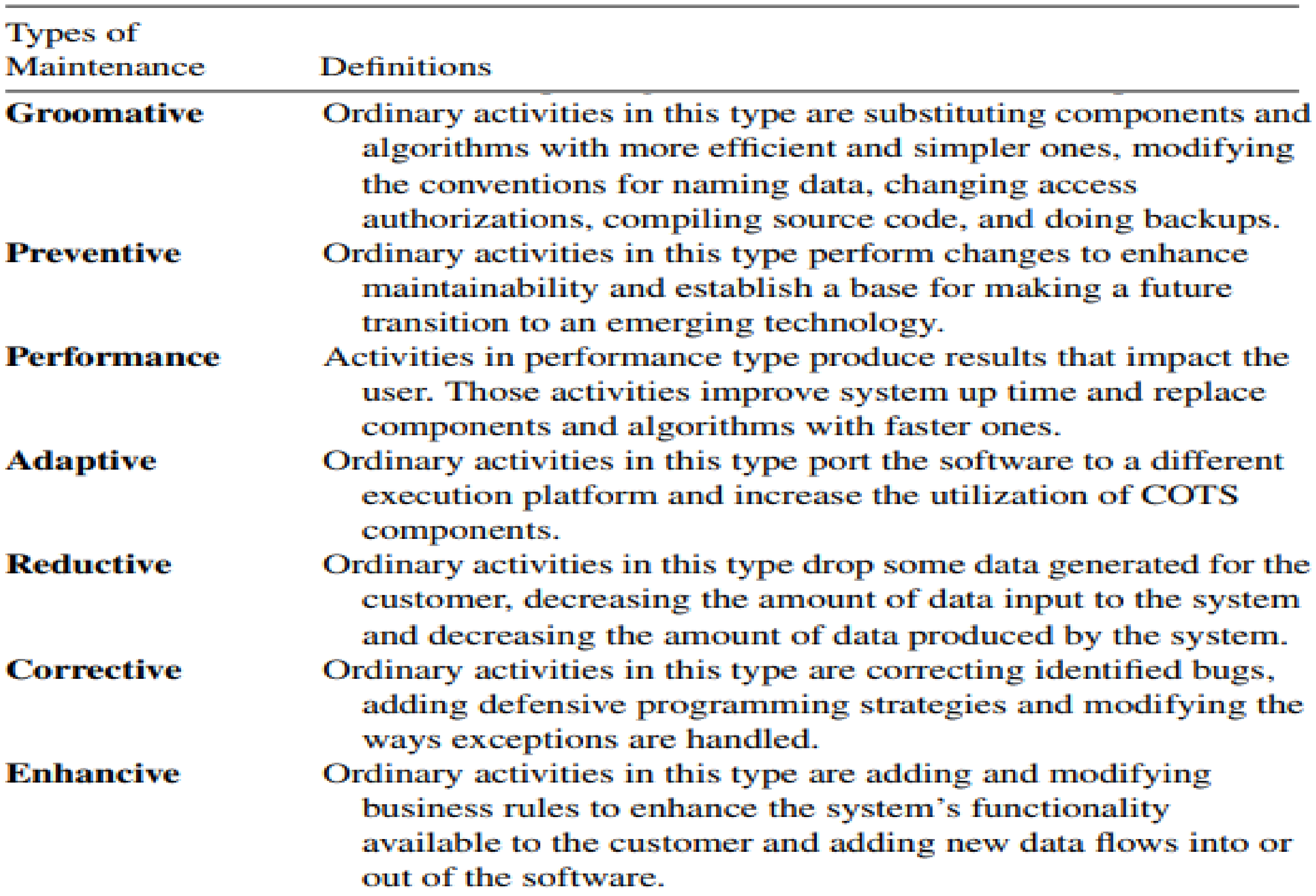


TABLE 2.1 Evidence-Based 12 Mutually Exclusive Maintenance Types

Types of Maintenance	Definitions
Training	This means training the stakeholders about the implementation of the system.
Consultive	In this type, cost and length of time are estimated for maintenance work, personnel run a help desk, customers are assisted to prepare maintenance work requests, and personnel make expert knowledge about the available resources and the system to others in the organization to improve efficiency.
Evaluative	In this type, common activities include reviewing the program code and documentations, examining the ripple effect of a proposed change, designing and executing tests, examining the programming support provided by the operating system, and finding the required data and debugging.
Reformative	Ordinary activities in this type improve the readability of the documentation, make the documentation consistent with other changes in the system, prepare training materials, and add entries to a data dictionary.
Updative	Ordinary activities in this type are substituting out-of-date documentation with up-to-date documentation, making semi-formal, say, in UML to document current program code, and updating the documentation with test plans.

