CS 430 Course Notes

Fall 2025 - Collin Roberts

Talha Yildirim,
tyildir [at] uwaterloo [dot] ca

Contents

1 The Scope of Software Engineering

1.1 The Classical and Object Oriented Paradigms

Definition 1.1.1 (Classical (Waterfall) Life Cycle Model)

1. **Requirements Phase**
   - Elicit Client Requirements
   - Understand client needs
2. **Analysis (specification) phase**
   - Analyze client requirements
   - Draft specification Documentation
   - Draft Software Project Management Plan
3. **Design phase**
   - Design architecture: Divide software functionality into components
   - Draft detailed design for each component
4. **Implementation phase**
   - Coding (development): Code and document each component
   - Unit test each individual component
   - Integration (system) testing: Combine components, test interfaces among components
   - 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.
   - Deploy to production environment
5. **Post delivery maintenance**
   - Maintain the software while it's being used to perform the tasks for which it was developed
6. **Retirement**
   - Product is removed from service: functionality provided by S/W is no longer useful / further maintenance is no longer economically feasible

♣

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

- Planning
- Testing
- Documentation

Solution.

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

> Remark
>
> Difference between Classical and Object Oriented paradigms
>
> **Classical paradigm** → One monolithic thing
>
> **Object Oriented Paradigm** → Many smaller classes that work together

Definition 1.1.2 (Corrective maintenance)

Removal of residual faults while software funcationality and specification remain relatively unchanged.

a.k.a fix production problems

♣

Definition 1.1.3 (Perfective Maintenance)

1. Implement changes the client thinks will improve effectiveness of the software product. (e.g. Additional functionality, reduce response time)
2. Specifications must be changed

♣

Definition 1.1.4 (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. Specifications may change to address the new environment

♣

> Important
>
> **The Importance of Post delivery Maintenance**
>
> - Shelf life of good software: 10, 20, even 30 years
> - Good software is a model of the real world, and the real world keeps changing, therefore software must change too
> - Cost of post delivery maintenance continues to go up, while cost of Implementation is nearly flat

Proposition 1.1.5 (Problems with the Classical Paradigms)

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

♠

Proposition 1.1.6 (Team Development Aspects)

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

♠

Proposition 1.1.7 (Ethical issues)

Software engineers commit to these ethical principles:

1. Public
2. Client and Employer
3. Product
4. Judgement
5. Management
6. Profession
7. Colleagues
8. Self

♠

2 Software Life Cycle Models

2.1 Iteration and Incrementation

Proposition 2.1.1 (Idealized Software Development)

$\varnothing \longrightarrow$ Requirements $\longrightarrow$ Analysis $\longrightarrow$ Design $\longrightarrow$ Implementation

♠

The *Classical model* is nost effective when the IT team can work without accepting change to the requirements after the requirements are complete.

Changing requirements negatively affects software:

- Quality
- Delivery dates
- Budget

Definition 2.1.2 (Moving Target Problem)

The **moving target problem** occurs when the requirements change while the sfotware is being developed

♣

Definition 2.1.3 (Scope Creep)

Scope creep or feature creep is a succession of small, almost trivial requests for additions to the requirements

♣

Definition 2.1.4 (Fault)

A **fault** is the observable result of a mistake made by any project staff member while working on any artifact

♣

Definition 2.1.5 (Regression Fault)

A **regression fault** occurs when a change in one part of the software product induces a fault in an apparently unrelated part of the software product

♣

Definition 2.1.6 (Regression Test)

A **regression test** provides evidence that we have not unintentionally changed something that we did not intend to change

♣

Definition 2.1.7 (Miller's Law)

Miller's Law states that, at any one time, a human is only capable of concentrating on approximately seven chunks of Integration

♣

Remark

Miller's Law Applied:

- Any large project will have much more then 7 components
- Hence we must start by working on ≤ 7 important components first temporarily ignoring the rest
- This technique is known as **stepwise refinement**