



SOEN 6431

Software Comprehension and Maintenance

Week 2 – Software
Maintenance and program
comprehension

Software Maintenance: Definition and objectives

- ISO/IEC and IEEE define maintenance as the modification of a software product after delivery to correct faults, improve performance (or other attributes) or to adapt the product to a modified environment (ISO/IEC 14764:2006(E); IEEE Std 14764-2006).
- The objective of maintenance is not to stop the unavoidable effects of aging, but to provide techniques and tools to understand its causes, to limit its effects and to prolong the life of software systems.

Maintenance Challenge

- Providing continuity of service.
- Supporting mandatory upgrades.
- Supporting user requests for improvements.
- Facilitating future maintenance work.

Importance of maintenance

- The importance of maintenance lies on the following observations:
 - 1) Surveys indicate that it is an activity which tends to consume a significant proportion of the resources utilized in the overall life cycle (consequently consuming a large part of the costs)
 - 2) Reliable changes to software tend to be time consuming. Prolonged delays during software change may result in a loss of business opportunities.

Maintenance Framework

- Software maintenance is not done in vacuum.
- It affects and interacts with the environment.
- ***Environment*** – the totality of conditions and influences which act from outside upon an entity
- This interaction is at the origin of SW change.
- It is important to understand the maintenance framework:
 - Components,
 - Relationship between these components.

Components of the Framework

- **Framework** – a set of ideas, conditions, or assumptions that determine how something will be approached, perceived, or understood.
- Components of the Framework
 - User
 - Environment (operating and organizational)
 - Maintenance process
 - Software product
 - Maintenance personnel

Components of the Framework

- User
- Environment (operating and organizational)
- Maintenance process
- Software product
- Maintenance personnel

User

- Users often have little understanding of software maintenance.
- They may take the view that:
 - Software maintenance is like hardware maintenance.
 - Changing software is easy.
 - Changes cost too much and take too long.
- Users may be unaware that their request :
 - May involve major structural changes to the software.
 - Must be feasible, desirable, prioritized, scheduled and resourced.

User (cont.)

- The implementation of such modifications/changes may necessitate:
 - ***progressive*** work to refine existing functions or to introduce new features
 - ***anti-regressive*** work to make program well structured, better documented, more understandable and capable of further development

Components of the Framework

- User
- Environment (operating and organizational)
- Maintenance process
- Software product
- Maintenance personnel

Environment

- Environments affecting software systems are:
 - Operating environment
 - Organizational environment.
- ***Operating environment*** – all software and hardware systems that influence or act upon a software product.
- ***Organizational environment*** – all non-software or non-hardware-related environmental factors.

Operating Environment

- ***Hardware innovations:*** The hardware platform on which a software system runs may be subject to change during the lifetime of the software.
- ***Software innovations:*** Changes in the host software may warrant a corresponding modification in the software product.
 - Operating systems
 - Database management systems
 - Compilers
 - Etc.

Organizational Environment

- Changes in the policies
 - Change in the business rules
 - Changes in taxation policies
- Competition in the market place
 - May result in substantial modifications so as to maintain the status of the product
 - Despite no direct financial input from the user, resources (both machine and human) still need to be allocated

Case study – Maintainability and VAT Rules

- The original system sets a fixed Valued Added Tax (VAT) rate on a specific list of goods.
- Continuous changes on VAT
 - Variable VAT rates depending on goods
 - Different rates of the same goods depends on how they are sold
- The mapping between the VAT rate and the list of goods has been “hard-wired” into the code in the early system

Methods of Maintenance

- Basic modification – declaration of constants and the use of internal tables
 - Still mean rewriting programs
 - Allow the misuse of values within the program
- Further modification – modular code
 - Encapsulation of parts of the code
 - Execution of a particular part of a program cannot accidentally modify irrelevant variables
 - Proper separation of data from code avoid unnecessary code modification
 - Store data in external tables or files

Methods of Maintenance (cont.)

- Even further modification – true interoperability between software systems using properly researched and formulated interfaces
 - Separation of data will not prevent the need to modify programs, since we cannot predict all new factors appearing in the future
 - E.g., VAT rates depend on a specific selling context such that the same goods sold in different contexts attract different rates
 - Programs should not rely on their own internal calculations for things over which they have no control
 - Access central data sources (e.g., a central VAT server)

Components of the Framework

- User
- Environment (operating and organizational)
- Maintenance process
- Software product
- Maintenance personnel

Maintenance Process

- ***Maintenance Process*** – any activity carried out or action taken either by a machine or maintenance personnel during software maintenance.
- Typical factors that are:
 - Capturing change requirements.
 - Variation in programming practice
 - Paradigm shift
 - Error detection and correction.

Maintenance Process (cont.)

- **Capturing change requirements**
 - It is the process of finding out exactly what changes are required
 - Fundamentally difficult to capture all requirements *a priori*
 - **Information gap** – the inconsistency between the body of knowledge that system users and system maintainers possess and the body of knowledge that each needs to have in order to satisfy a request for change
- **Variation in programming practice**
 - Refers to differences in approaches used for writing and maintaining programs.
 - Involves the use of features/operations imposing a particular program structure.
 - E.g., avoiding the use of 'GOTOs', use of meaningful identifier names, logical program layout, document design and implementation rationale through program comments.
 - These features have impacts on program comprehension.

Maintenance Process (cont.)