Types of Maintenance…

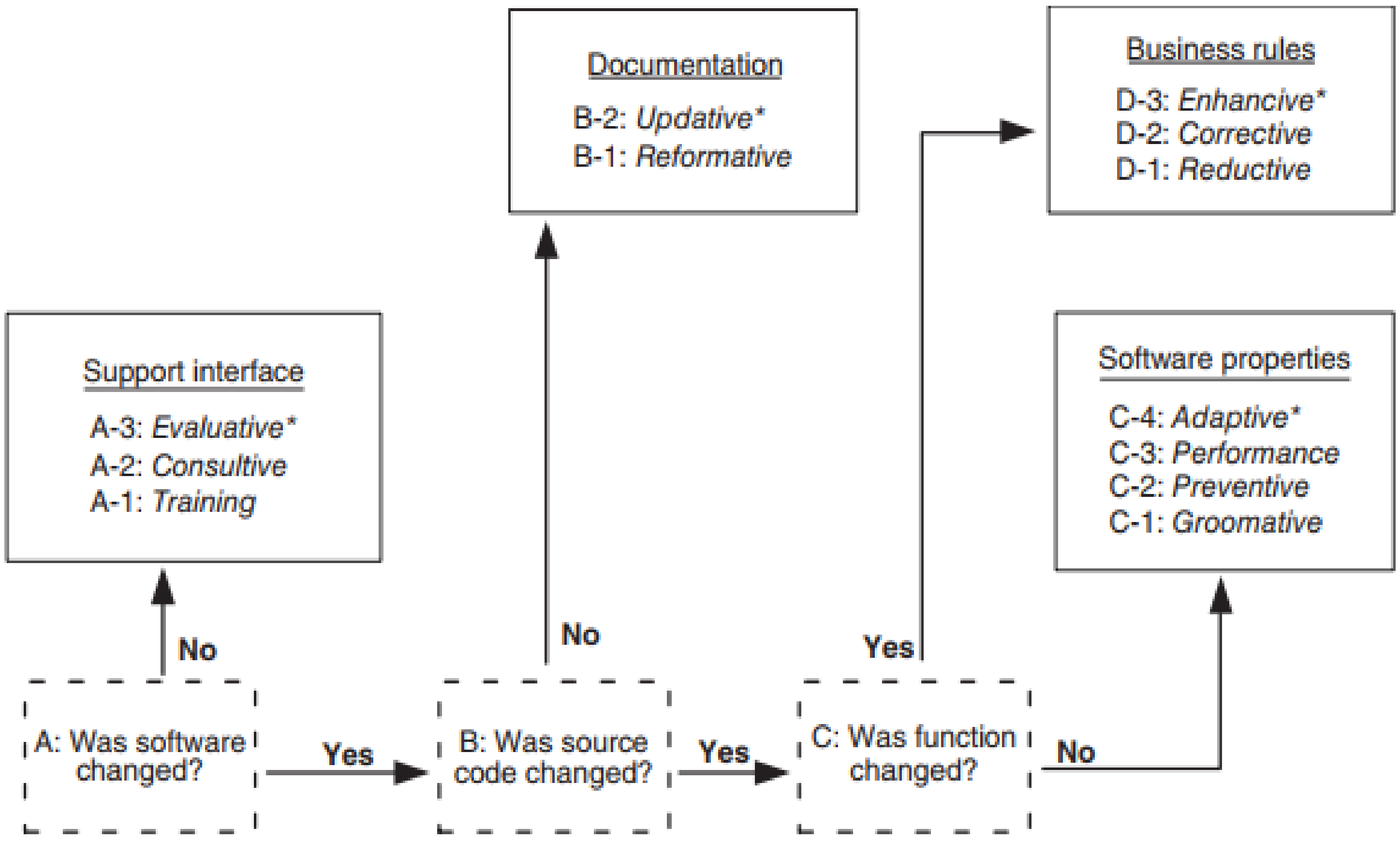
Evidence-Based…



Types of Maintenance	Definitions
Groomative	Ordinary activities in this type are substituting components and algorithms with more efficient and simpler ones, modifying the conventions for naming data, changing access authorizations, compiling source code, and doing backups.
Preventive	Ordinary activities in this type perform changes to enhance maintainability and establish a base for making a future transition to an emerging technology.
Performance	Activities in performance type produce results that impact the user. Those activities improve system up time and replace components and algorithms with faster ones.
Adaptive	Ordinary activities in this type port the software to a different execution platform and increase the utilization of COTS components.
Reductive	Ordinary activities in this type drop some data generated for the customer, decreasing the amount of data input to the system and decreasing the amount of data produced by the system.
Corrective	Ordinary activities in this type are correcting identified bugs, adding defensive programming strategies and modifying the ways exceptions are handled.
Enhancive	Ordinary activities in this type are adding and modifying business rules to enhance the system's functionality available to the customer and adding new data flows into or out of the software.

Types of Maintenance...

Evidence-Based...



Software Maintenance: COTS

- ♦ New components are developed by combining commercial off-the-shelf (COTS) components, custom-built (in-house) components, and open source software components.
- ♦ The major differences between component-based software systems (CBS) and custom-built software systems:
- ♦ *Skills of system maintenance teams:* Maintenance of CBS requires specialized skills to monitor and integrate COTS products.
- ♦ *Infrastructure and organization:* Running a support group for in-house products is necessary to manage a large product.
- ♦ *COTS maintenance cost:* This cost includes the costs of purchasing components, licensing components, upgrading components, and training maintenance personnel.
