SE 811: Software Maintenance

Toukir Ahammed

Evolution and Maintenance Models

Traditional SDLC Model

About one-fourth to one-third of all software life cycle costs are attributed to software development, and the remaining cost is due to operations and maintenance.

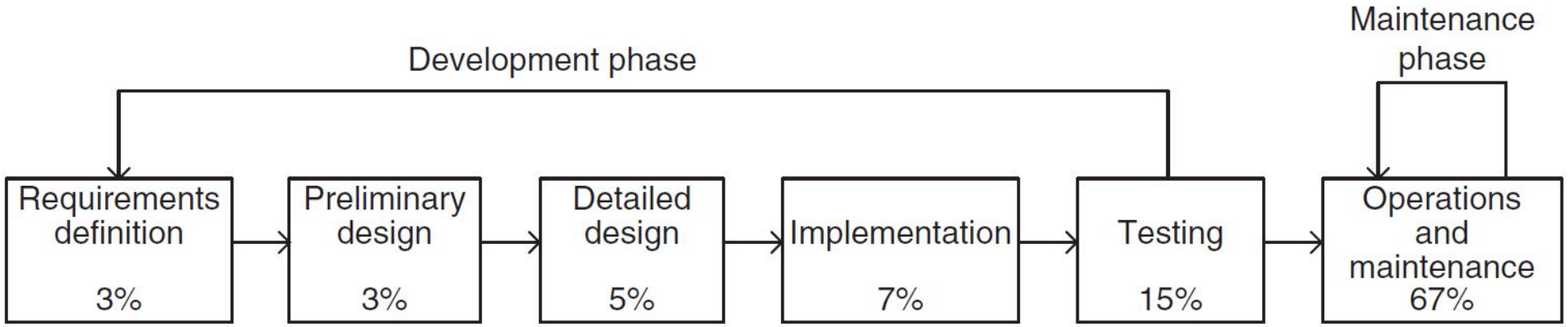

FIGURE 3.1 Traditional SDLC model. From Reference 1. © 1988 John Wiley & Sons

Characteristics of Software Maintenance

- ***Constraints of an existing system*** . Maintenance is performed on an operational system. Therefore, all modifications must be compatible with the constraints of the existing software 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 1 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. In addition, new test cases need to be created to adequately test the code changes.

Characteristics of Software Maintenance

- The software product which is released to a customer is in the form of executable code.
- Source code can be modified without affecting the executable version in use.
- Thus, strict control must be kept, otherwise exact source code representation of a particular executable version may not exist.
- In addition, documentation associated with the executable code must be compatible, otherwise the customer may not be able to understand the system.
- Therefore, tight documentation control is necessary.
- Software configuration management (SCM), Version Control System (VCS) are used to control it.

Reuse-oriented Model

Quick Fix Model

Necessary changes are quickly made to the code and then to the accompanying documentation

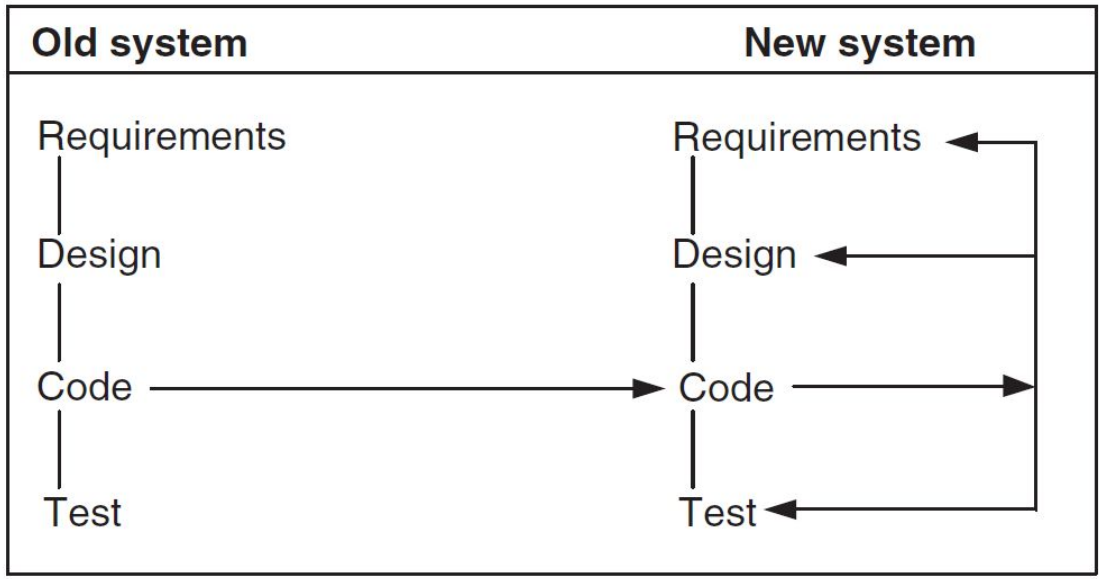

FIGURE 3.2 The quick fix model. From Reference 2. © 1990 IEEE

Quick Fix Model

The model works as follows:

- source code is modified to fix the problem;
- necessary changes are made to the relevant documents;
- 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.
- Moreover, resource constraints often entail that modifications performed to the code are not documented.