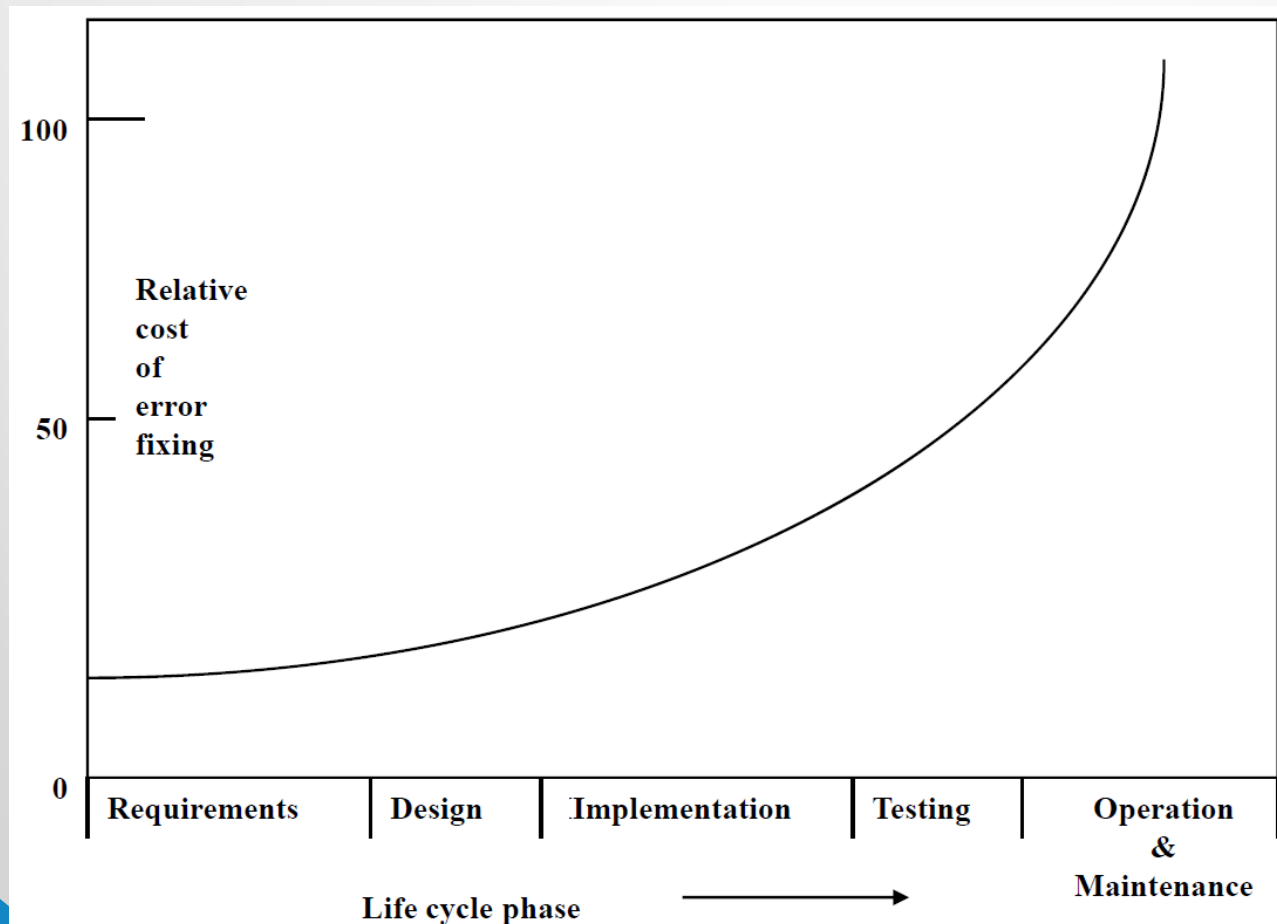
- **Paradigm shift**

- Refers to an alternation from one computational paradigm to another.
- From non-structured to structured.
- From procedural to object-oriented.
- Etc.
- **Note:** even if the programs are developed using state of the arts programming methods, they may gradually lose their structure after a series of unplanned and ad hoc 'quick fixes'.

Maintenance Process (cont.)

- **Error detection and correction**

- The latter is a problem identified, the costly is the fix




Components of the Framework

- User
- Environment (operating and organizational)
- Maintenance process
- Software product
- Maintenance personnel

Software Product

- Aspects of a software product that contribute to the maintenance challenge include:
 - Maturity and difficulty of the application domain
 - Inherent difficulty of the original problem
 - Quality of the documentation
 - Programs are often modified without corresponding update of the documents affected, if any.
 - Malleability of the programs
 - Inherent quality
 - A program/system has a tendency to decay as more changes are undertaken
 - Preventive maintenance needs to be undertaken to restore order in the program.



"A program that is used and that as an implementation of its specification reflects some other reality, undergoes continual changes or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the system with a recreated version."

Lehman, *Program Evolution*, 1985

Components of the Framework

- User
- Environment (operating and organizational)
- Maintenance process
- Software product
- Maintenance personnel

The Maintenance Personnel

- This refers to individuals involved in the maintenance of a software product
 - Including maintenance managers, analysts, designers, programmers, testers, etc.
- The aspects of personnel that affect maintenance activities include the following:
 - Staff turnover
 - Domain expertise
 - Working practices
- ***Staff turnover***
 - 60% of the total time was taken to identify the lines of code that need to be changed [Ede93]
 - No history of successful/unsuccessful change attempts tried [Ede93]

The Maintenance Personnel (cont.)

- ***Domain expertise***
 - Staff may end up working on a system for which they have neither the system domain knowledge nor the application domain knowledge.
 - Introduce changes without being aware of their effects on the system
- ***Working practices***
 - Factors that can make the maintenance more difficult
 - Desire to be clever
 - Undocumented assumptions
 - Undocumented design and implementation decisions, etc.

