Life-Cycle Model

Ran Liao

May 22, 2019

Waterfall Model

Waterfall model is characterized by its feedback loops and documentation-driven feature. Due to the documentation, maintenance is easier.

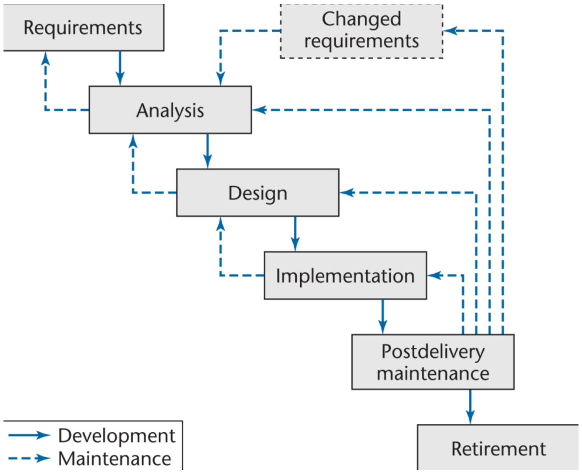


Figure 1: Waterfall Model

Iteration and Incrementation Model

The iterative-and-incremental life-cycle model is a model that applies waterfall model successively. Each increment is a waterfall mini project. Each successive version is intended to be closer to its target than its predecessor. The strengths of this model are as follows,

1. Since every iteration incorporates the test workflow, there are multiple opportunities for checking that the software product is correct. Therefore, faults can be detected and corrected early.

2. The robustness of the architecture can be determined early in the life cycle.

3. Risks can be mitigated (resolve) early.

4. A working version of the software product is available from the start. And delivered partial versions can help smooth the introduction of the new product in the client organization

Code-and-Fix Model

There's no design process or specifications in this model. It is the easiest way to develop software, but also the the most expensive way in terms of maintenance.

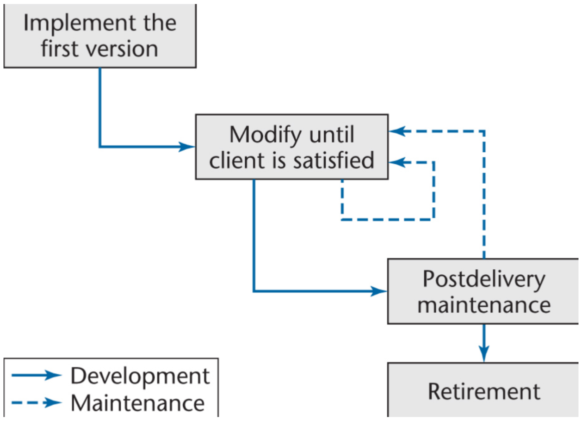


Figure 2: Code-and-Fix Model

Rapid Prototyping Model

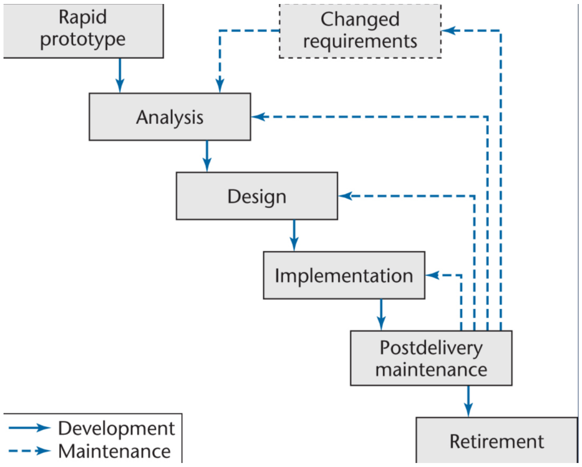


Figure 3: Rapid Prototyping Model

Open-Source Life-Cycle Model

There're two informal phases in this model. First, one individual builds an initial version and made available via the Internet. Then, if there is sufficient interest in the project, users become co-developers and the product is extended. In this model, individuals generally work voluntarily on the project in their spare time. The second informal phase consists solely of postdelivery maintenance including

- Reporting and correcting defects (Corrective maintenance)
- Adding additional functionality (Perfective maintenance)
- Porting the program to a new environment (Adaptive maintenance)

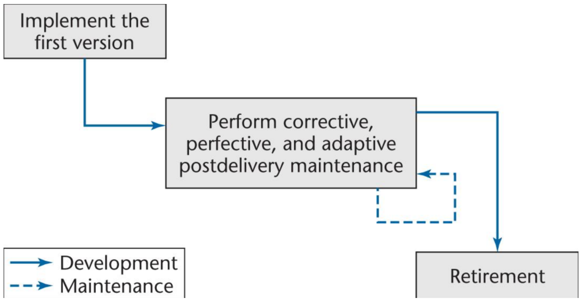

Implement the first version
Perform corrective, perfective, and adaptive postdelivery maintenance
Retirement
Development
Maintenance

Figure 4: Open-Source Model

The core group of an open-source software consists of a 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 and have the authority to install fixes. Peripheral group consists of users who choose to submit defect reports from time to time.