Iterative Enhancement Model

First changes are made to the highest level documents. Eventually, changes are propagated down to the code level.

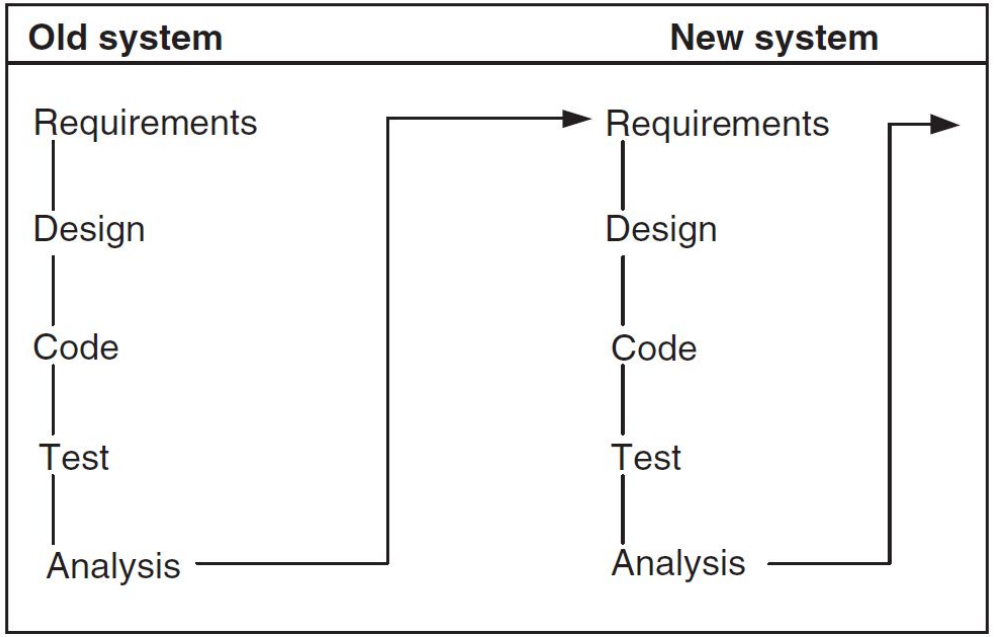

FIGURE 3.3 The iterative enhancement model. From Reference 2. © 1990 IEEE

Iterative Enhancement Model

The ideas behind the model:

- Sometimes it is difficult to fully comprehend a large set of requirements for a system and developers may find it difficult to build the full system in one go.
- Therefore, a complete system is developed in progressively larger builds

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.

Full Reuse Model

A new system is built from components of the old system and others available in the repository.