Software Maintenance Framework

Component	Feature
1. User requirements	<ul style="list-style-type: none">• Requests for additional functionality, error correction and improve maintainability• Request for non-programming related support
2. Organisational environment	<ul style="list-style-type: none">• Change in policies• Competition in the market place
3. Operational environment	<ul style="list-style-type: none">• Hardware innovations• Software innovations
4. Maintenance process	<ul style="list-style-type: none">• Capturing requirements• Creativity and undocumented assumptions.• Variation in programming practices.• Paradigm shift• Error detection and correction
5. Software product	<ul style="list-style-type: none">• Maturity and difficulty of application domain• Quality of documentation• Malleability of programs• Complexity of programs• Program structure• Inherent quality
6. Maintenance personnel	<ul style="list-style-type: none">• Staff turnover• Domain expertise

Relations between Maintenance Factors

- Three major types of relation:

1) *Product / environment:*

- As the hosting operating/organizational environment changes, the product must change in order to remain useful.

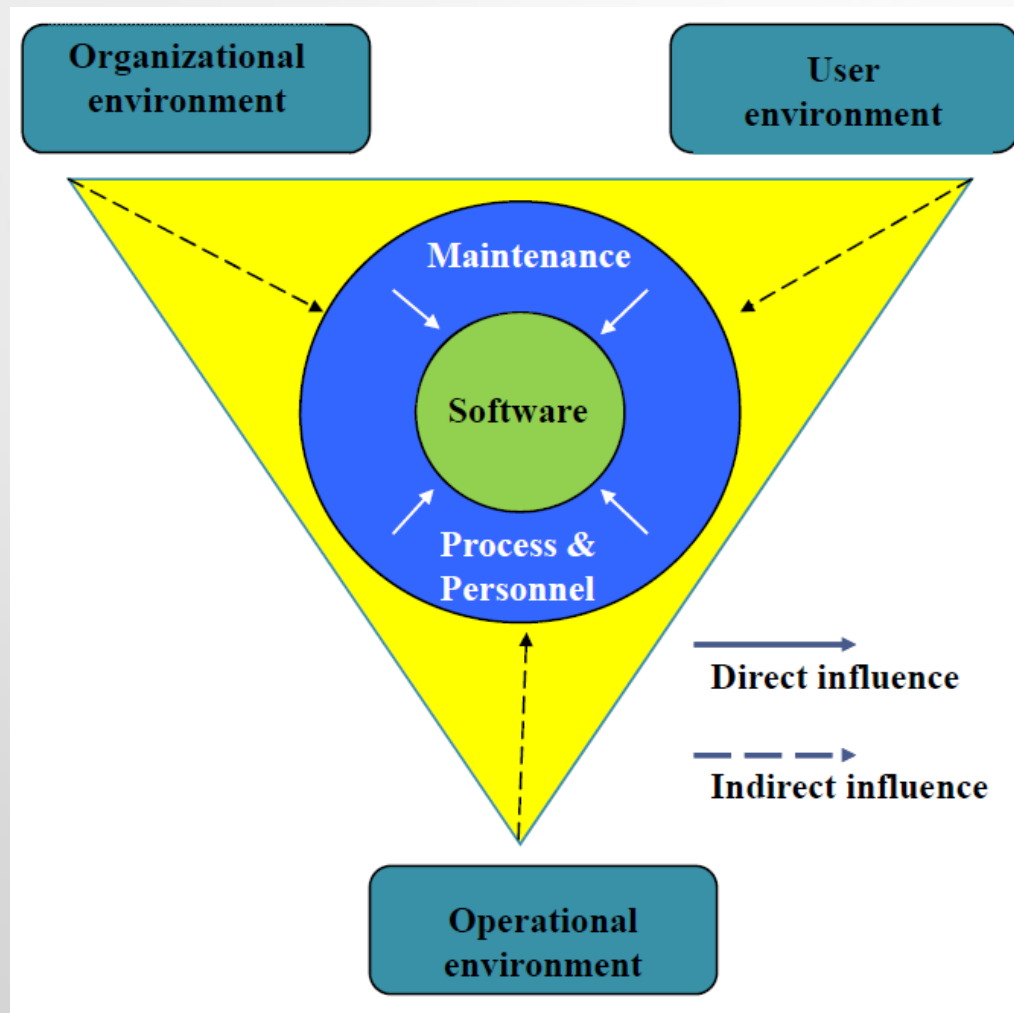
2) *Product / user:*

- To stay acceptable to users, the product has to change to accommodate new user requirements.

3) *Product / maintenance personnel.*

- The personnel should act as receptors to the change requests.

