Computers II Lesson 6

6.0 Evolution of Software

Software development does not stop when a system is delivered but continues throughout the lifetime of the system.

After a system has been deployed, it inevitably has to change if it is to remain useful.

Business changes and changes to user expectations generate new requirements for the existing software.

Parts of the software may have to be modified to:

- Correct errors that are found in operation
- Adapt it for changes to its hardware and software platform,
- Improve its performance or other non-functional characteristics.

Software evolution is important because organizations have invested large amounts of money in their software and are now completely dependent on these systems.

Their systems are critical business assets and they have to invest in system change to maintain the value of these assets.

The costs of software change are a large part of the IT budget for all companies.

Most large companies spend more on maintaining existing systems than on new systems development.

Based on an informal industry poll 85–90% of organizational software costs are evolution costs. Other surveys suggest that about two-thirds of software costs are evolution costs.

Software evolution may be triggered by:

- Changing business requirements
- Reports of software defects
- Changes to other systems in a software system's environment.

Useful software systems often have a very long lifetime.

Large military or infrastructure systems, such as air traffic control systems, may have a lifetime of 30 years or more.

Business systems are often more than 10 years old.

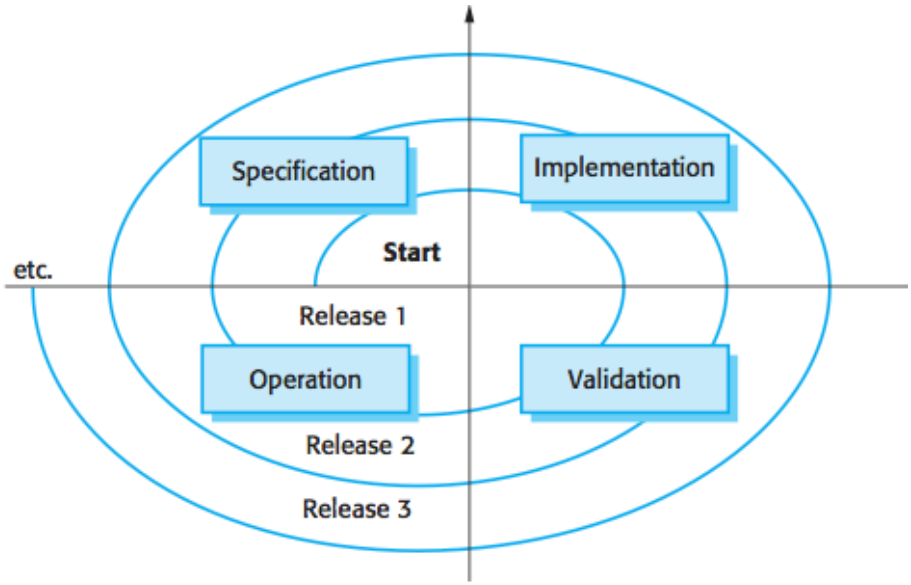
Software cost a lot of money so a company has to use a software system for many years to get a return on its investment.

The requirements of the installed systems change as the business and its environment change. Therefore, new releases of the systems, incorporating changes, and updates, are usually created at regular intervals.

You should think of software engineering as a spiral process with the below going on throughout the lifetime of the system.

- Requirements
- Design
- Implementation
- Testing

You first create release 1 of the system. Once delivered, changes are proposed and the development of release 2 starts almost immediately. The need for evolution may become obvious even before the system is deployed so that later releases of the software may be under development before the current version has been released.



- During evolution, the software is used successfully and there is a constant stream of proposed requirements changes.

- As the software is modified, its structure tends to degrade and changes become more and more expensive.

This often happens after a few years of use when other environmental changes, such as hardware and operating systems, are also often required.

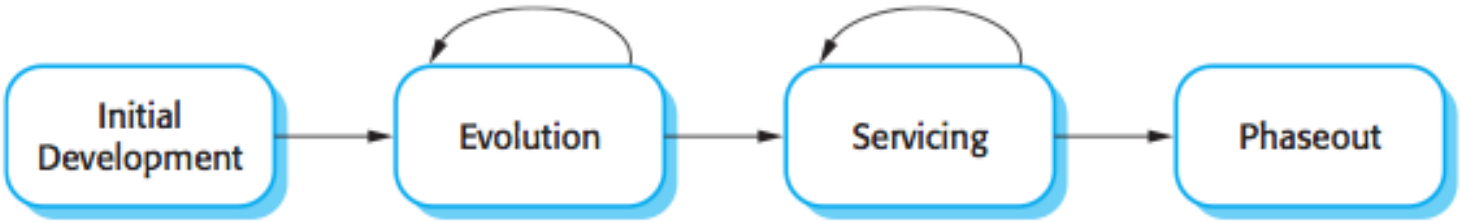
Eventually, at some stage in the life cycle, the software reaches a transition point.

- At this transition point, significant changes, implementing new requirements, become less effective and more expensive.

The difference between evolution and servicing:

- Evolution is the phase in which significant changes to the software architecture and functionality may be made.

- During servicing, the only changes that are made are relatively small, essential changes.



6.0 Lehman's Laws

A series of laws that Lehman and Belady formulated starting in 1974 with respect to software evolution.

The laws describe a balance between forces driving new developments on one hand, and forces that slow down progress on the other hand.

These laws are likely to be true for all types of large organizational software systems

Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

6.1 Software Maintenance

Software maintenance is the general process of changing a system after it has been delivered.

The term is usually applied to custom software in which separate development groups are involved before and after delivery.

Changes are implemented by modifying existing system components and, where necessary, by adding new components to the system.

Three different types of software maintenance:

1. Fault repairs - Coding errors are usually relatively cheap to correct; design errors are more expensive as they may involve rewriting several program components. Requirements errors are the most expensive to repair because of the extensive system redesign which may be necessary.

2. Environmental adaptation - This type of maintenance is required when some aspect of the system's environment such as the hardware, the platform operating system, or other support software changes. The application system must be modified to adapt it to cope with these environmental changes.

3. Functionality addition - This type of maintenance is necessary when the system requirements change in response to organizational or business change. The scale of the changes required to the software is often much greater than for the other types of maintenance.

More of the maintenance budget is spent on implementing new requirements than on fixing bugs.

Approximate distribution of maintenance costs: