CS 430 - Lecture 04 - Life-Cycle Models

Collin Roberts

September 19, 2023

Outline

1. Other Life Cycle Models
    1. Code and Fix Life-Cycle Model
    2. Waterfall (Modified) Life-Cycle Model
    3. Rapid Prototyping Life-Cycle Model
    4. Open Source Life-Cycle Model
    5. Agile Processes
    6. Synchronize and Stabilize Life-Cycle Model
    7. Spiral Life-Cycle Model
2. Comparison of Life-Cycle Models

Code and Fix Life-Cycle Model

Key Idea: Implement the product without requirements, specification or design.

Remarks:

1. See Figure 2.8 in the text or on slide 17 for Chapter 2; but know that it is the only possible picture without requirements, specification or design.

Code and Fix Life-Cycle Model

Strengths:

1. This technique may work on very small systems ($\leq$ 200 lines of code).
2. Easy to incorporate changes to requirements.
3. Generates a lot of lines of code (whether this is actually a strength depends on organizational norms).

Code and Fix Life-Cycle Model

Weaknesses:

1. This technique is totally unsuitable for systems of any reasonable size.
2. This technique is unlikely to yield the optimal solution.
3. Slow.
4. Costly.
5. Likelihood of regression faults is high.

Code and Fix Life-Cycle Model

Remarks:

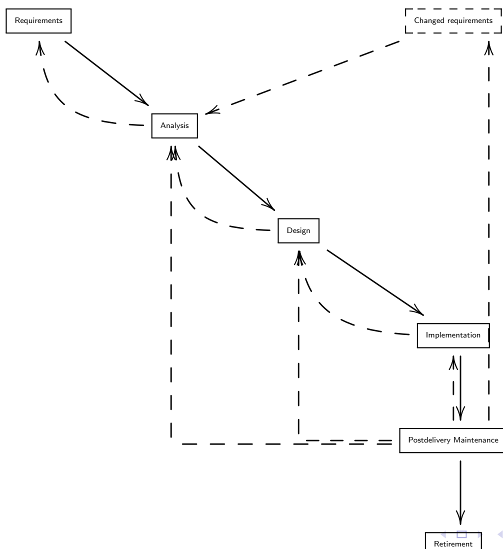
1. It is appropriate (and really the only choice) for a user base of size 1, e.g. for any programming assignment you would do for a CS assignment at uWaterloo.
2. We met this model once before: it was the only model in existence before the Waterfall model was introduced in 1970.

Waterfall (Modified) Life-Cycle Model

Key Idea: Augment the "vanilla" waterfall diagram, to add the "feedback loops" during the project, and for post-delivery maintenance.
Here is a sketch of Figure 2.9 in the text, using the key:

.

Waterfall (Modified) Life-Cycle Model



Waterfall (Modified) Life-Cycle Model

Remarks:

1. No phase is complete until all its documents are complete, and the output(s) of the phase are approved by the **SQA (Software Quality Assurance)** team.
2. Testing is carried out throughout the project.

Waterfall (Modified) Life-Cycle Model

Strengths:

1. Discipline enforced by SQA.

Waterfall (Modified) Life-Cycle Model

Weaknesses:

1. Specification documents are often written in a way that does not enable the client to understand what the finished product will look like.
   1. Hence specification documents may not be fully understood before they are approved.
   2. Hence the finished product may not actually meet the client's needs.

The next model, **rapid prototyping**, is an adaptation if the waterfall model to address this key weakness.