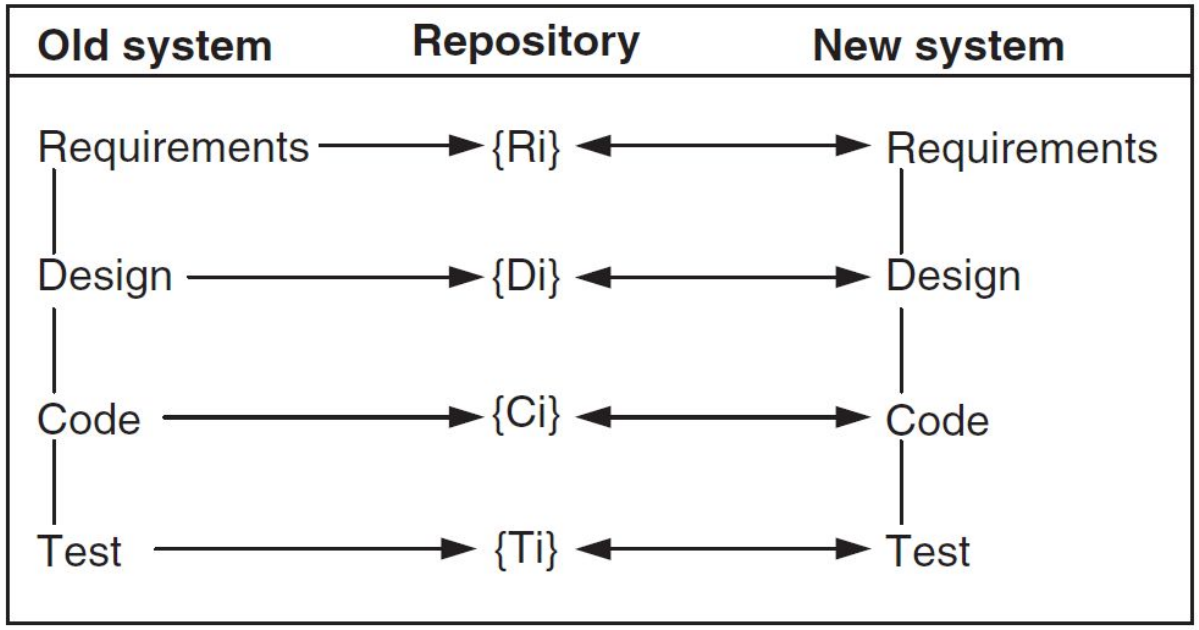

FIGURE 3.4 The full reuse model. From Reference 2. © 1990 IEEE

Full Reuse Model

Full reuse comprises two major steps:

- perform requirement analysis and design of the new system; and
- use the appropriate artifacts, such as requirements, design, code, and test from any earlier versions of the old system.

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; and
- integrate the modified components to form the newly developed system.

Staged Model

The Simple Staged Model for Closed Source Software

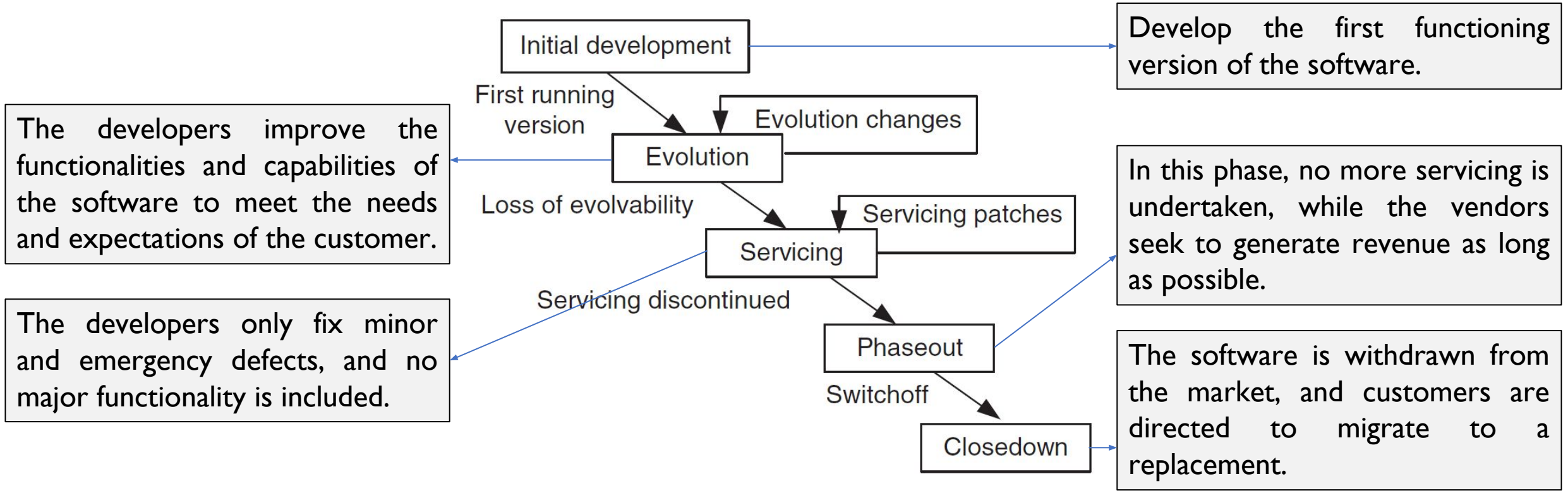

FIGURE 3.5 The simple staged model for the CSS life cycle. From Reference 6. © 2000 IEEE