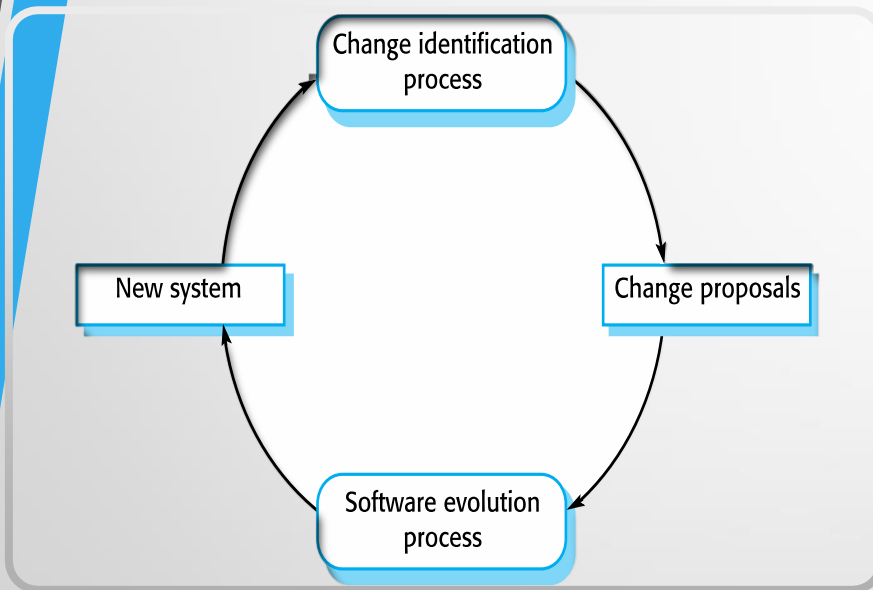
Relations between Maintenance Factors





Evolution Life Cycle

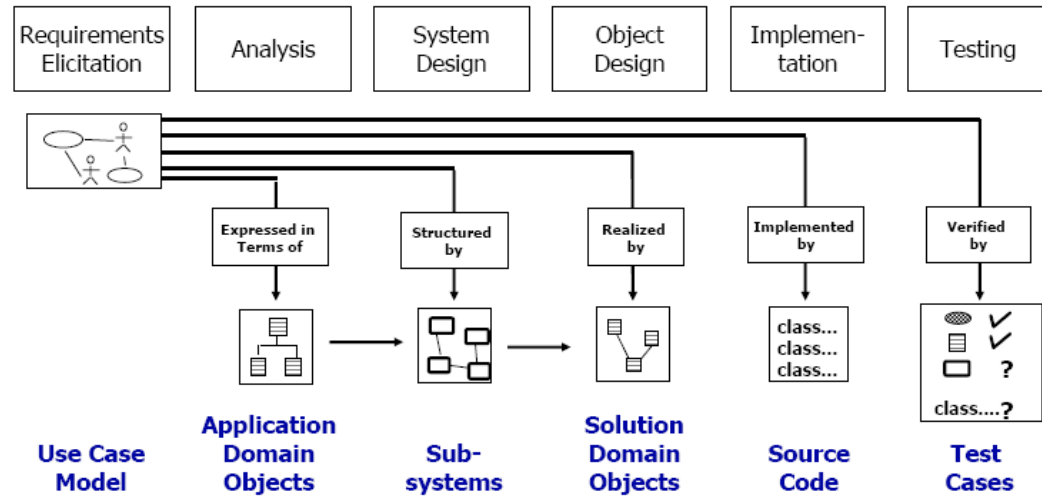
Change identification and evolution processes



- Proposals for change are the driver for system evolution.
 - Should be linked with components that are affected by the change, thus allowing the cost and impact of the change to be estimated.
- Change identification and evolution continues throughout the system lifetime.

General Idea

- Software Development we do have different Software Life Cycle Models, e.g.,
 - Waterfall
 - Spiral Model
 - Prototyping
 - Iterative
 - Open source
- These models describe the sequence of activities (e.g., requirements analysis, implementation, testing) to be performed during the development
- What about software maintenance?



Software Life Cycle and its models.

Software Evolution/ Maintenance Models: General Idea



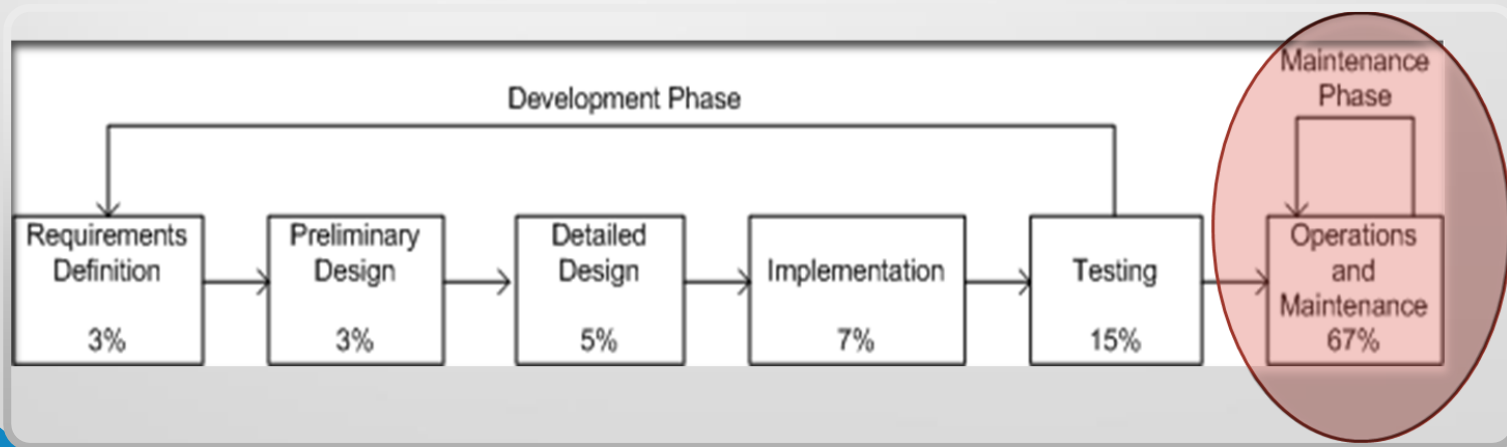
One traditional software development life cycle (SDLC) is shown below, which comprises two discrete phases, namely: development and maintenance



Maintenance approaching two-thirds of the product life-span.



The percentages indicate relative costs.



Software maintenance has unique characteristics:



Constraints of an existing system: Maintenance is performed on an operational system. Therefore, all modifications must be compatible with the constraints of the existing architecture, design, and code.



Shorter time frame: A maintenance activity may span from a few hours to a few months, whereas software development may span one or more years.



Available test data: In software development, test cases are designed from scratch, whereas software maintenance can select a subset of these test cases and execute them as regression tests.