- ♦ *Larger user community:* COTS users are part of a broad community of users, and the community of users can be considered as a resource, which is a positive factor.

Software Maintenance : COTS...

- ♦ *Modernization:* vendors of COTS components keep pace with changing technology and continually update the components. As a result, the system does not become obsolete.
- ♦ *Split maintenance function:* A COTS product is maintained by its vendor, whereas the overall system that uses the COTS product is maintained by the system's host organization. As a result, multiple, independent maintenance teams exist. The advantage of COTS-based development is that the system maintainers receive additional support from the COTS vendors.
- ♦ *More complex planning:* If a system depends upon multiple technologies and COTS products, the unpredictability and risk of change become high, and planning becomes complicated because coordination among a large number of vendors is more difficult.

Evolution And Maintenance Models

- ♦ Software maintenance should have its own software maintenance life cycle (SMLC) model.
- ♦ A number of SMLC models with some variations are available in literature. Three common features of the SMLC models found in the literature are:
  - **Understanding the code**
  - **Modifying the code**
  - **Revalidating the code**
- ♦ There are three main software maintenance and evolution models view
  - **Reuse-oriented Model**
  - **Staged model of maintenance**
  - **Change mini-cycle models**

Evolution And Maintenance Models

I. Reuse-oriented Model

- A new version of the system can be created after the maintenance activities are implemented on some of the old system's components.
- Based on this concept, three process models for maintenance have been proposed by Basili:

- ***Quick fix model***. In this model, necessary changes are quickly made to the code and then to the accompanying documentation (Figure 3.2).
- ***Iterative enhancement model***. In this model, as illustrated in Figure 3.3, first changes are made to the highest level documents. Eventually, changes are propagated down to the code level.
- ***Full reuse model.*** In this model, as illustrated in Figure 3.4, a new system is built from components of the old system and others available in the repository