The Versioned Staged Model for Closed Source Software

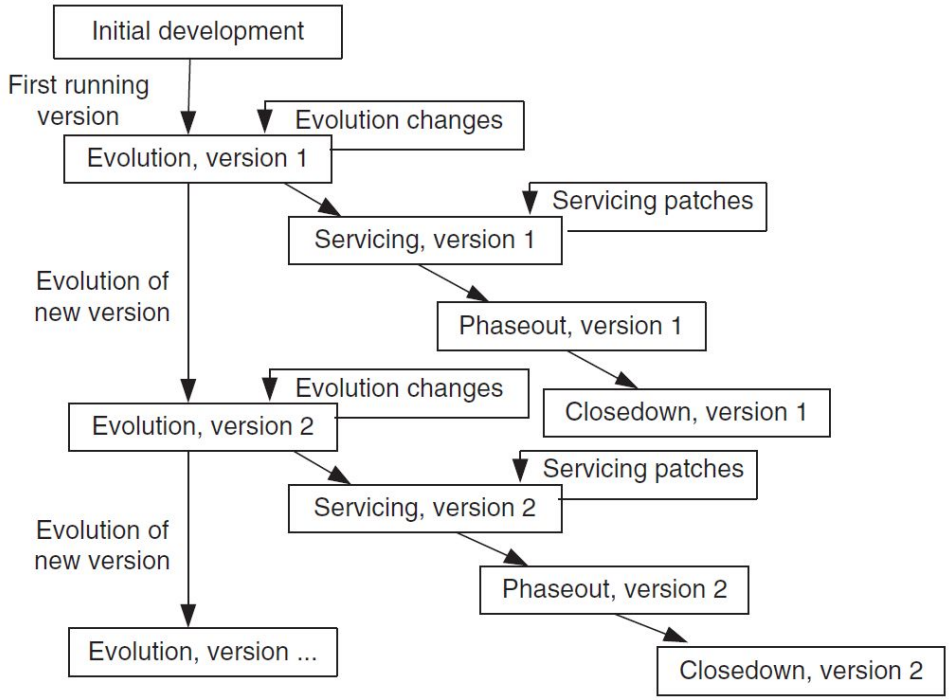

FIGURE 3.6 The versioned staged model for the CSS life cycle. From Reference 6.
© 2000 IEEE

Many organizations use a scheme such as <product><version><release><build>, where **version** reflects the strategic changes made to the system during evolution, **release** reflects the servicing patches, and **build** reflects the, say, daily internal build of the software.

The Staged Model for Free, Libre, Open Source Software

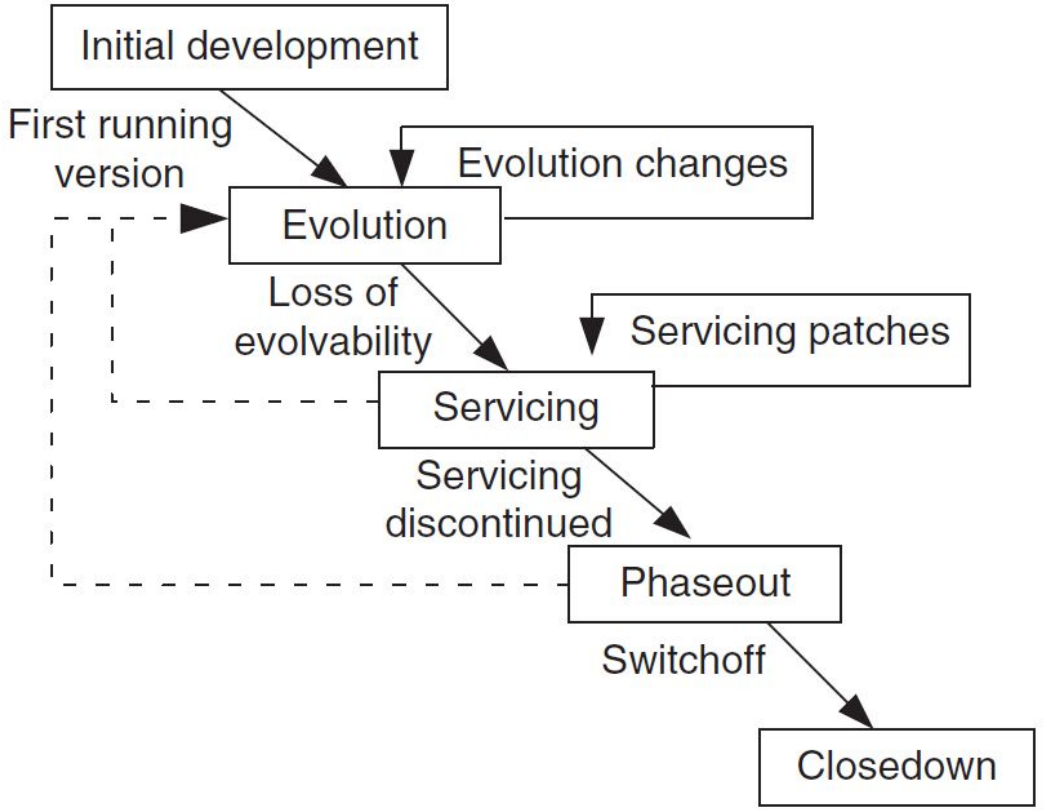

FIGURE 3.7 The staged model for the FLOSS system. From Reference 9. © 2007 ACM