Evolution and Maintenance Models

Idaho State University | Computer Science

Isaac Griffith

CS 4423 and CS 5523
Department of Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will:

- Know about the general idea of a model for software maintenance and evolution
- Be aware of the Reuse Oriented Model
- Be aware of the Staged Models for CSS and FLOSS
- Be aware of the Change Mini-Cycle Model

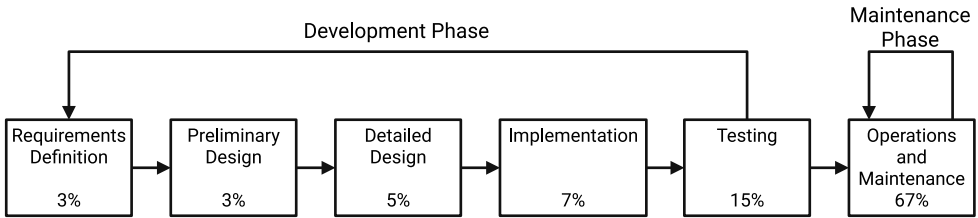
SME Models

CS 4423/5523

ROAR

General Idea

- One traditional software development life cycle (SDLC) is shown in the figure below, which comprises two discrete phases, namely:
  - **Development**
  - **Maintenance**
- Maintenance approaching two-thirds of the product life-span.
- The percentages in the figure indicate relative costs



General Idea

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.

Reuse Oriented Models

- One obtains a new version of an old system by modifying one of 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.

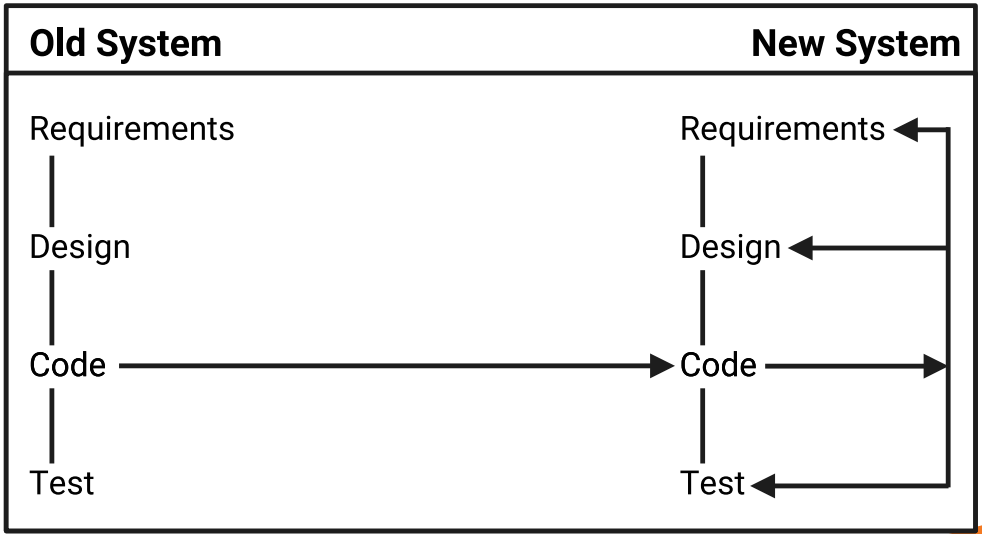
Reuse Oriented Models

In Quick Fix Model, illustrated below:

1. Source code is modified to fix the problem
2. Necessary changes are made to the relevant documents
3. 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

Quick-Fix Model



Reuse Oriented Models

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

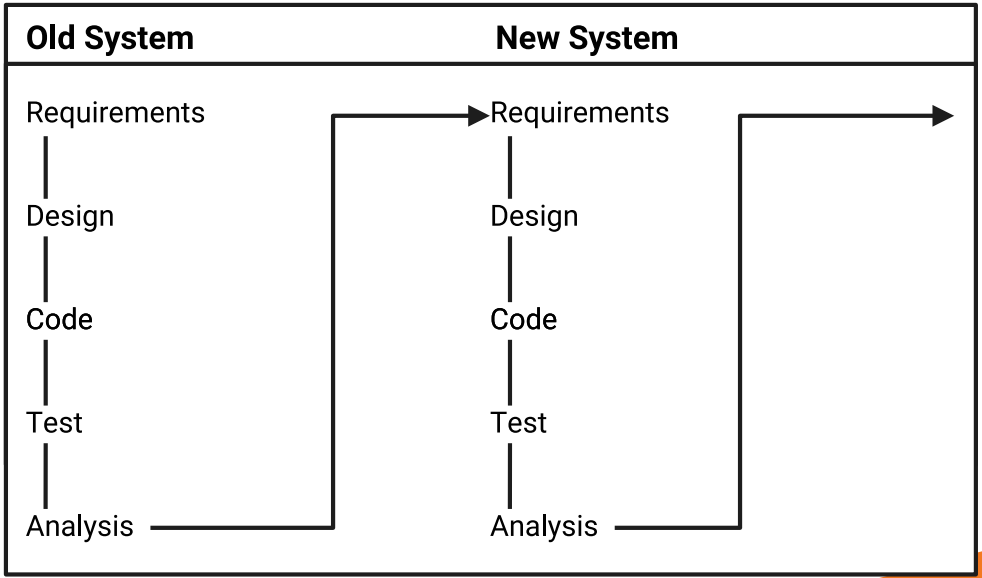
The **iterative enhancement model**, shown in the figure, depicts how changes flow from the very top level documents to the lowest-level documents.