Evolution And Maintenance Models

II. The Staged Model For Closed Source Software

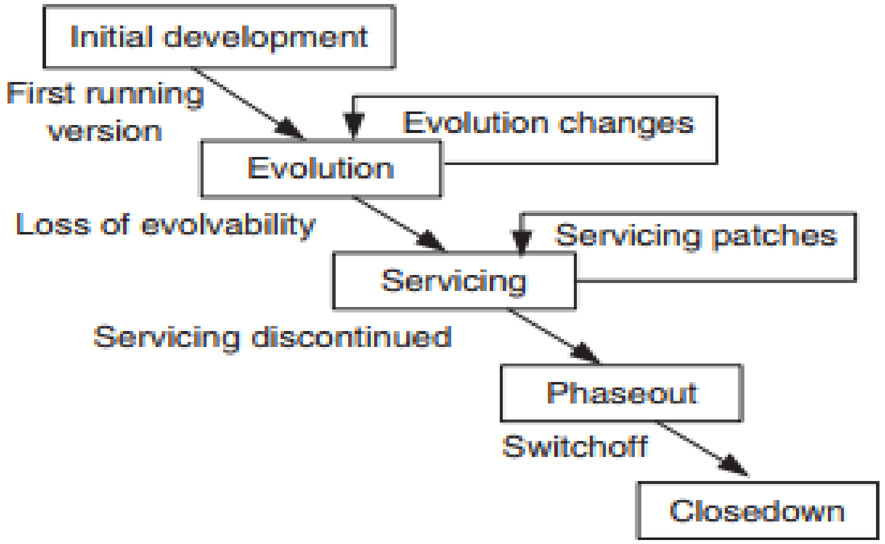


Fig: The simple staged model for the CSS life cycle

Evolution ...

- A different kind of software evolution model, called *staged model of maintenance and evolution*, has been proposed by Rajlich and Bennett.
- The model is descriptive in nature, and its primary objective is to improve the understanding of how long-lived software evolves.
- The model considers four distinct, sequential stages of the lifetime of a system, as explained below:

1. Initial development: When the initial version of the system is produced, detailed knowledge about the system is fresh. Before delivery of the system, it undergoes many changes. Eventually, a system architecture emerges and soon it stabilizes.
2. Evolution: Significant changes involve higher cost and higher risk. In the period immediately following the initial delivery, knowledge about the system is still almost fresh in the minds of the developers.

- In general, for many systems, their lifespan are spent in this stage, because the systems continue to be of importance to the organizations.

SOFTWARE EVOLUTION MODELS AND PROCESSES

3. Servicing: When the knowledge about the system has significantly decreased, the developers mainly focus on maintenance tasks, such as fixing bugs, whereas architectural changes are rarely effected.
4. Phaseout: When even minimal servicing of a system is not an option, the system enters its very final stage.
   - The organization decides to replace the system for various reasons:

     (i) it is too expensive to maintain the system; or

     (ii) there is a newer solution available.

Life Cycle Stage	**Maintenance Task**	**User Population**
Initial development	–	–
Evolution	Corrections, enhancements	Small
Servicing	Corrections	Growing
Phaseout	Corrections	Maximum
Closedown	Corrections	Declining

Staged Model Maintenance Task

III. Change Mini-cycle Model

- Software change is a fundamental ingredient of software evolution and maintenance.
- Let us revisit the first law of software evolution which is stated as:

  "A program undergoes continuing changes or becomes less useful. The change process continues until it becomes cost-effective to replace the program with a re-created version."

