CS 430 - Lecture 05 - The Unified Process I

Collin Roberts

September 21, 2023

Outline

Introduction to the Software Process

Definition 1

The **software process** *encompasses the activities, techniques and tools used to produce software.*

Introduction to the Software Process

Remarks:

1. With Definition 1, we could have defined the **software crisis** as our inability to manage the software process effectively.
2. The goal of Software Engineering is to improve the software process.

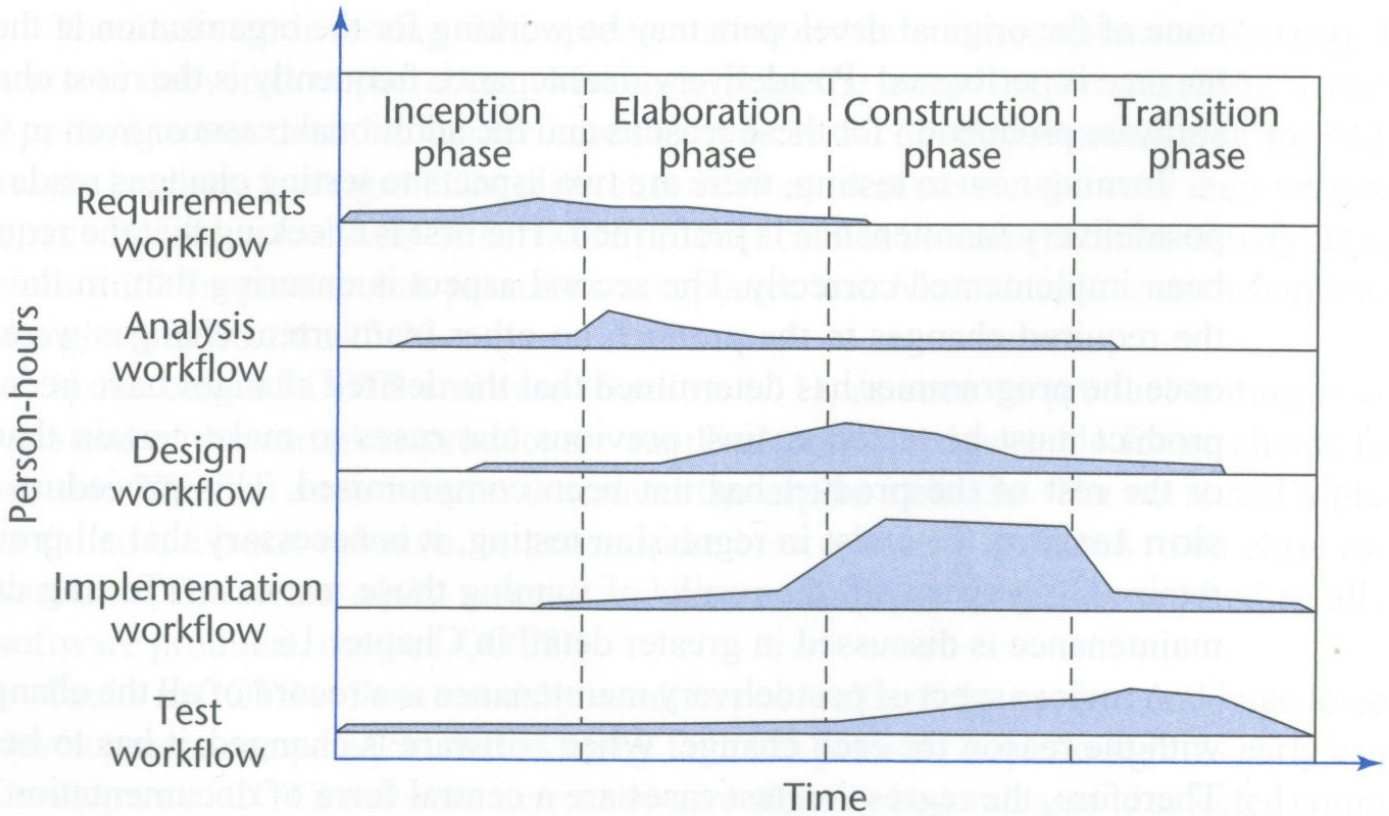
The Unified Process

1. **Idea:** We want to explore the **Unified Process**, which will be our software development methodology for the rest of the course. This methodology will be
   1. object-oriented, and
   2. extendable.