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.

Idaho State University
Computer Science

Iterative Enhancement Model

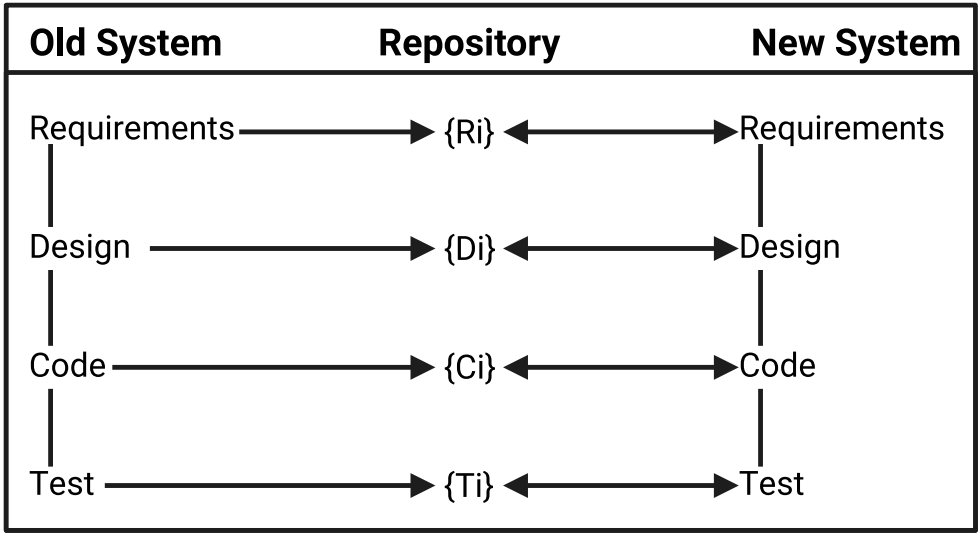


ROAR

Reuse Oriented Models

- The main assumption in this 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

Full Reuse Model



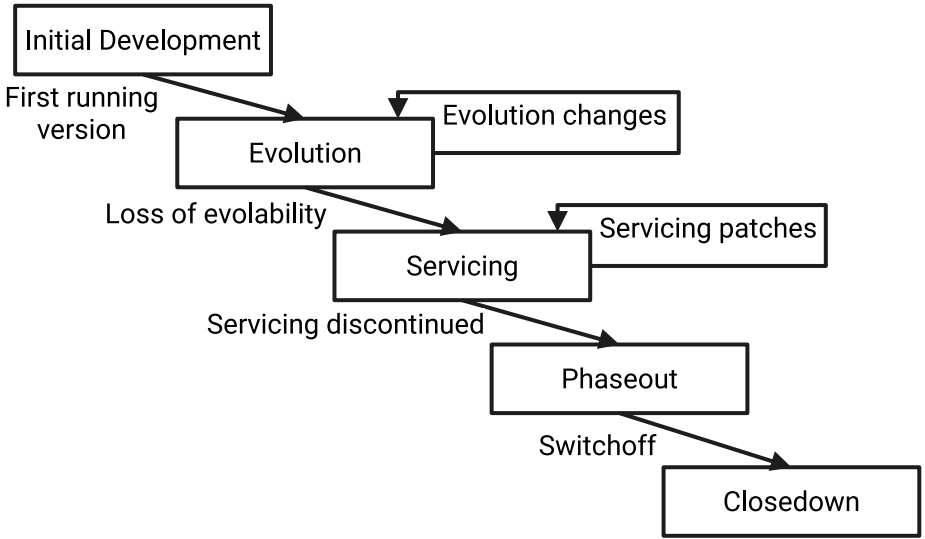
The Staged Model for CSS

- 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** – The first delivered version is produced. Knowledge about the system is fresh and constantly changing. In fact, change is the rule rather than the exception. Eventually, an architecture emerge and stabilizes.
  - **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.