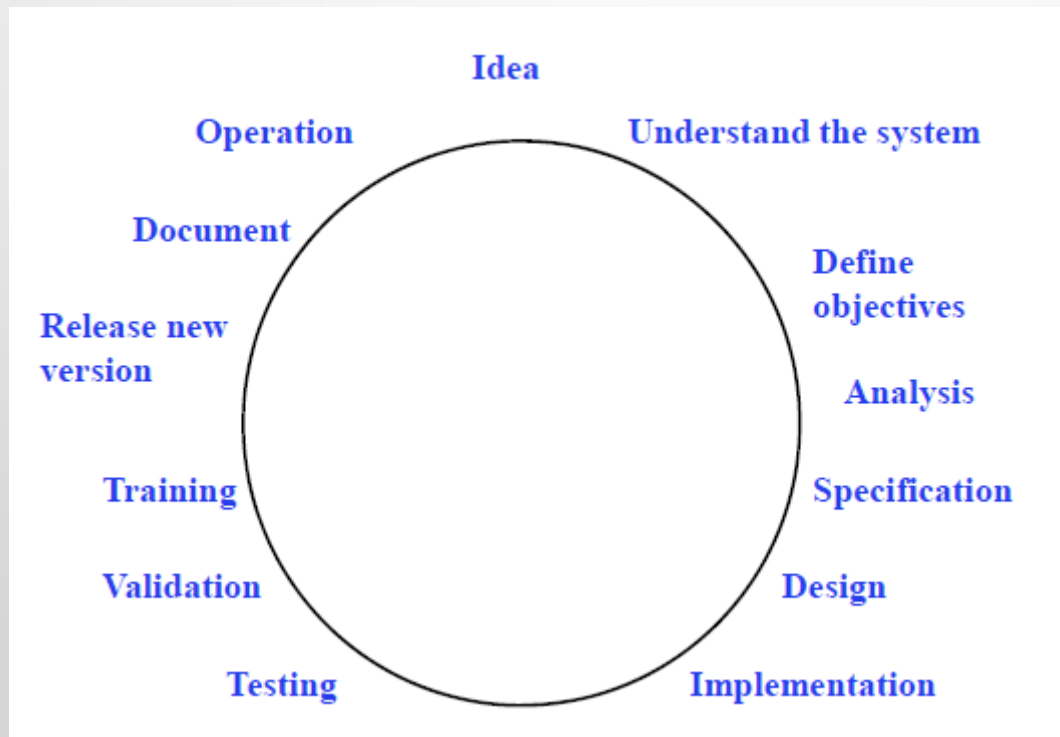


Software maintenance should have its own Software Maintenance Life Cycle (SMLC) model as it involves many unique activities.

**General
Idea**

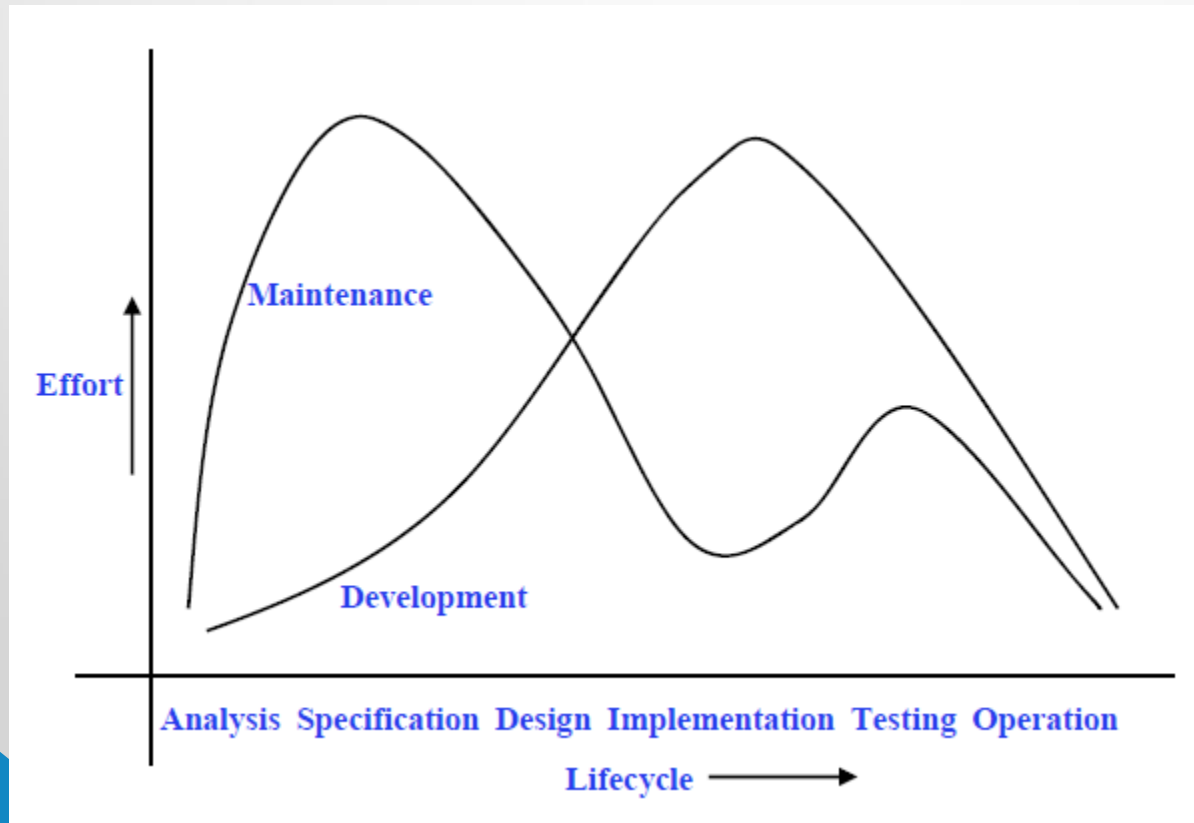
Maintenance Process Models

- Adding a new requirement to an existing software is totally different from adding it from the start.
 - Example of adding a 3rd room inside a house
- Stages in a maintenance-conscious model.