The Unified Process

2. This is Figure 3.1 on p88 of the text:

FIGURE 3.1
The core workflows and the phases of the Unified Process.



The Unified Process

3. The workflows have
   1. a **technical context**, e.g. the business case in the requirements workflow is technical, and
   2. a **task orientation**.
4. The phases have
   1. an **economic** context, e.g. the business case in the Inception phase is economic, and
   2. a **time orientation**.

The Unified Process

Definition 2

An **artifact** *is a work product from a workflow.*

The Unified Process

Questions From the Class

1. **Q:** Why are the artifacts tied to workflows, instead of to phases?
   A: Since it is more natural to think of the artifacts from a task point of view, it is more natural to tie the artifacts to the workflows than to the phases.

Iteration and Incrementation Within the Object-Oriented Paradigm

Key Idea: All variations on Iteration and Incrementation, including the Unified Process, attempt to preserve some Classical structure, while being more tolerant of change than the Classical model is. Under the **Unified Process**,

1. the phases are the increments, and
2. we iterate through the increments (each having a mini-Classical shape) to complete the project.

Iteration and Incrementation Within the Object-Oriented Paradigm

Definition 3

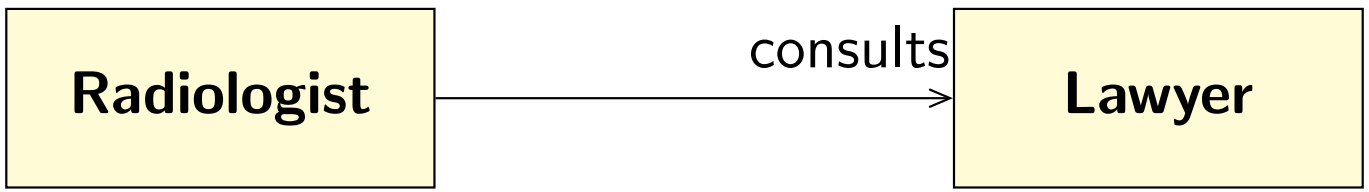
UML *stands for the* **Unified Modelling Language**.

Definition 4

A **model** *is a set of UML diagrams which describes one or more aspects of the software product to be developed.*

Iteration and Incrementation Within the Object-Oriented Paradigm

Example:



"The Radiologist class consults the Lawyer class."

Iteration and Incrementation Within the Object-Oriented Paradigm

Motivation:

1. Even the best software engineers almost never get their artifacts right on the first attempt. So **stepwise refinement** will be needed.
2. UML diagrams are **visual**, hence more intuitive than a block of verbiage. "A picture is worth a thousand words."
3. The visual nature of a UML model fosters collaborative **refinement**.

Iteration and Incrementation Within the Object-Oriented Paradigm

Remarks:

1. Presenting the entire Unified Process would take more time and space than we have during the remainder of this term, hence we will stick to the highlights.

Iteration and Incrementation Within the Object-Oriented Paradigm

2. The names of the workflows (mostly) match the names of the phases of the classical model. The descriptions of the workflow artifacts that follow are similar to the outputs of the corresponding classical phases.
3. The classical model tied tasks and time together in sequence. The unified model separates tasks and time.

Iteration and Incrementation Within the Object-Oriented Paradigm

Summary of Requirements, Analysis, Design and Implementation Workflows

1. Each workflow corresponds (task-based) with the Classical phase having the same name.
2. See the notes below for full details.

Requirements Workflow

Goal: Determine the client's needs, and determine what constraints there are (often referred to as **concept exploration**).