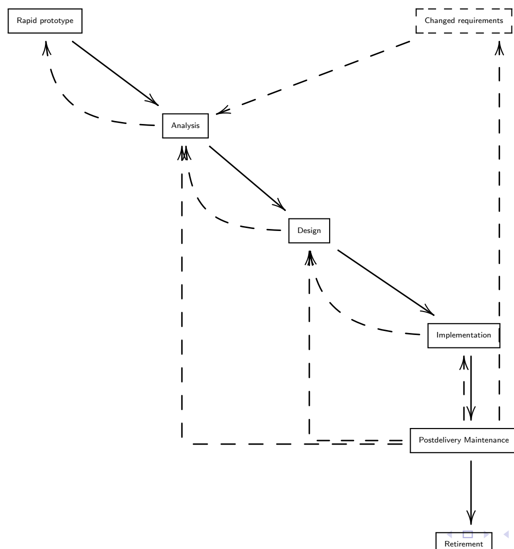
Rapid Prototyping Life-Cycle Model

Here is a sketch of Figure 2.10 in the text, using the key:

⟶	Development
- - ➤	Maintenance

.

Rapid Prototyping Life-Cycle Model



Rapid Prototyping Life-Cycle Model

Remarks:

1. This diagram looks almost identical to that for Waterfall (Modified).
2. **Key Difference:** Requirements has been replaced with Rapid Prototype. Huh?

Rapid Prototyping Life-Cycle Model

Definition 1

A **rapid prototype** *is a working model that is functionally equivalent to a subset of the software product.*

Rapid Prototyping Life-Cycle Model

Motivation: Develop a rapid prototype (during Requirements phase) to let the client interact and experiment with it early. This way the requirements document can be written with higher confidence that the software product it describes will meet the client's needs. Users can give better feedback from working with a rapid prototype than from reading a long requirements document.

Rapid Prototyping Life-Cycle Model

Examples:

1. If the product is a payroll system, then a rapid prototype might have a subset of the screens and might produce mocked-up pay stubs, but might not have any database updating or batch processing behind the scenes.

Rapid Prototyping Life-Cycle Model

Remarks:

1. The feedback loops from the waterfall model are less heavily used here.
2. The word "rapid" is crucial. Speed is of the essence!

Rapid Prototyping Life-Cycle Model

Summary: The purpose of a rapid prototype is to **improve requirements**.

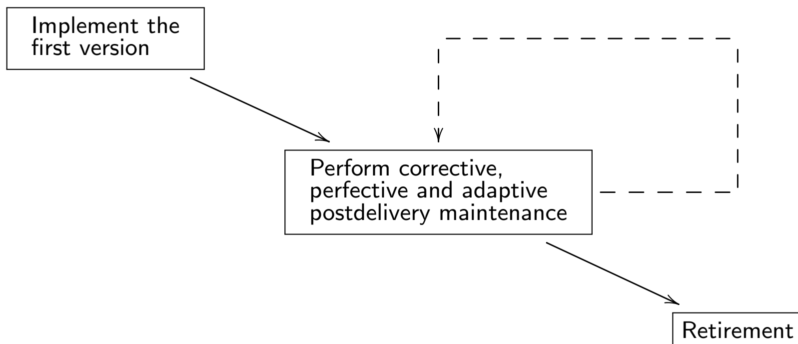
Open Source Life-Cycle Model

Here is a sketch of Figure 2.11 in the text, using the key:

$\longrightarrow$	Development
$- - \succ$	Maintenance

.

Open Source Life-Cycle Model



Open Source Life-Cycle Model

Key Idea: Open Source software projects proceed in two phases:

1. A single individual has an idea for a program (e.g. MySQL, LibreOffice, Notepad++, R, Linux, Firefox, Apache, etc.), builds the initial version, and makes it available free of charge to anyone who wants a copy.

Open Source Life-Cycle Model

2. (Informal) If there is sufficient interest, then users become co-developers (co-maintainers) for Post-Delivery Maintenance:
    1. Report / correct faults (Corrective Maintenance)
    2. Add additional functionality (Perfective Maintenance)
    3. Port the program to new platforms (Adaptive Maintenance)

Open Source Life-Cycle Model

3. All participants can offer suggestions:
    1. new features
    2. new platforms
4. Participation is voluntary and unpaid.
5. **Roles:**
    1. Core group: dedicated maintainers
    2. Peripheral group: suggest bug fixes from time to time
6. Success depends on the interest generated by the initial version.

