Software Life-Cycle Models

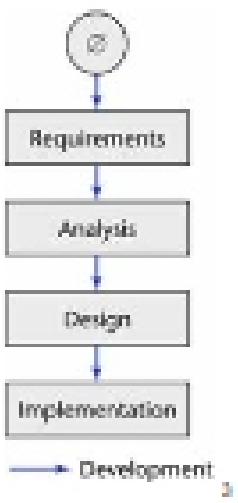
Xiaojun Qi

Life-Cycle Model

- It specifies the various phases/workflows of the software process, such as the requirements, analysis (specification), design, implementation, and postdelivery maintenance, and the order in which they are to be carried out.

Software Development in Theory

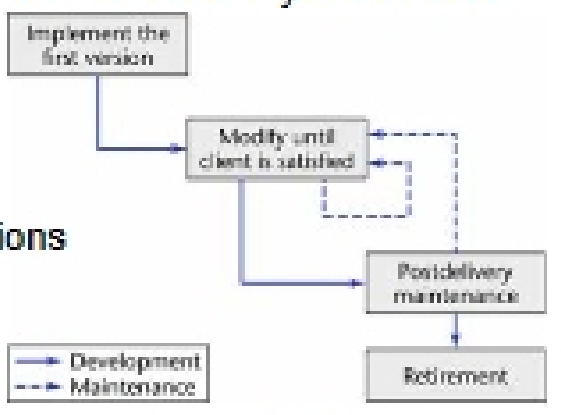
- Ideally, software is developed as described in Chapter 1
  - Linear
  - Starting from scratch



Software Development in Practice

- In the real world, software development is totally different and is more chaotic
  - Software professionals make mistakes
  - The client's requirements change while the software product is being developed
  - A software product is a model of the real world, and the real world is continually changing.

1. Code-and-Fix Life-Cycle Model



- No design
- No specifications

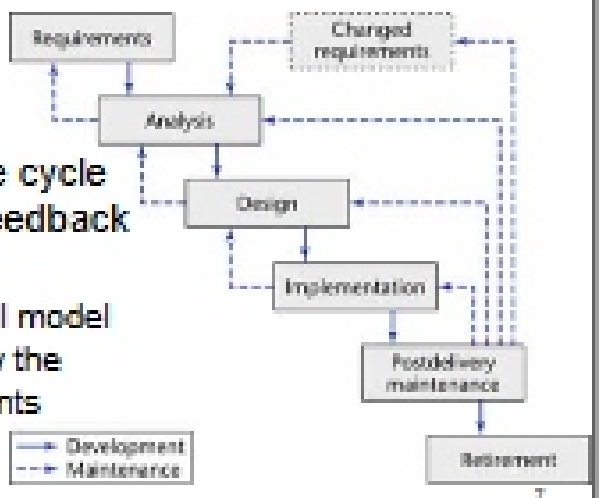
The easiest way to develop software
The most expensive way for maintenance
(i.e., maintenance nightmare)

Code-and-Fix Life-Cycle Model (Cont.)

- The product is implemented without requirements or specifications, or any attempt at design.
- The developers simply throw code together and rework it as many times as necessary to satisfy the client.
- It is used in small project and is totally unsatisfactory for products of any reasonable size.

2. Waterfall Life-Cycle Model

- The linear life cycle model with feedback loops
  - The waterfall model cannot show the order of events



7

Waterfall Life-Cycle Model (Cont.)

- No phase is complete until the **documentation** for that phase has been completed and the products of that phase have been approved by the **software quality assurance** (SQA) group.
- If the products of an earlier phase have to be changed as a consequence of following a **feedback loop**, that earlier phase is deemed to be complete only when the documentation for the phase has been modified and the modifications have been checked by the SQA group.

8

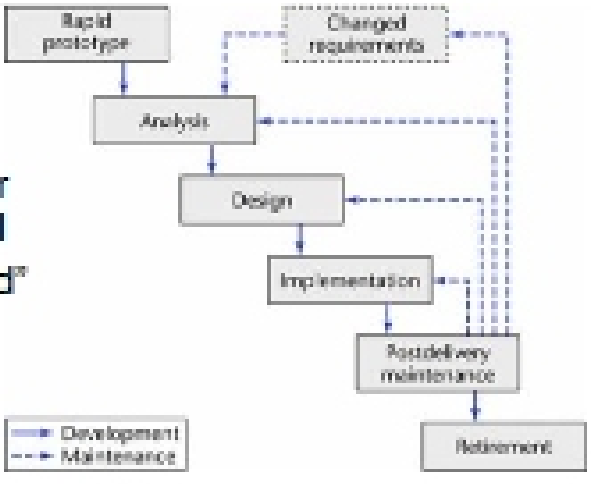
Waterfall Life-Cycle Model (Cont.)

