
Practical work presentation
Software engineering discipline
Software development process
THIS SESSION GOAL

Theme 2

SOFTWARE DEVELOPMENT PROCESS

Introduction

General goals:

- Learn the detailed activities involved in software development projects.
- Learn about the specific methodology we will be using in this subject

Before we start, some reflections…

- There is a well-known paper from 1986 titled **"No Silver Bullet: Essence and accidents of software engineering" by Dr. Frederick Brooks**, also published in his classic book Mythical Man-Month (20th anniversary edition). Both recommended readings! In essence, Brooks states that in software development ***there exist "werewolves"*** and that **we won't have silver bullets**:

"there is no single development, in either technology or management technique, which by itself promises even one-order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity"

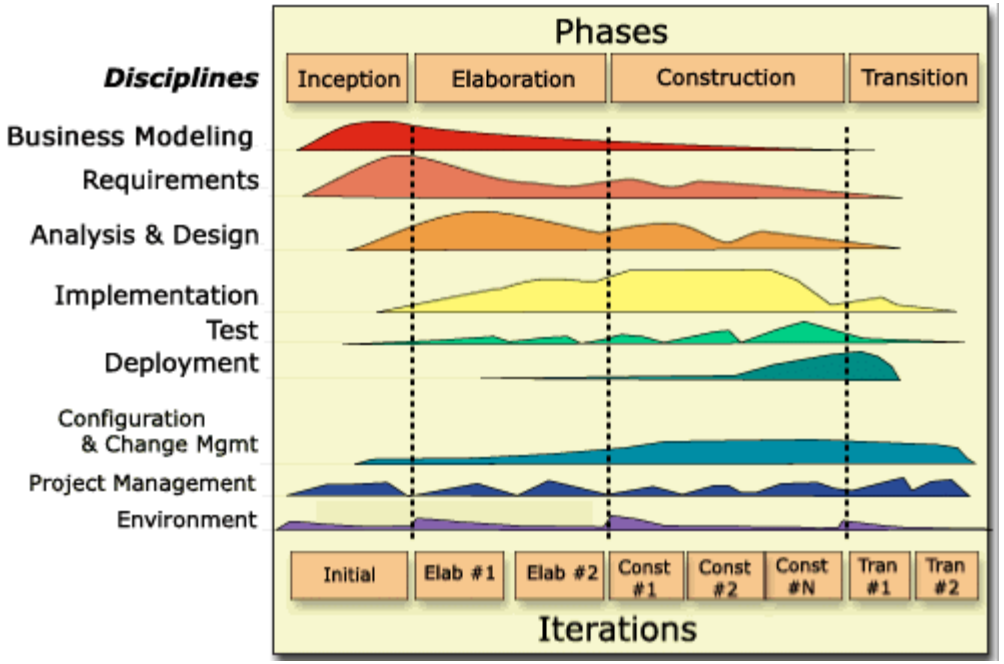
- Now after > 35 years:
  - What do you think about this statement? Is it still true, or do you identify a *"silver bullet"* that invalidates the statement?
  - The werewolves in 1986 was succinctly the "unpredictable emergent behavior in complex systems". Do you think the werewolves in software development remain the same in 2023?
  - As opposed to hardware and the famously "flawless" Moore's Law, why is software development inherently complex when trying to get better, cheaper, more productive products/people?
  - Do you think that Chatgpt and the new AI generation related bots can possibly be the *"silver bullet"*?
  - Is building software systems like building large construction projects such as buildings, bridges and skycrapers?

…take a time to think about these questions. We will revisit later…

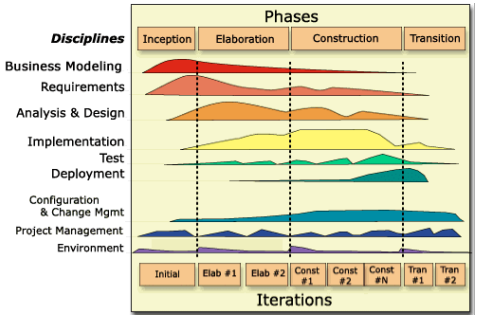
Introduction

- During the practical work we **will be using a variant of the UP methodology in an agile OO "flavor"** with a stage configuration 1-1-2-1

- Let's learn more about the **processes involved in UP and the Craig-Larman variant**



The Unified Process

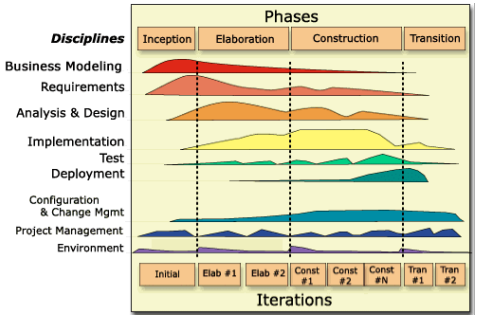


- The ***Unified Software Development Process*** or Unified Process is an **iterative and incremental software development process** framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP) which we will refer to.

- The Unified Process is not simply a process, but rather an **extensible framework which should be customized** for specific organizations or projects.

- The first book to describe the process was titled **The Unified Software Development Process** (ISBN 0-201-57169-2) and published in 1999 by Ivar Jacobson, Grady Booch and James Rumbaugh.