Open Source Life-Cycle Model

Many open source projects do not amount to anything. But there have been some spectacularly successful examples (mentioned at the beginning of the section).

Open Source Life-Cycle Model

Reasons Why Open Source Projects Are Successful:

1. Perception that the initial release is a "winner" (most important)
2. Large potential user base

Open Source Life-Cycle Model

Instructor Remarks:

1. Participation in an Open Source project is voluntary and unpaid.
2. The idea of Open Source is in direct conflict with a corporation's need to achieve competitive advantage, by writing good software.

Agile Processes

Guiding Principles

1. Communication
2. Speed: Satisfying the Client's needs as quickly as possible (ideally new versions every 2-3 weeks)

Agile Processes

According to the **Scrum Method**, we iterate through the following two phases until the backlog of tasks is empty.

Requirements	Sprints
User Stories Prioritization Build Backlog of Tasks	Daily Meetings Eventually Reassign Tasks

Agile Processes

Techniques to ensure frequent delivery of new versions:

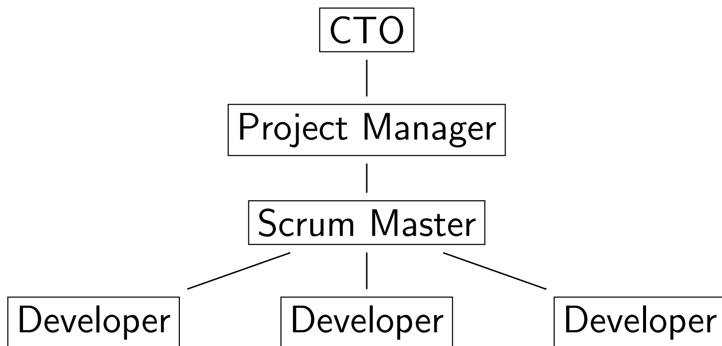
1. **timeboxing:** Fix an amount of time to work on a task; do as much as possible on the task during that time window. Agile processes demand fixed time, not fixed features.

Agile Processes

2. **daily 15 minute stand-up meeting (to raise and resolve issues):** Each team member answers five questions:
    1. What have I done since yesterday's meeting?
    2. What am I working on today?
    3. What problems are preventing me from achieving my goal for today?
    4. What have we forgotten?
    5. What did I learn that I would like to share with the team?

Differences Between Agile and Classical

Diagram of Team Organization:



1. 1-week “sprints”
2. Each sprint gets us closer to the ultimate goal.

Agile Processes

1. Iterative process
2. One phase need not finish before the next can start
3. A client representative sits with the IT team
4. No specializations
5. Members from all different areas work together at different times
6. Working software is prioritized over detailed documentation
7. test-driven development

Agile Processes

Strengths:

1. Speed
2. Flexibility
3. Team Cohesion
4. Some history of success with smaller projects.

Agile Processes

Weaknesses:

1. Heavy on meetings
2. Not scalable with team size
3. This technique is untested on large projects (many software professionals have expressed doubts that this will be successful)

Agile Processes

When you have time, you may enjoy watching this YouTube video about Iteration & Incrementation Leading to Agile Processes: `https://youtu.be/Vlc2r_U30yo`

Agile Processes

Remarks on Agile Processes:

1. The text makes a big deal of **Extreme Programming (XP)**, and states that a key feature of XP is **pair programming**. I had always suspected that this was a bit too rigid - now we have this suspicion confirmed by presentations from students who have worked under this model. It made a lot more sense to me that the groups formed to do the work need not always be pairs - they are whatever is appropriate to the task at hand.

Synchronize and Stabilize Life-Cycle Model

This is Microsoft's adaptation of Iteration and Incrementation.

Synchronize and Stabilize Life-Cycle Model

1. Pull requirements from the clients.
2. Write Specification document.
3. Divide the work into four **builds** (most important features in earlier builds):
   1. critical
   2. major
   3. minor
   4. trivial

   N.B. Developers can add requirements during a build.