Maintenance Process Models

- The generic stages appear similar to the traditional development model stages on the surface.
- But within stages, there are great differences
 - Effort repartition along the life-cycle is different



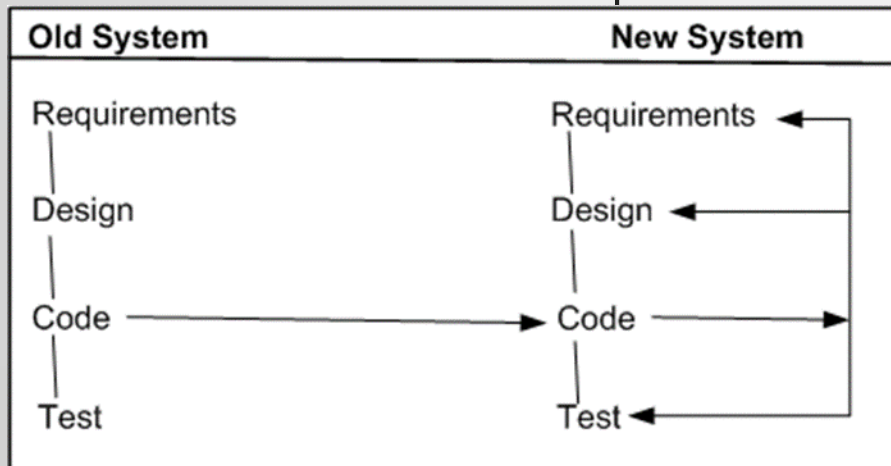
Reuse Oriented Models

- One obtains a new version of an old system by modifying one or several components of the old system and possibly adding new 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.
 - **Iterative enhancement model:** In this model, first changes are made to the highest level documents. Eventually, changes are propagated down to the code level.
 - **Full reuse model:** In this model, a new system is built from components of the old system and others available in the repository.

Reuse Oriented Models

In Quick Fix Model

- source code is modified to fix the problem;
- necessary changes are made to the relevant documents; and
- the new code is recompiled to produce a new version.
- Often changes to the source code are made with no prior investigation such as analysis of impact of the changes, ripple effects of the changes, and regression testing.





Iterative Vs. Incremental



The terms **iteration** and **increment** are liberally used when discussing iterative and incremental development.



However, they are not synonyms in the field of software engineering.



Iteration implies that a process is basically cyclic, thereby meaning that the activities of the process are repeatedly executed in a structured manner.



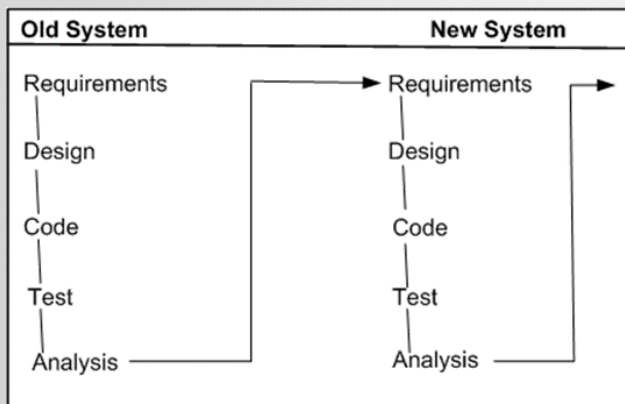
Iterative development is based on scheduling strategies in which time is set aside to improve and revise parts of the system under development.



Incremental development is based on staging and scheduling strategies in which parts of the system are developed at different times and/or paces, and integrated as they are completed.

Reuse Oriented Models

Reuse Oriented Models

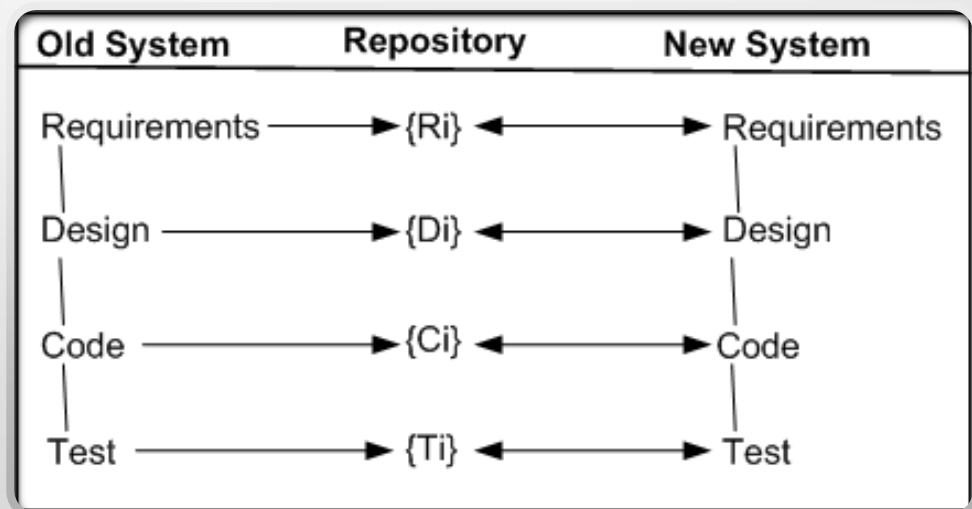


The **iterative enhancement model** shows how changes flow from the very top level documents to the lowest-level documents.

- The model works as follows:
- It begins with the existing system's artifacts, namely, requirements, design, code, test, and analysis documents.
- It revises the highest-level documents affected by the changes and propagates the changes down through the lower-level documents.
- The model allows maintainers to redesign the system, based on the analysis of the existing system.

Reuse Oriented Models

- The main assumption in **full reuse model** is the availability of a repository of artifacts describing the earlier versions of the systems.
- In the **full reuse model**, reuse is explicit, and the following activities are performed:
 - identify the components of the old system that are candidates for reuse
 - understand the identified system components.
 - modify the old system components to support the new requirements.
 - integrate the modified components to form the newly developed system.