The Unified Process RUP

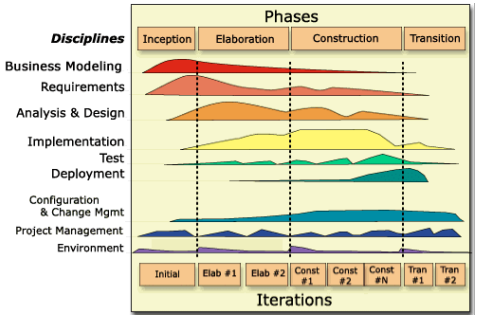


- The RUP has determined a project life-cycle consisting of **four phases**. These phases allow the process to be presented at a high level in a similar way to how a 'waterfall'-styled project might be presented, although **in essence the key to the process lies in the iterations of development** that lie within all of the phases.
- Each **phase has one key objective** and milestone at the end that denotes the objective being accomplished.

- Within each **phase iteration**, the tasks are categorized into **nine disciplines**:

Engineering disciplines	Supporting disciplines
Business modelling	**Configuration and Change Management**
Requirements	**Project Management**
Analysis and Design	Environment
Implementation	
Test	
Deployment	

The Unified Process
RUP. Phases & Disciplines



1. Inception phase

The primary objective is to **scope the system adequately as a basis for validating initial costing and budgets**. In this phase the **business case** which includes business context, success factors and financial forecast is established. To complement the business case, a **basic use case model, project plan, initial risk assessment and project description** (the core project requirements, constraints and key features) are generated.

2. Elaboration phase

The primary objective is to **mitigate the key risk items identified by analysis** up to the end of this phase. The elaboration phase is where the project starts to take shape. In this phase the problem domain analysis is made and the architecture of the project gets its basic form.

The outcome of the elaboration phase is:

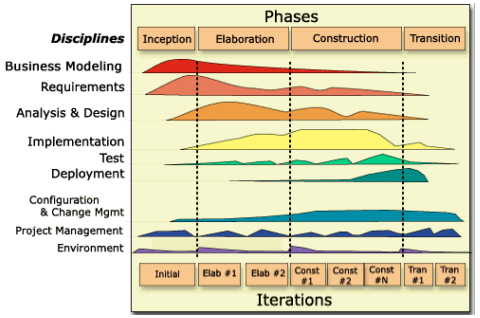
- A **use-case model** in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. (80%)
- A **description of the software architecture** in a software system development process.
- Business case and risk list which are **revised**.
- A **development plan** for the overall project.
- **Prototypes** that demonstrably mitigate each identified technical risk.

3. Construction phase

The primary objective is to **build the software system**. In this phase, the main focus is on the development of components and other features of the system. This is the phase when the bulk of the coding takes place. In larger projects, several construction iterations may be developed in an effort to divide the use cases into manageable segments to produce demonstrable prototypes.

4. Transition phase. That we will call deployment, in which the complete system is installed in the house of our client.

The Unified Process RUP. Best Practices

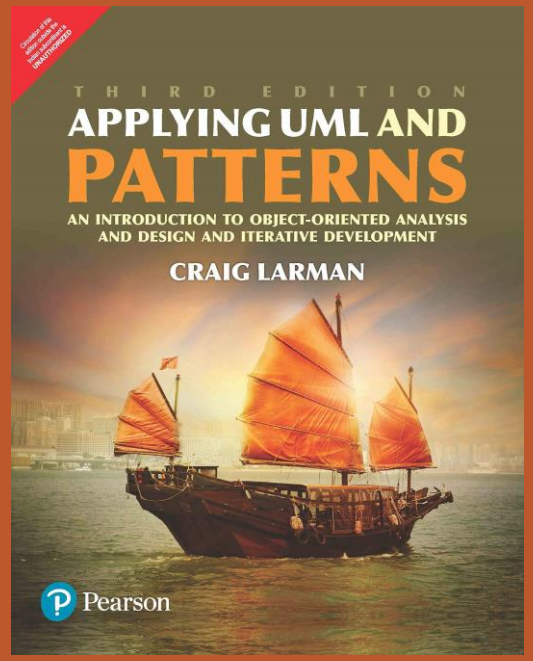


- Six **best software engineering practices (*)** are defined for software projects in RUP to minimize faults and increase productivity:

1. **Develop iteratively.** It is best to know all requirements in advance; however, often this is not the case. Several software development processes exist that deal with providing solutions to minimize cost in terms of development phases.
2. **Manage requirements.** Always keep in mind the requirements set by users.
3. **Use components.** Breaking down an advanced project is not only suggested but in fact unavoidable. This promotes ability to test individual components before they are integrated into a larger system. Also, code reuse is a big plus and can be accomplished more easily using object-oriented programming.
4. **Model visually.** Use diagrams to represent all major components, users, and their interaction. "UML", short for Unified Modeling Language, is one tool that can be used to make this task more feasible.
5. **Verify quality.** Always make testing a major part of the project at any point of time. Testing becomes heavier as the project progresses but should be a constant factor in any software product creation.
6. **Control changes.** Many projects are created by many teams, sometimes in various locations, different platforms may be used, etc. As a result, it is essential to make sure that changes made to a system are synchronized and verified constantly. (See Continuous integration).