Synchronize and Stabilize Life-Cycle Model

4. Carry out each build using small teams working in parallel.
5. **Synchronize** at the end of each day, then
6. **Stabilize** at the end of each build (then freeze).

Synchronize and Stabilize Life-Cycle Model - Strengths

1. Users' needs are met
2. Components are successfully integrated
3. Tolerant of changes to specifications
4. Encourages individual developers to be innovative and creative
5. Daily synchronization and Build-ly stabilization ensure developers will all work in the same direction
6. Good for large projects

Synchronize and Stabilize Life-Cycle Model - Weaknesses

1. So far, this has only been used successfully at Microsoft

Spiral Life-Cycle Model

This incorporates elements of several of the earlier models.
Key Problem: There are many risks associated with software development projects, which if realized will mean that the project is a failure.

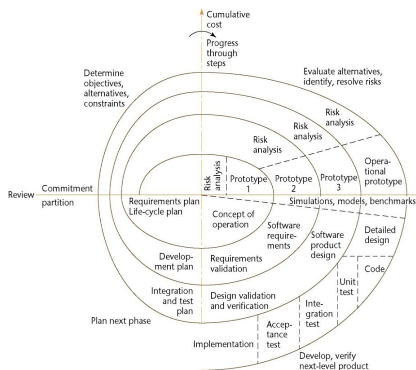
Spiral Life-Cycle Model

Key Ideas:

1. Minimize risks inherent in software development by the (repeated) use of **proof-of-concept prototypes** and other means.
2. N.B. Unlike **rapid prototypes**, which aim to improve requirements by letting users interact with a subset of the target functionality, a **proof-of-concept prototype** aims to determine whether an architecture design is good (e.g. will it perform quickly enough?)

Spiral Life-Cycle Model

Figure 2.13: Spiral, Full



Spiral Life-Cycle Model

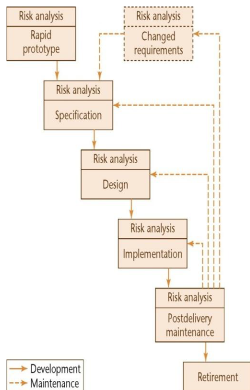
Remarks:

1. The quadrants in the above diagram could be labelled:

1. Planning / Requirements	2. Risk Analysis
4. Plan Next Phase	3. Develop and Verify

Spiral Life-Cycle Model

Figure 2.12: Spiral, Simplified



Spiral Life-Cycle Model - Strengths

1. Emphasis on alternatives and constraints supports re-use, and software quality.
2. This technique encourages doing the correct amount of testing.

Spiral Life-Cycle Model - Weaknesses

1. This model is only meant for internal building of large-scale software.
2. If risks are not analyzed correctly, then all may appear fine even when the project is headed for disaster.
3. Makes the (often wrong) assumption that software is developed in discrete phases, when in reality, software is developed iteratively and incrementally (like in the Winburg example).

Comparison of Life-Cycle Models

Here is Figure 2.14 from the text:

Comparison of Life-Cycle Models

Life-Cycle Model	Strengths	Weaknesses
Evolution Tree (§2.2)	-Closely models real-world software production -Equivalent to iteration and incrementation	
Iteration and Incrementation (§2.5)	-Closely models real-world software production -Underlies the Unified Process	
Code-and-fix (§2.9.1)	-Fine for short programs that require no maintenance	-Totally unsuitable for non-trivial programs
Waterfall (§2.9.2)	-Disciplined approach -Document driven	-Delivered product may not meet client's needs
Rapid Prototyping (§2.9.3)	-Ensures the delivered product meets the client's needs	-Not yet proven beyond all doubt
Open Source (§2.9.4)	-Has worked extremely well in a small number of instances	-Limited applicability -Usually does not work
Agile Processes (§2.9.5)	-Works well when the client's requirements are vague	-Appear to work on only small-scale projectes
Synchrionize-and-stabilize (§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 (§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