Open-source softwares are a lot different from closed-source softwares. See Table 1 for details.

An open-source project generally has no specifications and no design. Some open-source projects so successful because they have attracted some of the world's finest software experts. Those experts can function effectively without specifications or designs.

When the open-source model has worked, it has sometimes been incredibly successful. However, the open-source life-cycle model is inapplicable unless the target product is viewed by a wide range of users as useful to them. In reality, about half of the open-source projects on the Web have not meet this requirement. Even where work has started, the overwhelming preponderance will never be completed.

Table 1: Difference between open-source software and closed-source software

Open-source	Closed-source
Maintained by unpaid volunteers.	Maintained and tested by employees.
Users are strongly encouraged to submit defect reports, both failure reports and fault reports.	Users can submit failure reports but never fault reports (the source code is not available).
The core group releases a new version of an open-source product as soon as it is ready.	New versions of closed-source software are typically released roughly once a year.
Core group performs minimal testing. Extensive testing is performed by the members of the peripheral group in the course of utilizing the software.	Carefully tested by the SQA group.

Agile Processes

Agile processes are a collection of new paradigms characterized by

- Less emphasis on analysis and design
- Earlier implementation (working software is considered more important than documentation)
- Responsiveness to change
- Close collaboration with the client

In agile processes, the team will deliver working software frequently, ideally every 2 or 3 weeks. Typically, 3 weeks for each iteration is set aside for a task. The team members then do the best job they can during that time.

Another common feature of agile processes is stand-up meetings. It's a short meeting that held at a regular time each day. During the meeting, participants stand in a circle. They do not sit around a table to ensure the meeting lasts no more than 15 minutes. The aim of a stand-up meeting is to raise problems, but not solve them. Solutions are found at follow-up meetings, preferably held directly after the stand-up meeting.

In conclusion, agile processes appear to be a useful approach to building small-scale software products when the client's requirements are vague. Also, some of the proven features of agile processes can be effectively utilized within the context of other life-cycle models.

Synchronize-and Stabilize Model

This is Microsoft's life-cycle model. First, a requirements analysis is conducted. They may complete this through interview potential customers. Then they draw up specifications and divide project into 3 or 4 builds. Note that each build is carried out by small teams working in parallel. At the end of the day, they synchronize (test and debug) codes. At the end of the build, they stabilize (freeze) the build.

Spiral Model

Spiral Model is simplified form rapid prototyping model plus risk analysis preceding each phase.

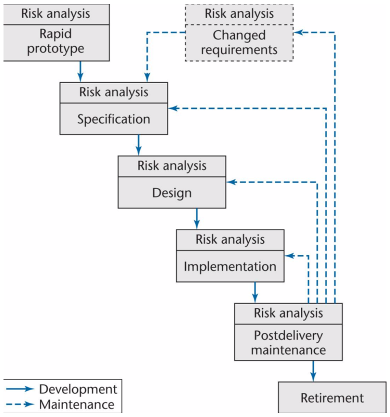


Figure 5: Spiral Model

In full spiral model, each phase is preceded by alternatives and risk analysis. And each phase is followed by evaluation and planning of the next phase.