(*) Stephen Schach (2004). Classical and Object-Oriented Software Engineering. 6/e, WCB McGraw Hill, New York, 2004.
(*)Rational Unified Process white paper Archived 2009-05-01 at the Wayback Machine

Development Process Defined by Craig Larman



Given many possible activities from requirements through to implementation, **how should a team proceed?**

The method defined by Craig Larman, is a reduced version of one of the process models described in the Unified Process UP.

Study material → Object Oriented Development With UML

"Requirements analysis and OOA/D needs to be presented and practiced in the context of some development process. In this case, an **agile (light, flexible)** approach to the well-known Unified Process (UP) is used as the sample **iterative and incremental development process** within which these topics are introduced.

However, the analysis and design topics that are covered are common to many approaches and learning them in the context of an agile UP **does not invalidate their applicability to other methods**, such as Scrum, Feature-Driven Development, Lean Development, Crystal Methods, and so on."

Craig Larman Method Characteristics … … and our practical work

- Evolutionary (iterative)
- Incremental
  - Less time until you have something working
  - Contrastable with the client / user
- Driven by Use Cases
  - Develop the software from the inside out
  - Agile Modeling

Deliverable	Subject Themes	Dossier sections	Deadline
D1	Theme 3 Offer and Budget	1, 2, 3	February 24th 23:59 No presentation required
D2	Themes 6, 7. SCM and SQA	4, 5	March 3rd 23:59 No presentation required
D3	Theme 4 Feasibility analysis	Full 8.1, 10, 11	March 10th 23:59 No presentation required
D4	Theme 5 Use cases	8.2, 8.3, 8.4, 10, 11	March 31st 23:59 No presentation required
D5	Theme 8 Estimation	6, 10, 11	April 14th 23:59 No presentation required
D6	Theme 9 Planning	7, 10, 11	April 14th 23:59 Presentation required on April 25th and 26th
D7	Themes 10, 11 Analysis & Design	9.1.1, 9.2.1, 10, 11	May 5th 23:59 Whole final delivery
D8	Themes 10, 11 Analysis & Design	9.1.2, 9.2.2, 10, 11, 12, 13	Presentation required on May 9th and 10th

Craig Larman Method Characteristics … … and our practical work

- Evolutionary (iterative)
- Incremental
  - Less time until you have something working
  - Contrastable with the client / user
- Driven by Use Cases
  - Develop the software from the inside out
  - Agile Modeling

Deliverable	Subject Themes	Dossier sections	Deadline
D1	Theme 3 Offer and Budget	1, 2, 3	February 24th 23:59 No presentation required
D2	Themes 6, 7. SCM and SQA	4, 5	March 3rd 23:59 No presentation required
D3	Theme 4 Feasibility analysis	Full 8.1, 10, 11	March 10th 23:59 No presentation required
D4	Theme 5 Use cases	8.2, 8.3, 8.4, 10, 11	March 31st 23:59 No presentation required
D5	Theme 8 Estimation	6, 10, 11	April 14th 23:59 No presentation required
D6	Theme 9 Planning	7, 10, 11	April 14th 23:59 Presentation required on April 25th and 26th
D7	Themes 10, 11 Analysis & Design	9.1.1, 9.2.1, 10, 11	May 5th 23:59 Whole final delivery
D8	Themes 10, 11 Analysis & Design	9.1.2, 9.2.2, 10, 11, 12, 13	Presentation required on May 9th and 10th

