CS 430 - Lecture 02 - The Classical and Object-Oriented Paradigms

Collin Roberts

September 14, 2023

Outline

Example: Classical (Waterfall) Life-Cycle Model

Refer to Fig 1.2 in the text for the phases of the Classical (Waterfall) life-cycle model:

1. Requirements phase
2. Analysis (specification) phase
3. Design phase
4. Implementation phase
5. Postdelivery maintenance
6. Retirement

Classical Paradigm

Refer to the Examples document on LEARN (Lecture 02)

Classical Paradigm

Why does the Waterfall life-cycle model not have any of the following phases?

1. Planning
2. Testing
3. Documentation

Classical Paradigm

Answer:

1. All three activities are crucial to project success.
2. Therefore all three activities must happen throughout the project and cannot be limited to just one project phase.

Classical Paradigm

Question from the Class: Why do we study the Classical life-cycle model in CS 430?

Classical Paradigm

Answer:

1. Understand why OO is better.
2. Many organizations still use Classical.
3. Much legacy code still exists, that was written using Classical techniques.

Object-Oriented Paradigm

Refer to the Examples document on LEARN (Lecture 02)

Object-Oriented Paradigm

Question from the Class: Why does OO not come with a life-cycle picture, as Classical does?

Object-Oriented Paradigm

Answer:

1. The change from Classical to Object Orientation is more a change of mindset than of methodology.
2. We change our mindset from building **one monolithic thing** (Classical) to building **many smaller classes that do work for us together** (OO). Many life-cycle models (including Waterfall) can be used to build these classes effectively.

Maintenance Aspects

We will look at maintenance in the context of the Classical (aka Waterfall) Life-Cycle Model, invented in 1970. Phases:

1. Requirements
   1. Elicit client requirements.
   2. Understand client needs.
2. Analysis
   1. Analyze client requirements.
   2. Draft specification document - formal.
   3. Draft Software Project Management Plan (SPMP).
3. Design
   1. Design architecture - divide software functionality into components.
   2. Draft detailed design for each component.

Maintenance Aspects

4. Implementation
    1. Coding (development) - code & document each component
    2. Unit test each individual component
    3. Integration (system) testing - combine components, test interfaces among components
    4. Acceptance testing - use live data in client's test environment. Clients participate in testing & verification of test results, and sign off when they are happy with the results.
    5. Deploy to production environment.

Maintenance Aspects

5. Post delivery maintenance - maintain the software while it's being used to perform the tasks for which it was developed.

Maintenance Aspects

Definition 1

Corrective Maintenance: *Removal of residual faults while software functionality & specs remain relatively unchanged. (aka fix* **production problems**)

Maintenance Aspects

Definition 2

Perfective Maintenance:

1. *Implement changes the client thinks will improve effectiveness of product (e.g. additional functionality, reduce response time) (aka* **enhancements** *or* **upgrades***)*
2. *Specs must be changed*

Maintenance Aspects

Definition 3

Adaptive Maintenance:

1. *Change the software to adapt to changes in environment (e.g. new policy, tax rate, regulatory requirements, changes in systems environment) - may not necessarily add to functionality. You allow software to survive*
2. *Specs may change to address the new environment*

Maintenance Aspects

6. Retirement
    1. Product is removed from service: functionality provided by software is no longer useful / further maintenance is no longer economically feasible.

The Importance of Postdelivery Maintenance

- Shelf life of good software: 10, 20, even 30 years
- Good software is a model of real world & real world keeps changing, therefore software must change too.
- Cost of Post delivery Maintenance continues to go up, while (possibly surprisingly) cost of implementation is nearly flat.

The Importance of Postdelivery Maintenance

Example: My first project at OpenText was to develop a **Consolidated Customer Database**. After the initial scrubbing of the data, management opted not to re-scrub the following year. The database withered and died because management was unwilling to pay for post delivery maintenance.

Requirements, Analysis and Design Aspects

Key Facts:

1. The earlier in the life cycle a fault is found, the cheaper it is to fix. (See Figures 1.5 and 1.6 on pp13-14 of the text.)
2. Correcting a fault in the early phases usually just requires changing a document.
3. Hence the requirements, analysis and design phases need to be improved.

Team Development Aspects

Remarks:

1. Hardware keeps getting cheaper and cheaper, and able to run more and more complex programs.
2. Hence modern software must be developed by **teams**.
3. But this can lead to problems, e.g.
    1. Communication becomes challenging when teams are far apart geographically, especially when they are in different time zones.
    2. Interpersonal problems can undermine team effectiveness.
    3. if a call to a module written by another developer mentions the arguments in the wrong order. (If the variable types are the same, then even the compiler may

Team Development Aspects

4. Software Engineering must include techniques for ensuring teams are properly managed.

The Object-Oriented Paradigm

Problems With The Classical Paradigm

1. Works well for small systems (≤ 5000 lines of code), but does not scale effectively to larger systems.
2. Fails to address growing costs of post-delivery maintenance.

Reason: Classical techniques focus on **data** or **operation**, but **not both**.

The Object-Oriented Paradigm

Contrast With The Object-Oriented Paradigm:

1. The object-oriented paradigm treats **data** (attributes) and **operations** (methods) together, as equally important.

The Object-Oriented Paradigm In Perspective

Remarks:

1. Like any software production technique, the OO paradigm must be applied correctly to be effective.
2. The OO paradigm is the best technique invented so far; yet is is sure to be superseded by a superior technique in the future.

Ethical Issues

Remarks:

1. Since software is developed by people, there are ethical issues connected with software development.

Ethical Issues

2. Software engineers commit to these ethical principles (each is explained more fully in the text):
    1. Public
    2. Client and Employer
    3. Product
    4. Judgment
    5. Management
    6. Profession
    7. Colleagues
    8. Self