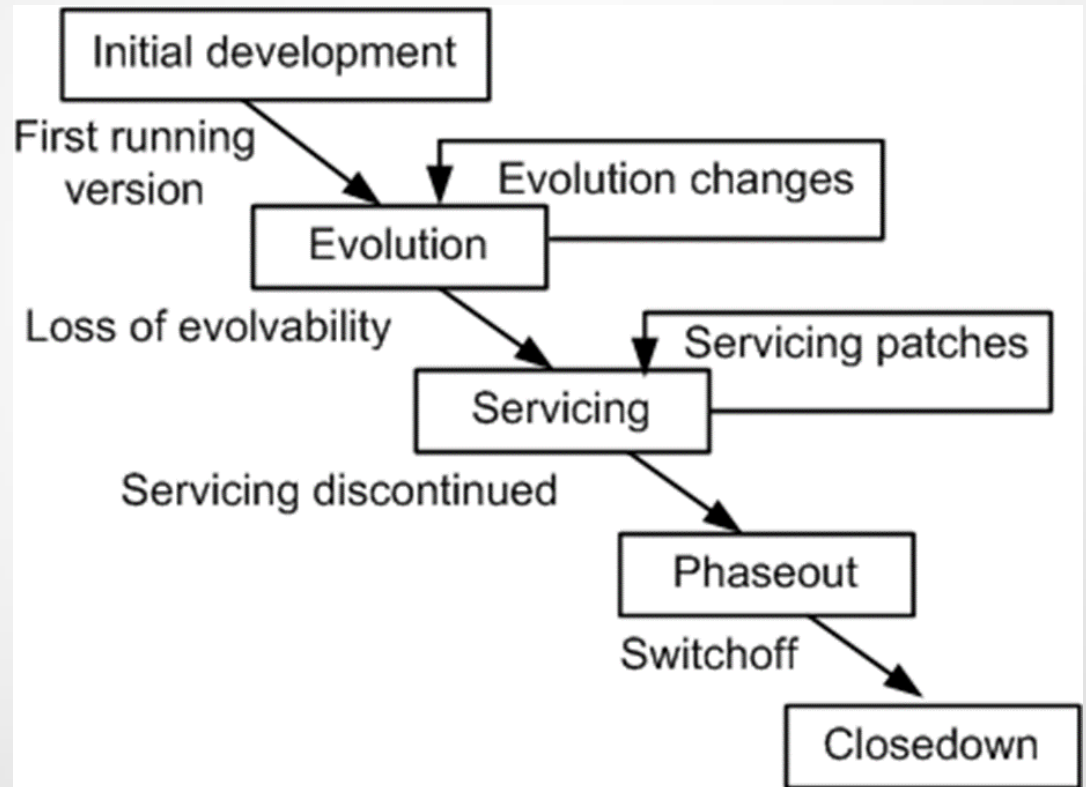


The full reuse model ©IEEE, 1990

The Staged Model for Software Systems

- Rajlich and Bennett have presented a descriptive model of software evolution called the staged model of maintenance and evolution.
- Its primary objective is at improving understanding of how long-lived software evolves, rather than aiding in its management.
- Their model divides the lifespan of a typical system into four stages:
 - Initial development,
 - Evolution
 - Servicing
 - Phaseout and closedown
- The main contribution is to separate the “maintenance” phase into an *evolution* stage followed by a *servicing* and *phase out* stages
 - Two staged models are proposed
 - Simple staged model
 - Versioned staged model

The Staged Model for Software Systems



original
version
from 1990

Initial Development

- Initial development would produce a deployable system (the first operating version).
- Two important outcomes of the initial development stage include:
 - **Software team expertise**
 - Knowledge of the application domain
 - Solutions and algorithms
 - Data formats
 - **System architecture**
 - User requirements
 - Role of the application in business process
 - Strengths and weaknesses of the program architecture
 - Operating environments

Evolution

- Simple changes are easily performed and more major changes are possible too, albeit the cost and risk are now greater than in the previous stage.
- Knowledge about the system is still good, although many of the original developers will have moved on.
- For many systems, most of its lifespan is spent in this phase.