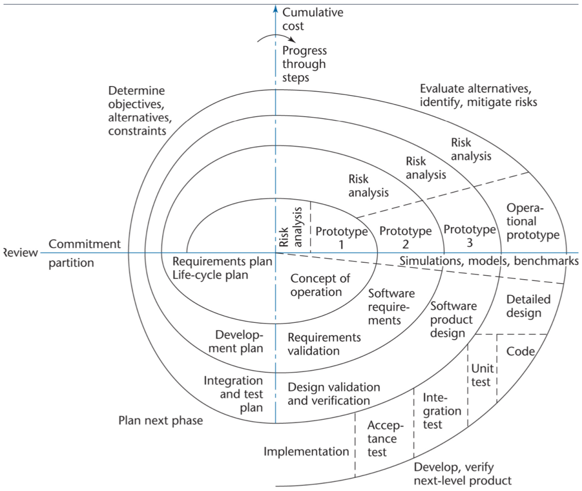


Figure 6: Full Spiral Model

Evolution-Tree Model

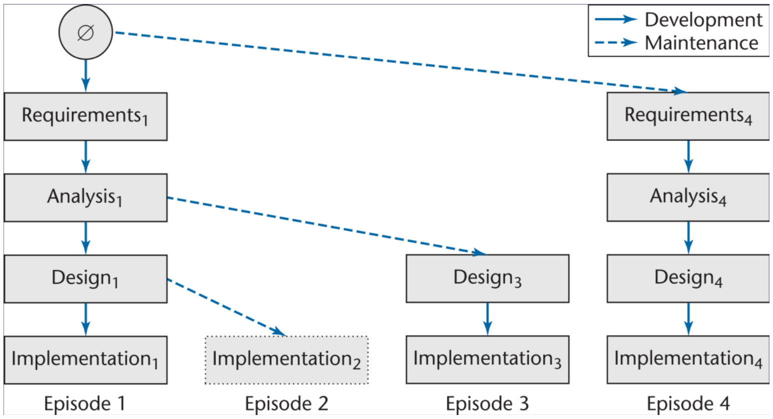
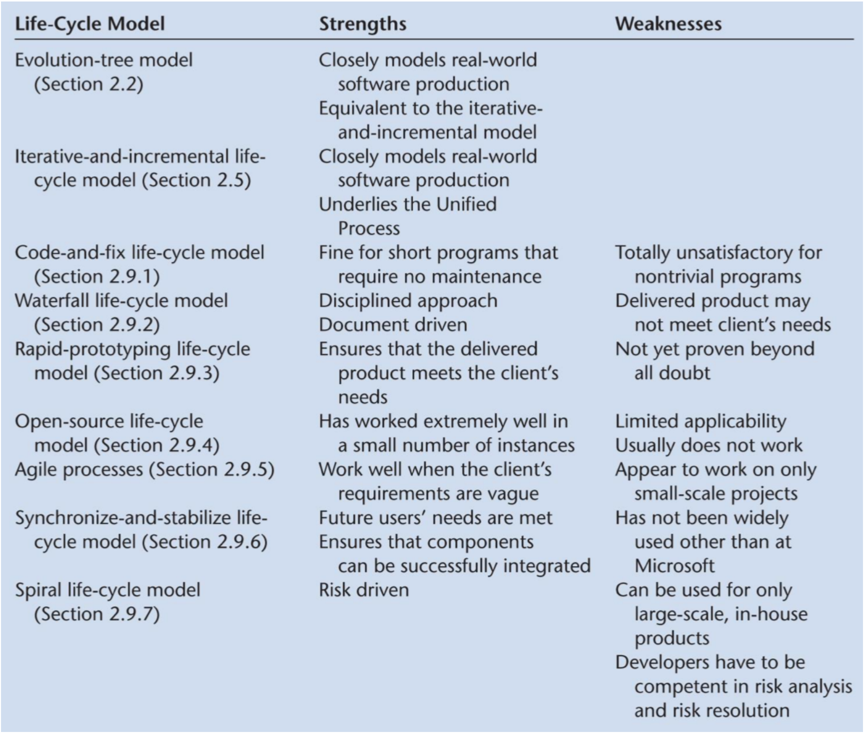


Figure 7: Evolution-Tree Model

Comparison of Life-Cycle Models



Life-Cycle Model	Strengths	Weaknesses
Evolution-tree model (Section 2.2)	Closely models real-world software production Equivalent to the iterative-and-incremental model	
Iterative-and-incremental life-cycle model (Section 2.5)	Closely models real-world software production Underlies the Unified Process	
Code-and-fix life-cycle model (Section 2.9.1)	Fine for short programs that require no maintenance	Totally unsatisfactory for nontrivial programs
Waterfall life-cycle model (Section 2.9.2)	Disciplined approach Document driven	Delivered product may not meet client's needs
Rapid-prototyping life-cycle model (Section 2.9.3)	Ensures that the delivered product meets the client's needs	Not yet proven beyond all doubt
Open-source life-cycle model (Section 2.9.4)	Has worked extremely well in a small number of instances	Limited applicability Usually does not work
Agile processes (Section 2.9.5)	Work well when the client's requirements are vague	Appear to work on only small-scale projects
Synchronize-and-stabilize life-cycle model (Section 2.9.6)	Future users' needs are met Ensures that components can be successfully integrated	Has not been widely used other than at Microsoft
Spiral life-cycle model (Section 2.9.7)	Risk driven	Can be used for only large-scale, in-house products Developers have to be competent in risk analysis and risk resolution

Figure 8: Comparison of Life-Cycle Models