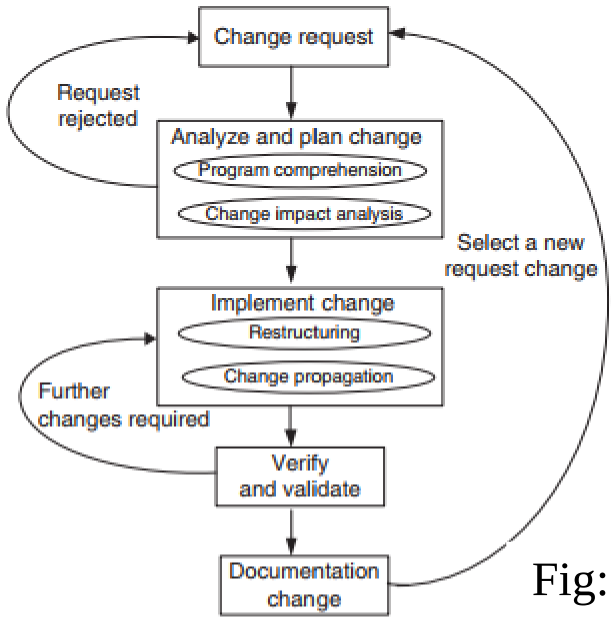


Fig: The change min-cycle

Change Mini-cycle Model …

- The change mini-cycle model consists of five major phases: CR, analyze and plan change, implement change, verify and validate, and documentation change.

I. **Change request:** A CR generally originates from the management, users of the software, or customers. A CR may take one of the following two forms: defect report and enhancement request.

II. **Analyze and plan change:** In the second phase, program comprehension and impact analysis are conducted.

- ***Program comprehension*** is essential to understanding which parts of the software will be affected by a CR.
- Program comprehension is basically a process of acquiring useful information from source code.
- ***Impact analysis*** is conducted to identify the potential consequences of a change and estimate the resources needed to accomplish the change

Change Mini-cycle Model …

III. **Implement change:** The CR is implemented after the feasibility of a change is established.

- before the implementation of the CR, restructuring or refactoring of the software is performed in order to accommodate the requested modification.

IV. **Verify and validate:** In this phase the software system is verified and validated in order to assure that the integrity of the system has not been compromised.

- This activity includes code review, regression testing, and execution of new tests if necessary.
- Regression testing comprises a subset of the unit-, integration-, and system-level tests

V. **Documentation change:** The final phase of the change mini-cycle deals with updating the program documentation.

- It is time to complete the documentation aspect which may include updating the requirements, functional specifications, and design specifications to be consistent with the code.

Software Maintenance Standards

- ♦ ***Software Maintenance Standards:*** A well-defined process for software maintenance can be observed and measured, and thus improved.
- ♦ IEEE and ISO have both addressed s/w maintenance processes.
- ♦ IEEE/EIA 1219 and ISO/IEC 14764 as a part of ISO/IEC12207 (life cycle process).
- ♦ IEEE/EIA 1219 organizes the maintenance process in seven phases:

- ***Problem identification***, ***Analysis***, ***Design***, ***Implementation***, ***System test***, ***Acceptance test*** *and* ***Delivery***.
- for each phase, the standard identifies the ***input*** and ***output deliverables***, the ***supporting processes*** and the ***related activities***, and ***a set of evaluation metrics***.

Software Configuration Management

- ***Software Configuration Management:*** Configuration management (CM) is the discipline of managing and controlling changes in the evolution of software systems.
    - The goal of CM is to manage and control the various extensions, adaptations, and corrections that are applied to a system over its lifetime.
- An SCM system has four different elements, each element addressing a distinct user need as follows:
    - **Identification of software configurations**: This includes the definitions of the different artifacts, their baselines or milestones, and the changes to the artifacts.
    - **Control of software configurations:** controlling the ways artifacts or configurations are altered with the necessary technical and administrative support.
    - **Auditing software configurations:** Making the current status of the software system in its life cycle visible to management and determining whether or not the baselines meet their requirements.
    - **Accounting software configuration status:** providing an administrative history of how the software system has been altered, by recording the activities

Reengineering

- ***Reengineering*** implies a single cycle of taking an existing system and generating from it a new system, Whereas evolution can go forever.
- To a large extent, software evolution can be seen as repeated software reengineering.
- Reengineering includes some kind of reverse engineering activities to design an abstract view of a given system.
  - The new abstract view is restructured, and forward engineering activities are performed to implement the system in its new form.
- The aforementioned process is captured by Jacobson and Lindst¨orm with the following expression:
- ***Reengineering*** = ***Reverse engineering*** + Δ + ***Forward engineering***.