Requirements Workflow

Pitfalls: Do the Lecture 05 Example Here.

Requirements Workflow

Problems Found With Requirements Given in Example

1. Standings Changes: do we actually want all increases, all decreases, or both?
2. Standing Display: If we don't include previous, then changes will not be clear from the report
3. Some students with no changes in their standings should be included (e.g. if they are still on probation)
4. MAV used for criteria, but only CAV is displayed on the report - unclear
5. Conflicting sort criteria in different parts of the specification
6. Missing criterion for filtering down to just the program for which I am responsible
7. Should the two inclusion criteria be combined with `AND` or with `OR`?
8. And possibly more...

Requirements Workflow

Moral: To summarize the Example, requirements **artifacts** can be

1. incorrect (only the client can detect this)
2. ambiguous (e.g. `AND` versus `OR` in the inclusion criteria - IT can detect this)
3. incomplete (e.g. missing criterion to filter down to the program - only the client can detect this)
4. contradictory (e.g. conflicting sort criteria - IT can detect this)

Requirements Workflow

Using UML diagrams correctly helps to mitigate the above problems with requirements.

Analysis Workflow

1. **Goal:** Analyze and refine the requirements to achieve the level of detail needed to begin designing the software, and to maintain it effectively later.
2. Once the analysis is complete, the cost and duration of the development are estimated → create the Software Project Management Plan (SPMP).
3. Terminology: **deliverables**, **milestones** and **budget**.

Design Workflow

1. **Goal:** Show **how** the product is to do what it must do.
2. More precisely, refine the artifacts of the analysis workflow until the result is good enough for the developers to implement it.
3. There are differences between the classical and the object-oriented paradigms here.
4. It is important to keep detailed records about design decisions.

Implementation Workflow

1. **Goal:** Implement the target software product in the chosen implementation language.
2. Usually code artifacts are implemented by different developers, and integrated once implemented - thus design shortcomings may not come to light until the time of integration.

Test Workflow

Goal: Ensure the correctness of the artifacts from the other workflows.

Requirements

1. Key Idea: **traceability:** every later artifact must trace back to a requirement artifact.
2. Key Observation: Until Implementation, there will be no code to test, only documents. Hence we test by holding a **review** of the document, with the key stakeholders. We will delve deeper into this in Chapter 6.

Analysis

1. Tactic: Hold a **review** of analysis artifacts with the key stakeholders, chaired by SQA.
2. Review the SPMP too.

Design

1. Again, design artifacts must trace back to analysis artifacts.
2. Tactic: Again, hold a **review** of design artifacts (likely without the client this time)

Implementation

Remarks:

1. This will be explained in detail in Chapter 6.

Implementation

The testing must include

1. desk checking (programmer)
2. unit testing (SQA)
3. integration testing (SQA)
4. product testing (SQA)
5. (user) acceptance testing (SQA and client)

Implementation

Remarks:

1. Some projects also incorporate **alpha** and **beta** testing (usually the beta version is the first version that the public would see).
2. Although it is tempting, **alpha** testing should **not** replace thorough testing by the SQA group.

Post-Delivery Maintenance

1. This is **not** an afterthought - it must be planned from the start.
2. **Pitfall:** lack of adequate documentation (deadline pressures during initial delivery contribute to this)
3. Testing of changes must include **positive** and **regression** testing.

Post-Delivery Maintenance

Definition 5

Positive testing *means testing that what you intended to change was changed in the desired way.*

Post-Delivery Maintenance

Strategy:

1. Select test cases exercising the changed business rules.
2. Compare pass 0 (no changes) against pass 1 (with changes).
3. Confirm that the pass 1 output has the desired changes applied.

Retirement

1. This is triggered when post-delivery maintenance is no longer feasible or cost-effective.
2. Usually a software product is **replaced** at this point. The software product must be replaced if the business need persists.
3. True **retirements** are rare.