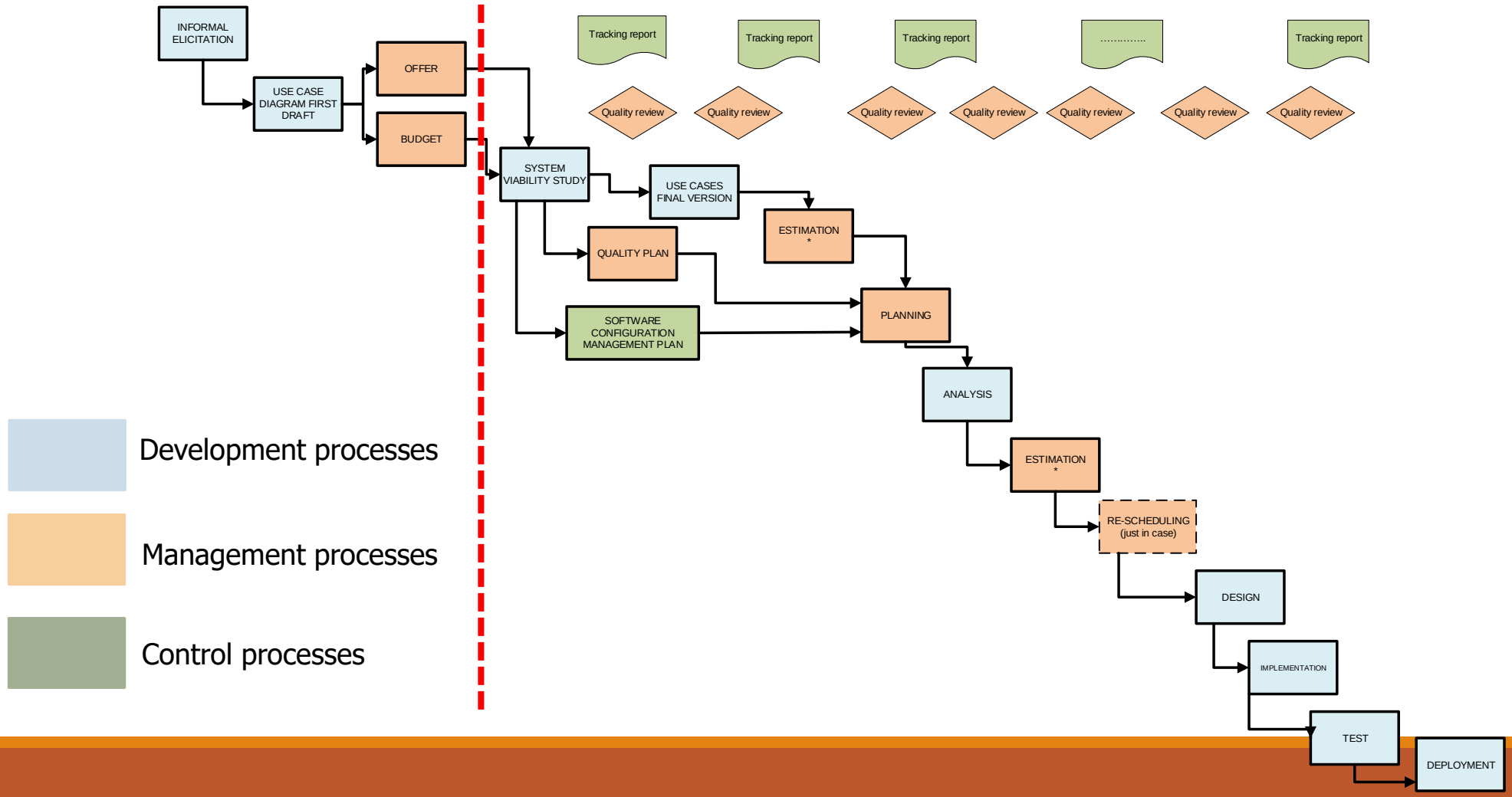
Craig Larman Method Stages – Engineering view

Planning and requirements specification phase (1 time)	Construction phase (n times)	Installation phase (1 time)

• Feasibility analysis (including requirements specification)
• Use case model
• Prioritization of use cases
• Definition of high-level use cases
	• **ITERATION 1**
	  ○ **ANALYSIS**
	  Use cases in expanded format
	  Conceptual model
	  Operation Contracts
	  ○ **DESIGN**
	  Class diagram
	  Sequence Diagrams
	  State Transition Diagrams
	  ○ **CODING**
	  ○ **TESTING**
	• **ITERATION 2 (idem)**
	• **……**
	• **ITERATION n (idem)**
		• Deployment

 Planning, requirements definition, prototype construction…	 Construction of the system itself	 Starting up the system in the environment of use

Craig Larman Method
Stages – Engineering + Management view



Craig Larman Method
Stages – Engineering + Management view

Phase 0

Not in Larman model

- First draft of the use case model (if you want to make an estimation based on use case points)
- Offer and Budget
- Quality Plan
- SCM Plan
- SCM Plan Review
- Estimation
- Estimation review
- Schedule
- Schedule review

Planning and requirements specification phase

(1 time)

- Feasibility analysis (including requirements specification)
- Feasibility review
- Use cases model
- Use cases model review
- Definition of high-level use cases.
- Definition of high-level use cases review
- Prioritization of use cases
- Prioritization of use cases review.

Construction Phase

(n times)

- Iteration 1
  - ANALYSIS

  Use cases in extended format

  Use cases in extended format review

  Conceptual model

  Conceptual model review

  Operation Contracts

  Operation Contracts review
  - DESIGN

  Class diagram

  Class diagram review

  Sequence diagrams

  Sequence diagrams review

  Transition states diagram

  Transition states diagrams review
  - CODING
  - TESTING
- Iteration 2 (idem)
- ……
- Iteration n (idem)