The Staged Model for CSS

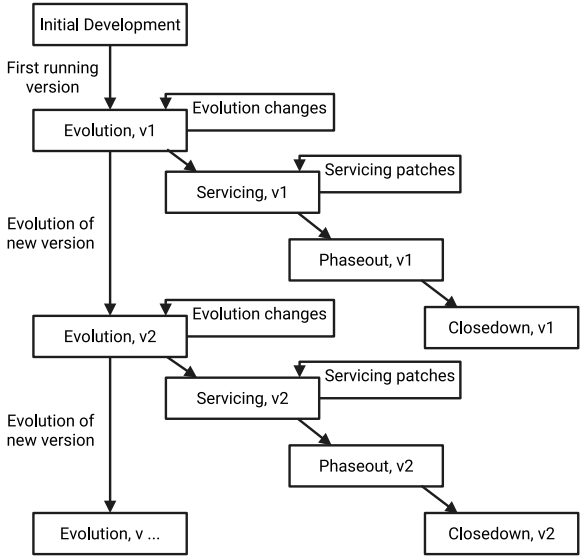
- Rajlich and Bennett have presented a descriptive model of software evolution called the staged model of maintenance and evolution.
- Its primary objective is at improving understanding of how long-lived software evolves, rather than aiding in its management.
- Their model divides the lifespan of a typical system into four stages:
  - **Servicing** – The system is no longer a key asset for the developers, who concentrate mainly on maintenance tasks rather than architectural or functional change. Knowledge about the system has lessened and the effects of change have become harder to predict. The costs and risks of change have increased significantly.
  - **Phase out** – It has been decided to replace or eliminate the system entirely, either because the costs of maintaining the old system have become prohibitive or because there is a newer solution waiting in the pipeline. An exit strategy is devised and implemented, often involving techniques such as wrapping and data migration. Ultimately, the system is shut down.

The Staged Model for CSS



Idaho State University
Computer Science

The Staged Model for CSS



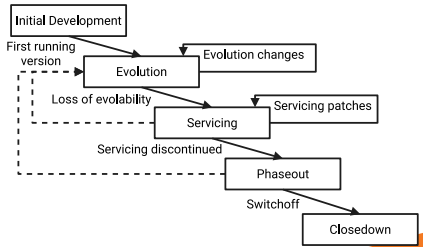
ROAR

Idaho State University
Computer Science

The Staged Model for FLOSS

- Three major differences are identified between CSS systems and FLOSS systems
- In the figure, the rectangle with the label “Initial development” has been visually highlighted because it can be the only initial development stage in the evolution of FLOSS systems. In other words, it does not have any evolution track for FLOSS systems.
- With some systems that were analyzed, after a transition from Evolution to Servicing, a new period of evolution was observed. This possibility is depicted in the Figure as a broken arc from the Servicing stage to the Evolution stage.

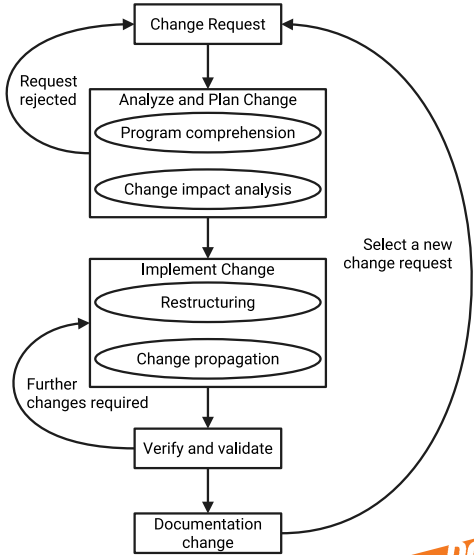
- In general, the active developers of FLOSS systems get frequently replaced by new developers. Therefore, the dashed line in the Figure exhibits this possibility of a transition from Phaseout to Evolution stage.



ROAR

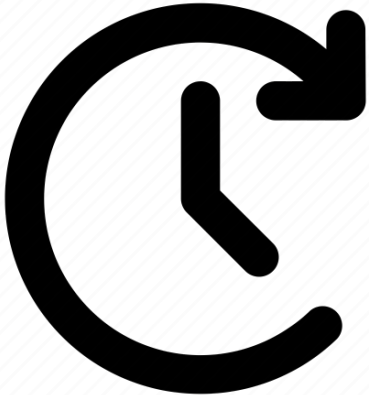
Change Mini-Cycle Model

- Software change is a process that may introduce new requirements to the existing system, or may need to alter the software system if requirements are not correctly implemented.
- In order to capture this, an evolution model known as **change mini-cycle** was proposed by Yau et al. in the late seventies.



For Next Time

- Review EVO Chapter 3.1 - 3.5
- Read EVO Chapter 3.6 - 3.7
- Watch Lecture 05
- **4423: Work on Homework 01**
- **4423: Course Project Part 1: Team Selection is due Friday at 11:00 pm**



Are there any questions?