Reengineering...

- The first element "***reverse engineering***" is the activity of defining a more abstract and easier to understand representation of the system.
  - For example, the input to the reverse engineering process is the source code of the system, and the output is the system architecture.
  - The core of reverse engineering is the process of examination of the system, and it is not a process of change.
- The third element "***forward engineering***" is the traditional process of moving from a high-level abstraction and logical, implementation-independent design to the physical implementation of the system.
- The second element "Δ" captures alterations performed to the original system.

Legacy Systems

- ♦ A legacy software system is an old program that continues to be used because it still meets the users' needs, in spite of the availability of newer technology or more efficient methods of performing the task.
- ♦ It is the phase out stage of the software evolution model of Rajlich and Bennet described earlier.
- ♦ There are a number of options available to manage legacy systems. Typical solution include:
- ♦ ***Freeze***: The organization decides no further work on the legacy system should be performed.
- ♦ ***Outsource***: An organization may decide that supporting software is not core business, and may outsource it to a specialist organization offering this service.

Legacy Systems…

- ***Carry on maintenance:*** Despite all the problems of support, the organization decides to carry on maintenance for another period.
- ***Discard and redevelop:*** Throw all the software away and redevelop the application once again from scratch.
- ***Wrap:*** It is a black-box modernization technique – surrounds the legacy system with a software layer that hides the unwanted complexity of the existing data, individual programs, application systems, and interfaces with the new interfaces.
- ***Migrate:*** Legacy system migration basically moves an existing, operational system to a new platform, retaining the legacy system's functionality and causing minimal disruption to the existing operational business environment as possible.

Impact Analysis

- Impact analysis is the task of estimating the parts of the software that can be affected if a proposed change request is made.
- Impact analysis techniques can be partitioned into two classes:
  - Traceability analysis In this approach the high-level artifacts such as requirements, design, code and test cases related to the feature to be changed are identified.
  - A model of inter-artifacts such that each artifact in one level links to other artifacts is constructed, which helps to locate a pieces of design, code and test cases that need to be maintained.
  - Dependency (or source-code) analysis Dependency analysis attempt to assess the affects of change on semantic dependencies between program entities.
  - This is achieved by identifying the syntactic dependencies that may signal the presence of such semantic dependencies.
    - The two dependency-based impact analysis techniques are: call graph based analysis and dependency graph based analysis.

Refactoring

- ***Refactoring*** is the process of making a change to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- It is the object-oriented equivalent of ***restructuring***.
- ***Refactoring***, which aims to improve the internal structure of the code, achieve through the removal of duplicate code, simplification, making code easier to understand, help to find defects and adding flexibility to program faster.
- There are two aspects of the above definition:
  - It must preserve the “observable behavior” of the software system (through regression).
  - To improve the internal structure of a software system (improve maintainability).

Software Reuse

- Software reuse was introduced by Dough McIlroy in his 1968 seminal paper:
  - The development of an industry of reusable source-code software components and the industrialization of the production of application software from off-the-shelf components.
- Software reuse is using existing artifacts or software knowledge during the construction of a new software system.
  - Reusable assets can be either reusable artifacts or software knowledge.
- Capers Jones identified four types of reusable artifacts:
  - data reuse, involving a standardization of data formats, Reusable functions imply a standard data interchange format.
  - architectural reuse, which consists of standardizing a set of design and programming conventions dealing with the logical organization of software,
  - design reuse, for some common business applications, and
  - program reuse, which deals with reusing executable code.

Software Reuse

Benefits of Reuse

- ♦ Increased reliability
- ♦ Reduced process risk
- ♦ Increase productivity
- ♦ Standards compliance
- ♦ Accelerated development
- ♦ Improve maintainability
- ♦ Reduction in maintenance time and effort

References

- 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