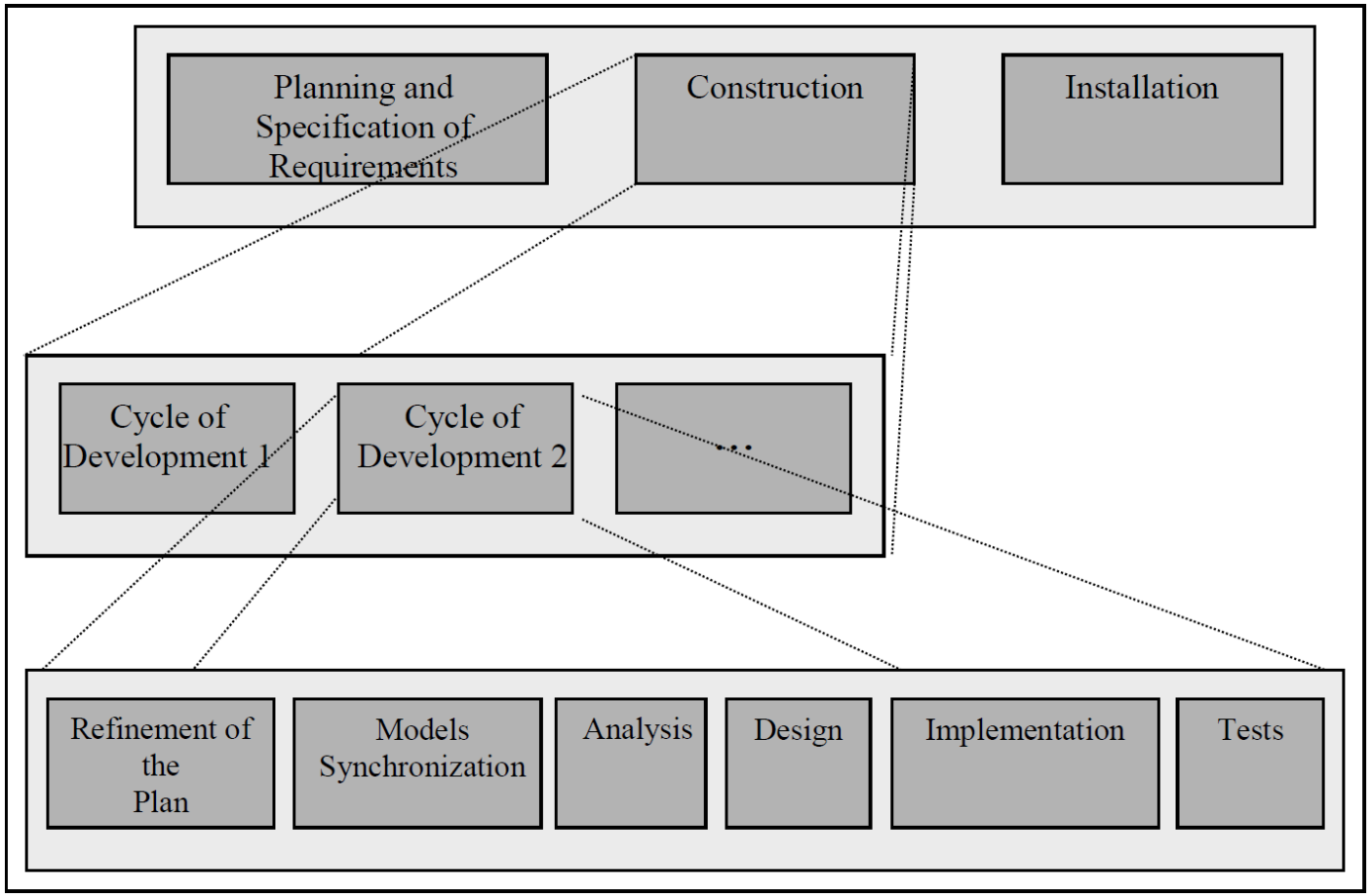
Installation phase

(1 time)

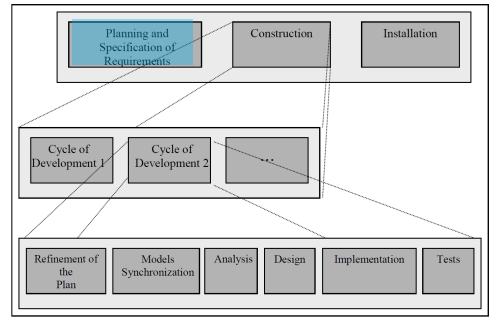
- Deployment

- IN RED: All the revisions are derived from the quality plan. And they will be the ones that appear in that quality plan.
- IN BLUE: Management and control processes

Craig Larman Method
Stages – Acknowledge Iterative Development



Craig Larman Method
Phase: Planning and Requirements Specification

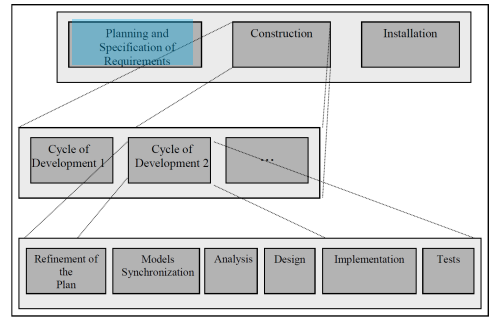


- Activities:
  1. Defining the Plan-Draft
  2. Create Preliminary Research Report
  3. *Defining the Requirements*
  4. Register Glossary Terms
  5. Implement Prototype
  6. *Define Use Cases and Prioritize*
  7. *Defining the Conceptual Model - Draft*
  8. *Defining System Architecture - Draft*
  9. *Refining the Plan*

- Description of **needs or aspirations for a product**. Goal is **to identify what is really needed**. Format is not defined by UML

- **Recommended minimum content** of the requirements document:
  - Purpose
  - Scope and Users
  - Functional requirements:
    - ReqF1: .....
  - Non functional requirements
    - Quality: ReqC1:....
    - Security: ReqS1:...
    - Performance: ReqR1:...