Servicing

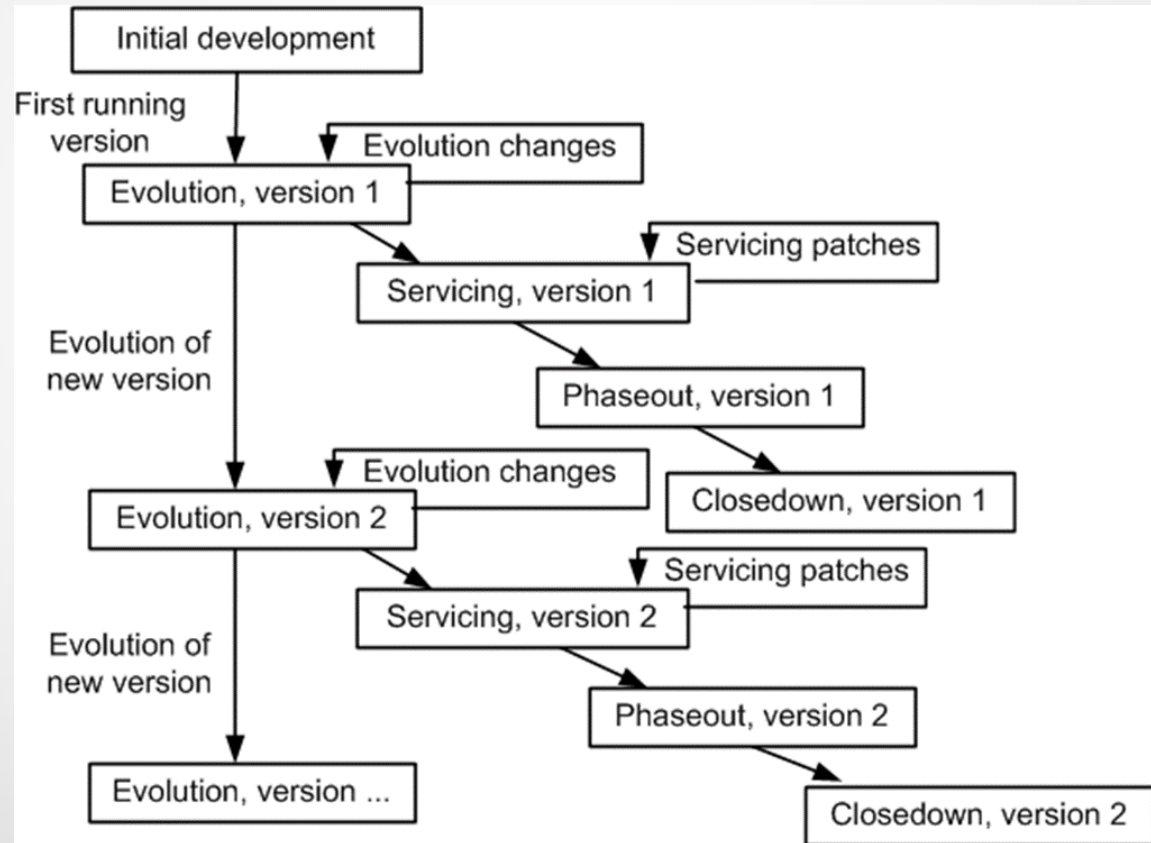
- To evolve easily, software must have both an appropriate architecture and a skilled development team.
- When these are lacking, evolution is no longer viable and the software enters the servicing or saturation stage when it is considered aging, decayed, or legacy.
- Only small changes are possible during this stage.
- Changes are both difficult and expensive, so developers minimize them or do them as wrappers, which are simply modifications to inputs and outputs, leaving the old software untouched.
- Still, each change further degrades the architecture, pushing it deeper into servicing. ⁴⁸

Phase-out and Close-down

- During the phase-out stage
 - No more servicing is being taken, but the system may still be in production
 - Users must work around known deficiencies
 - It is difficult to return to the previous servicing stage because of the growing backlog of change requests
- During the close-down stage
 - Software use is disconnected
 - Users are directed towards a replacement
 - The company may have residual responsibilities, e.g., source code retention and legal liability
 - As a system moves from phase-out to close-down, managers should be very carefully plan and organize migration of data to a new system.

The Staged Model for Software Systems

Updated
version
from 2000



Versioned Staged Model (cont.)

- The backbone of the versioned staged model is the evolution stage
- At certain intervals, a company completes a version of its software and release it to customers
- Evolution continues, with the company eventually releasing another version and only servicing the previous version

Versioned Staged Model (cont.)

- Finally, once servicing is no longer viable the system enters a phaseout stage where deficiencies are known but not addressed.
- At closedown, the system is withdrawn from the market.
 - In an alternative model (versioned staged model), during evolution a version is publicly released and subsequently enters the servicing stage whereas the system continues to evolve in order to produce the next version.

Staff Expertise and Software Architecture

- Staff expertise
 - Critical in the initial development and evolution stages
 - Less important during servicing, where the expertise is limited to inputs and outputs
 - Even less important during phase-out, where the expertise is limited to how to execute the program
- Software architecture
 - During evolution
 - As changes accumulate, the architecture loses its original lucidity and integrity
 - Sometimes require restructuring so as to partially rebuild the architecture to facilitate future evolution
 - During servicing
 - The architecture may be damaged by wrapping.
 - Minimize the impact on damaging the architecture.

Software Decay

- There is a positive feedback between the loss of software architecture coherence and the loss of the software knowledge
 - In most cases, the loss of knowledge is triggered by loss of key personnel
- Only experts can fully understand
 - When a change is tactical
 - When a change will have profound effects on the architecture
 - When a change will cause serious problems
- This expertise is nearly impossible to document.
- Re-engineering is an attempt to reverse decay, but it is slow and expensive, with many risks
- Current solution is wrapping

References

- **[Ede93]** D. Vera Edelstein, "*Report on the IEEE STD 1219-1993 – Standard for Software Maintenance*", ACM SIGSOFT Software Engineering Notes, October, 1993.
- **[Leh96]** M. M. Lehman, "*Laws of Software Evolution Revisited*", Proceedings of the 5th European Workshop on Software Process Technology, 1996
- **[Jones07]** C. Jones, Geriatric Issues of Aging Software, CrossTalk – The Journal of Defense Software Engineering, 20(12), December 2007, pp. 4 - 8.
- International Standards Organization/ International Electrotechnical Commission; Institute of Electrical and Electronics Engineers (2006). ISO/IEC 14764:2006(E); IEEE Std 14764-2006: International Standard: Software Engineering Software Life Cycle Processes Maintenance (2nd ed.). Geneva, Switzerland and Piscataway, NJ, USA: ISO/IEC; IEEE.

References

- **[Boe88]** Barry W. Boehm, *A Spiral Model of Software Development and Enhancements*, IEEE Computers, Vol. 21, Issue 5, 1988
- **[Par05]** Nilesh Parekh, *Spiral Model - A New Approach Towards Software Development*, 2005
- **[Bas90]** B. R. Basili, *Viewing Maintenance as Reuse-Oriented Software Development*, IEEE Software, January 1990, pp 19-25.
- **[Harg90]** Warren Harrison, *Insights on Improving the Maintenance Process Through Software Measurement*, ICSM, 1990
- **[RBoo]** Vaclav T. Rajlich and Keith H. Bennett, *A Staged Model for the Software Life Cycle*, IEEE Computer, Vol. 33, No. 7, July, 2000.
- **[BRoo]** K. H. Bennett, V. T. Rajlich, *Software maintenance and evolution: A roadmap*, ICSE, 2000.