CS 430 - Lecture 03 - Iteration and Incrementation

Collin Roberts

September 14, 2023

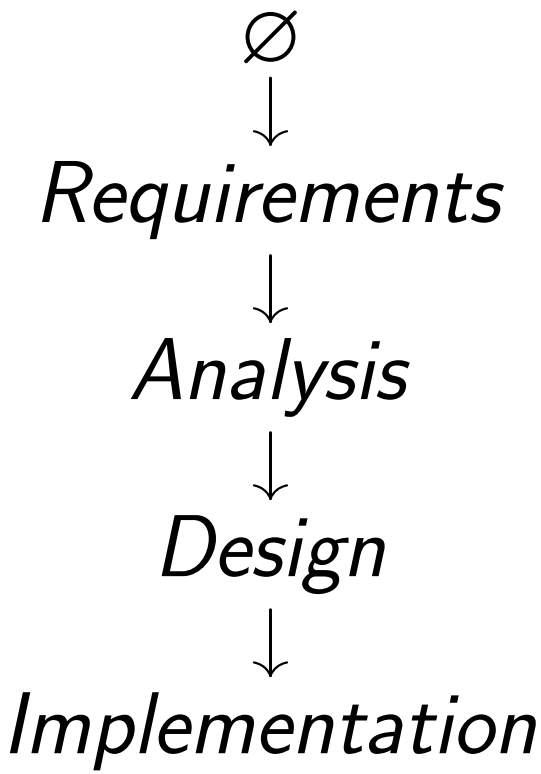
Outline

Introduction to Software Development Life-Cycle Models

Where Chapter 1 attempted to describe software development in the ideal world, Chapter 2 attempts to describe software development in the real world.

Software Development in Theory

Idealized Software Development



Software Development in Theory

In theory, we do not have to deal with any changes once the Requirements phase is complete.

Winburg Example

See the description in the text and in the examples for the course.

Winburg Example

Key Observations from the Example:

- Anecdote from a business line at a bank:
  IT was perceived as very slow to respond to requests for changes to their systems.
  In Lecture 02 we stated that the slowness of getting projects done using the Classical model was a drawback of that model.
  Corollary: IT resisted accepting changes to the requirements once the requirements were complete.

Winburg Example

- The Example Provides an Example of the Soft

1. Requirements were incomplete: there was no requirement describing performance (not meeting client's needs).
2. Assuming the project completes at the end of Episode 4, the project was
   1. very late, and
   2. over budget.
3. Nothing in the Example explicitly says that there were faults in the completed software product.

Winburg Example

- There was lots of rework, which was needed because each episode spawned a classical life-cycle effort (iteration), in which work done in one iteration had no easy way to feed into the next, if they overlapped in time.
- This was caused, in part, by the overall slowness of the Classical model.
- Starting to develop the single-precision fix before confirming it would provide the desired performance improvement was a waste of time.

Winburg Example

- Some re-use was achieved when the scanning software was packaged and re-sold.
- There was testing throughout the case.
- More testing throughout Episode 1 might have revealed the performance problems sooner.

Winburg Example

- **Instructor Remark:** Perhaps a small pilot project, prototyping the scanning hardware and software together would have revealed the performance problems earlier. This is a **proof of concept prototype**. We will discuss such prototypes again in Chapter 5.

Winburg Example

- Packaging and re-selling was a win.
- The project ultimately did satisfy the specification.
- Based on our own work experience to date, this is not the worst case we have seen so far. (The text agrees with us on this point.)

Winburg Example

Morals of the Example:

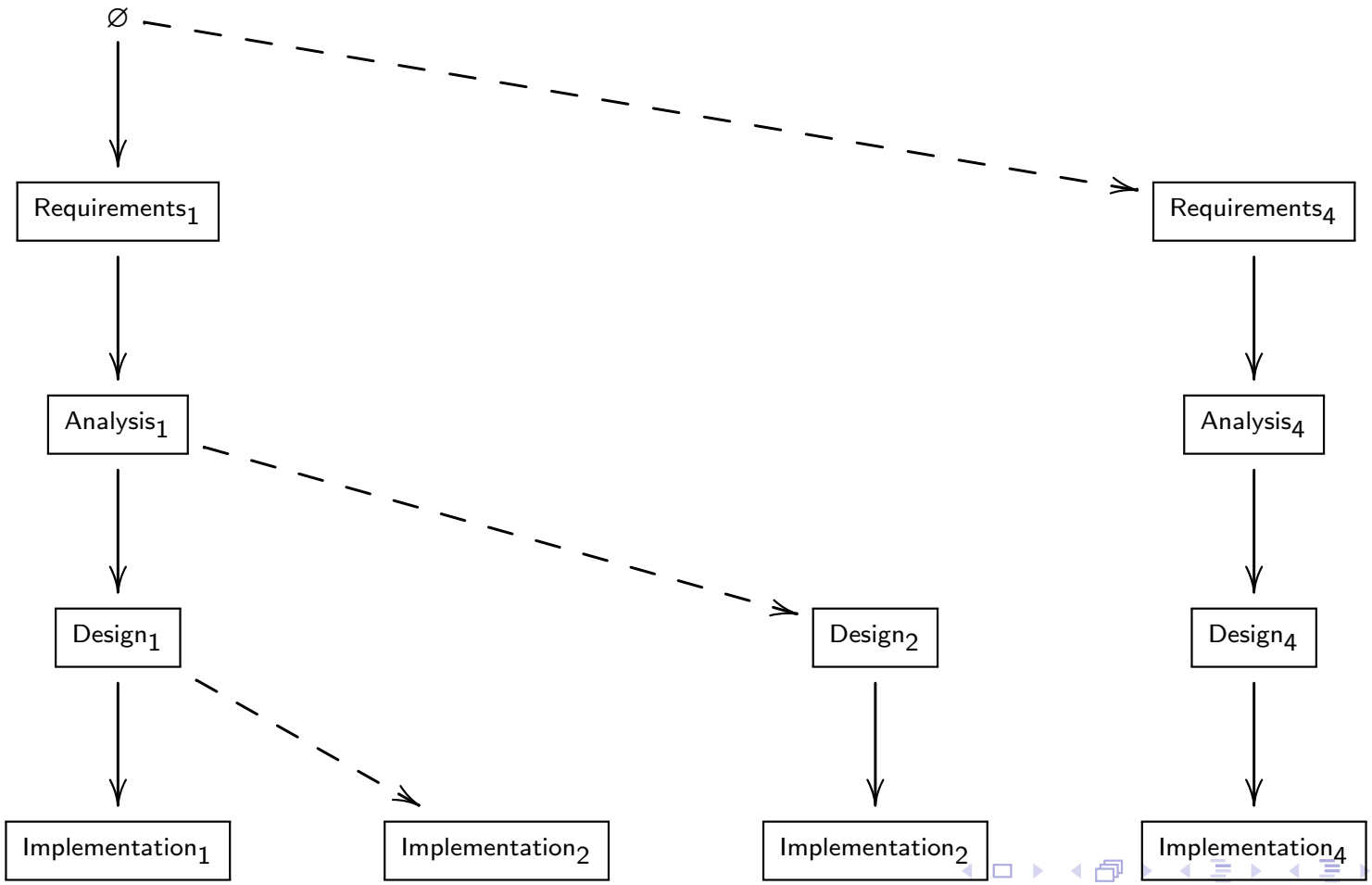
- The Classical model is most effective when the IT team can work without accepting changes to the requirements after the requirements are complete. Changes to requirements (e.g. adding the performance requirement, the Mayor's later change) negatively affects software quality, delivery dates, and budgets.

Winburg Example

- BUT in the real world, change is inevitable. We cannot prevent change; we must learn to manage it.

Winburg Example

Here is a sketch of Figure 2.2 in the text for an example of the **evolution-tree life cycle model** for this example, using the key:



Winburg Example

Key Idea: Each Episode spawns a new (sometimes partial) instance of the Classical development life-cycle model.

Winburg Example

- The rest of Chapter 2 is concerned with adapting the Classical life-cycle model to manage change.

Iteration and Incrementation

Key Idea: Think of Iteration and Incrementation as a generalization of the ad-hoc, **evolution tree** life-cycle from the Example. Break the project into (say 4) **increments**, then each increment runs as a small waterfall project. See Figures 2.4 through 2.6 in the text.

Iteration and Incrementation

Goals:

1. Get the benefits of Classical structure, while
2. Being more tolerant of change than the Classical model is.

Moving Target Problem

Useful Definitions:

Definition 1

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

Unfortunately this problem has no solution!

FeatureCreep

Definition 2

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

Remarks:

1. If the IT team can refuse such changes, then scope creep need not contribute to the moving target problem.
2. All too often the IT team does not have this power.

Fault

Definition 3

A **fault** *is the (observable) result of a coding mistake made by a programmer.*

Regression Fault

Definition 4

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.*

Regression Test

Definition 5

A **regression test** *provides evidence that we have not unintentionally changed something that we did not intend to change (i.e. that there are no regression faults).*

Regression Test

Typical Strategy:

1. Choose test cases that all fall under all the business rules **not** touched by the project specification.
2. Execute the production and the modified code against the chosen test cases.
3. Compare the outputs. **Success = no differences**.

Miller's Law

Definition 6

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

Miller's Law

Why this Matters for Software Engineering:

- One person can effectively work on at most seven items at once.
- Any software project of significant size will have many more than seven components.
- Hence we must start by working on ≤ 7 highly important things first, temporarily ignoring all the rest.

Miller's Law

- This is the technique of **stepwise refinement**.
- This technique will come up again in Chapter 5.

Conclusion

When you have time, you may enjoy listening to this YouTube video (the visual is just a static image) about Iteration & Incrementation:
`https://youtu.be/FTygpfEFFKw`

Conclusion

Next Time: (Almost) all the remaining life-cycle models are variations on Iteration and Incrementation.