Craig Larman Method
Phase: Planning and Requirements Specification

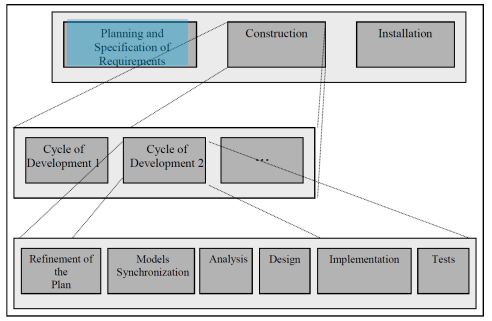


- Activities:
  1. Defining the Plan-Draft
  2. Create Preliminary Research Report
  3. *Defining the Requirements*
  4. Register Glossary Terms
  5. Implement Prototype
  6. *Define Use Cases and Prioritize*
  7. *Defining the Conceptual Model - Draft*
  8. *Defining System Architecture - Draft*
  9. *Refining the Plan*

- Narrative document **that describes the sequence of events** of an actor who uses a system to complete a process. Story or **particular way of using a system**.
- Use cases **are not exactly requirements** or functional specifications, but illustrate and imply requirements in the stories they tell.
- They are **represented in the UML notation** that only defines how to represent the relationship between actors and use cases in a diagram (Use Case Diagram).
- At first it is interesting to **approach a use case from a high abstraction level**, it is what is called High Level and Essential Use Cases.

We will see activities in detail in Theme 5

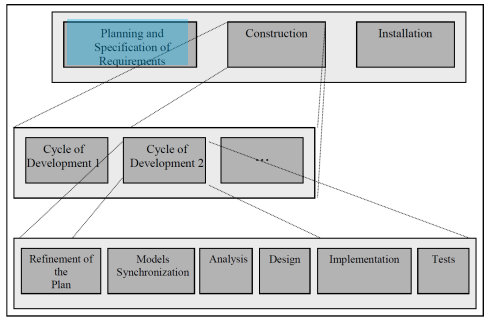
Craig Larman Method
Phase: Planning and Requirements Specification



- Activities:
  1. Defining the Plan-Draft
  2. Create Preliminary Research Report
  3. *Defining the Requirements*
  4. Register Glossary Terms
  5. Implement Prototype
  6. *Define Use Cases and Prioritize*
  7. *Defining the Conceptual Model - Draft*
  8. *Defining System Architecture - Draft*
  9. *Refining the Plan*

- Part of the problem domain research is to **identify the concepts that make up the problem**. In order to represent these concepts, a **Static Structure Diagram of UML will be used**, which will be called Conceptual Model.
- In the Conceptual Model you have a **representation of real-world concepts**, not of software components.
- The objective of creating a Conceptual Model is to **increase understanding of the problem**.
- **Steps**:
  - Concept Identification
  - Creation of the Conceptual Model
  - Identification of associations
  - Attribute Identification

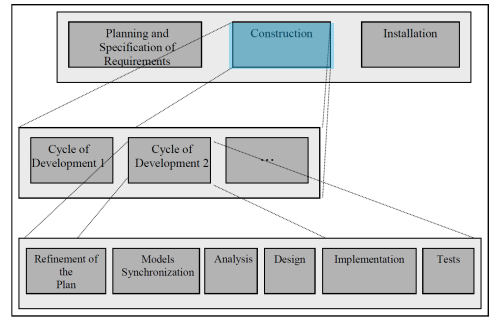
Craig Larman Method
Phase: Planning and Requirements Specification



- Activities:
  1. Defining the Plan-Draft
  2. Create Preliminary Research Report
  3. *Defining the Requirements*
  4. Register Glossary Terms
  5. Implement Prototype
  6. *Define Use Cases and Prioritize*
  7. *Defining the Conceptual Model - Draft*
  8. *Defining System Architecture - Draft*
  9. *Refining the Plan*

- The so-called **"Three-Level Architecture"** is the most common in information systems that, in addition to having a user interface, contemplate the persistence of data.
- A description of the three levels would be as follows:
  - **Level 1:** Presentation - windows, reports, etc.
  - **Level 2:** Application Logic - tasks and rules that govern the process.
  - **Level 3:** Storage - storage mechanism
- The three-level architecture **can be called Multi-Level** if we take into account the fact that all the levels of the three-level architecture can be decomposed each of them more and more.