- Advantages:
  - Documentation is provided at each phase
  - All the products of each phase (including the documentation) are meticulously checked by SQA. → Maintenance is easier
- Disadvantages:
  - Specification documents are long, detailed, and boring to read.

9

3. Rapid-Prototyping Life-Cycle Model

- Linear model
- "Rapid"



10

Rapid-Prototyping Life-Cycle Model (Cont.)

- A rapid prototype is a working model that is functionally equivalent to a subset of the product.
- The first step is to build a rapid prototype and let the client and future users interact and experiment with the rapid prototype.
- Strength:
  - The development of the product is essentially **linear**, proceeding from the rapid prototype to the delivered product.
  - The feedback loops of the waterfall model are less likely to be needed in the rapid prototyping model.
  - It is built rapidly and modified rapidly to reflect the client's needs. → Speed is of the essence.

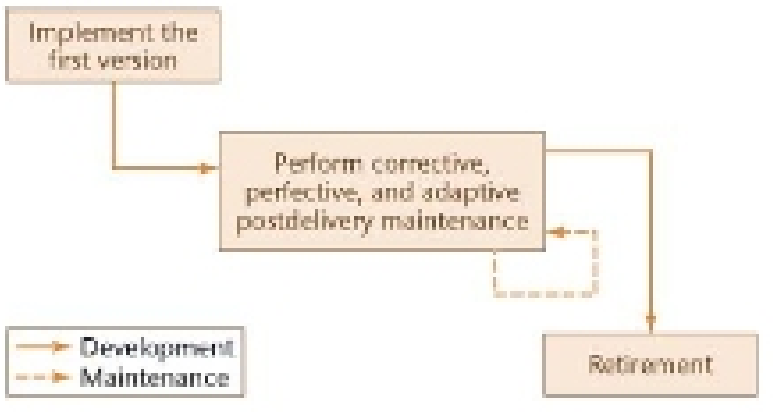
11

Rapid-Prototyping Life-Cycle Model (Cont.)

- Weakness:
  - One the client's real needs have been determined, the rapid prototype implementation is discarded.
- The lessons learned from the rapid prototype implementation are retained and used in subsequent development phases.

12

4. Open-Source Life-Cycle Model



Postdelivery maintenance life-cycle model

Open-Source Life-Cycle Model (Cont.)

- **The first informal phase**
  - One individual builds an initial version and makes it available via the Internet (e.g., SourceForge.net)
  - If there is sufficient interest in the project, the initial version is widely downloaded; users become co-developers; and the product is extended
- Key point: Individuals generally work voluntarily on an open-source project in their spare time

Open-Source Life-Cycle Model (Cont.)

- **The second Informal Phase**
  - Reporting and correcting defects
    - Corrective maintenance
  - Adding additional functionality
    - Perfective maintenance
  - Porting the program to a new environment
    - Adaptive maintenance
- The second informal phase consists *solely* of postdelivery maintenance
  - The word "co-developers" on the previous slide should rather be "co-maintainers"

Open-Source Life-Cycle Model (Cont.)

- An initial working version is produced using the rapid-prototyping model, the code-and-fix model, and the open-source life-cycle model.
- The initial version of the rapid-prototyping model is then discarded. The initial versions of Code-and-fix model and open-source life-cycle model become the target product
- There are generally no specifications and no design. However, open-source software production has attracted some of the world's finest software experts. They can function effectively without specifications or designs

Open-Source Life-Cycle Model (Cont.)

- A point will be reached when the open-source product is no longer maintainable
- The open-source life-cycle model is restricted in its applicability
  - It can be extremely successful for infrastructure projects, such as : Operating systems (Linux, OpenBSD, Mach, Darwin), Web browsers (Firefox, Netscape), Compilers (gcc), Web servers (Apache), and Database management systems (MySQL)
  - There cannot be open-source development of a software product to be used in just one commercial organization
  - The open-source life-cycle model is inapplicable unless the target product is viewed by a wide range of users as useful

Open-Source vs. Closed-Source

- Closed-source software is maintained and tested by employees
  - Users can submit failure reports but never fault reports
- Open-source software is generally maintained by unpaid volunteers
  - Users are strongly encouraged to submit defect reports, both failure reports and fault reports
    - Core group: Small number of dedicated maintainers with the inclination, the time, and the necessary skills to submit fault reports ("fixes"); They take responsibility for managing the project; They have the authority to install fixes
    - Peripheral group: Users who choose to submit defect reports from time to time