Craig Larman Method Phase: Construction

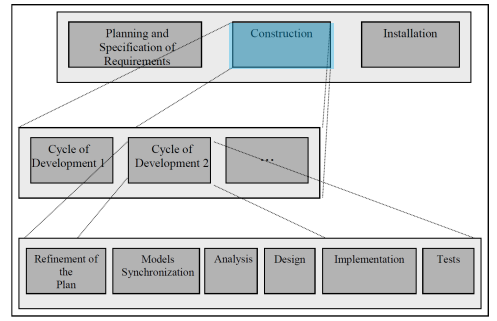


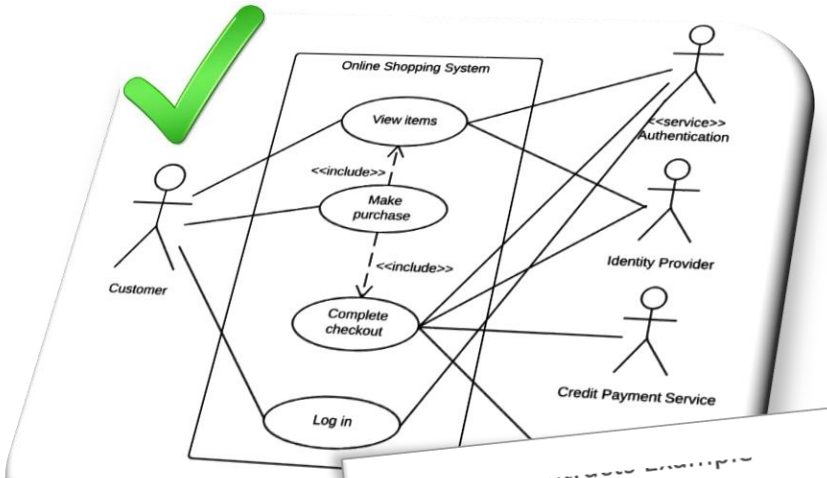
- Activities:
  1. Refinement of the Plan
  2. Model Synchronization
  3. *Analysis*
  4. *Design*
  5. *Implementation*
  6. *Tests*

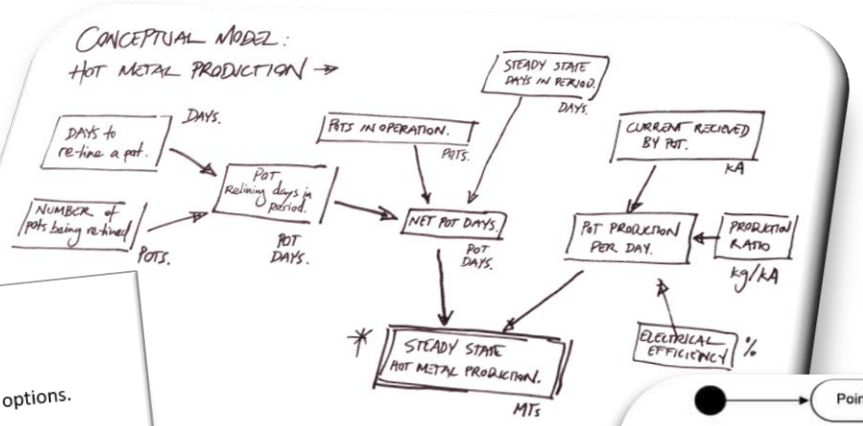
- **Research on the domain of the problem**, limited only to the use cases treated in the present cycle.
- It tries to **reach a good understanding** of the problem.
- It's **focused on the what** rather than the how.
- Activities:
  1. **Define Use Cases in Expanded Format**
  2. **Refine Usage Cases Diagram**
  3. **Refining the Conceptual Model**
  4. **Refining the Glossary**
  5. **Defining System Sequence Diagrams**
  6. **Define Operation Contracts**
  7. **Define State Diagrams**

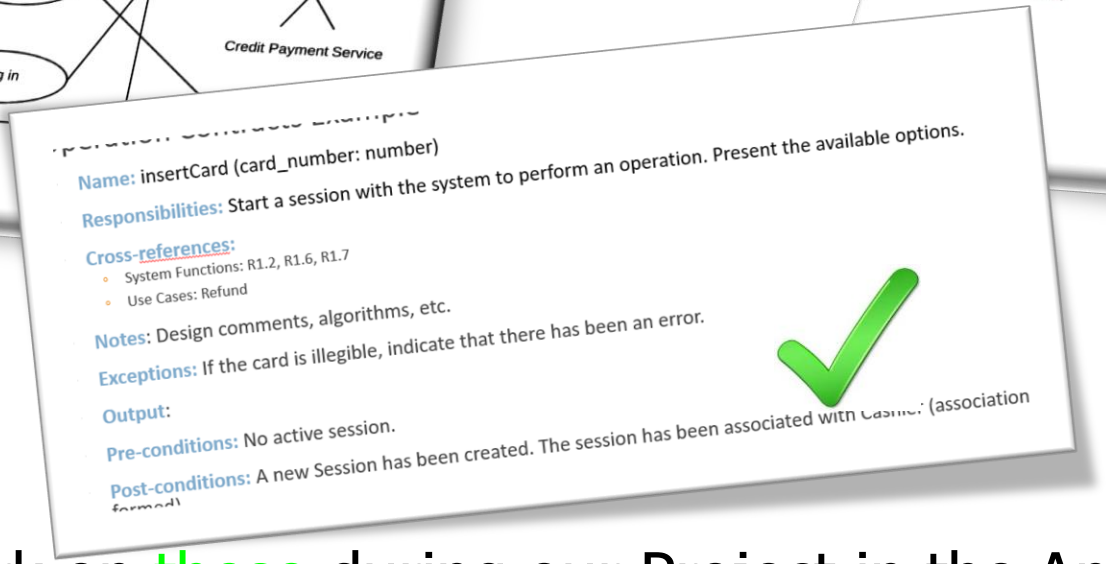
We will see activities in detail in Theme 10

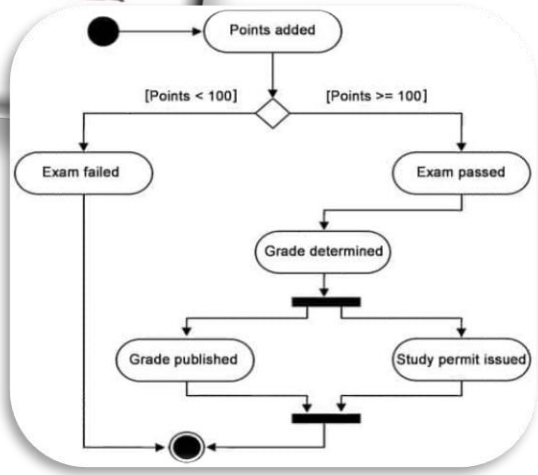
Craig Larman Method
Phase: Construction - Analysis





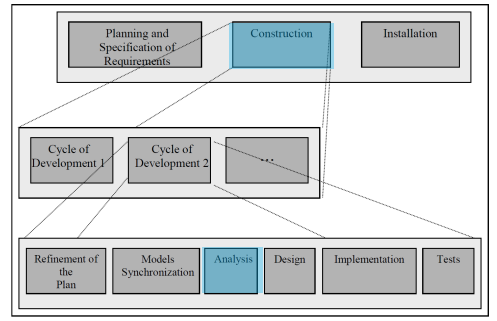






We will work on these during our Project in the Analysis phases

Craig Larman Method
Phase: Construction - Analysis



- Operation Contracts

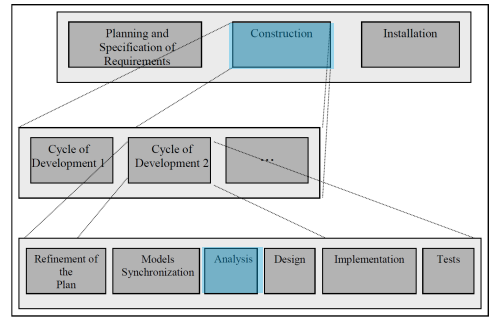
Once the System Operations are identified, the expected behaviour of the system in each operation is described by means of contracts.

A Contract is a **document that describes what is expected** of an operation. It has a declarative style of writing, emphasizing the ***what* rather than the *how***. The most common is to express contracts in the form of pre- and post-conditions around state changes.

A contract can be written for an individual method of a software class, or for an entire system operation. **At this point you will see only the latter case.**

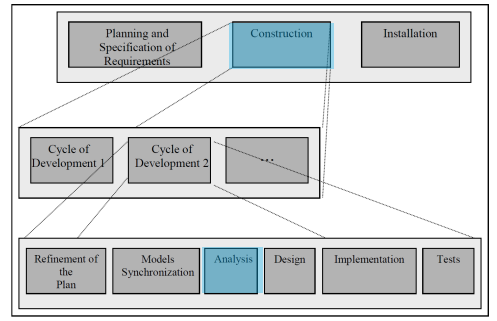
A System Operation Contract **describes changes in system status** when a system operation is invoked.

Craig Larman Method
Phase: Construction - Analysis



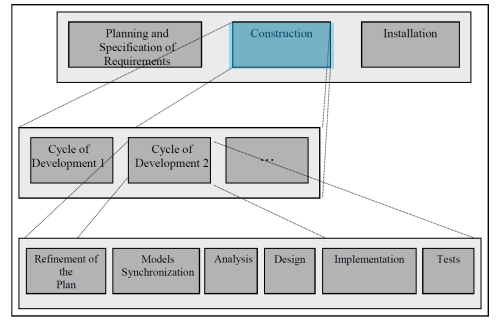
- Operation Contracts
  - **Name**: Name of the operation and parameters.
  - **Responsibilities**: An informal description of the responsibilities that the operation should perform.
  - **Cross-references**: functional requirements, use cases, etc.
  - **Notes**: Design comments, algorithms, etc.
  - **Exceptions**: Exceptional cases. Situations that we should be aware of can happen. It also indicates what is done when the exception occurs.
  - **Output**: Outputs that do not correspond to the user interface, such as messages or logs that are sent outside the system. (In most operations of the system this section remains empty)
  - **Pre-conditions**: Assumptions about system status before running the operation. Something we don't take into account that can happen when this system operation is called.
  - **Post-conditions**: The status of the system after the operation is completed.

Craig Larman Method
Phase: Construction - Analysis



- Operation Contracts Example
  - **Name:** insertCard (card_number: number)
  - **Responsibilities:** Start a session with the system to perform an operation. Present the available options.
  - **Cross-references:**
    - System Functions: R1.2, R1.6, R1.7
    - Use Cases: Refund
  - **Notes**:
  - **Exceptions:** If the card is illegible, indicate that there has been an error.
  - **Output**:
  - **Pre-conditions:** No active session.
  - **Post-conditions:** A new Session has been created. The session has been associated with Cashier (association formed).

Craig Larman Method Phase: Construction

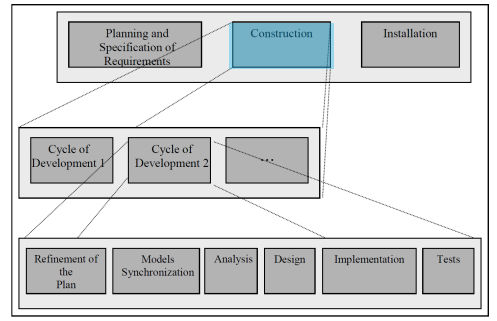


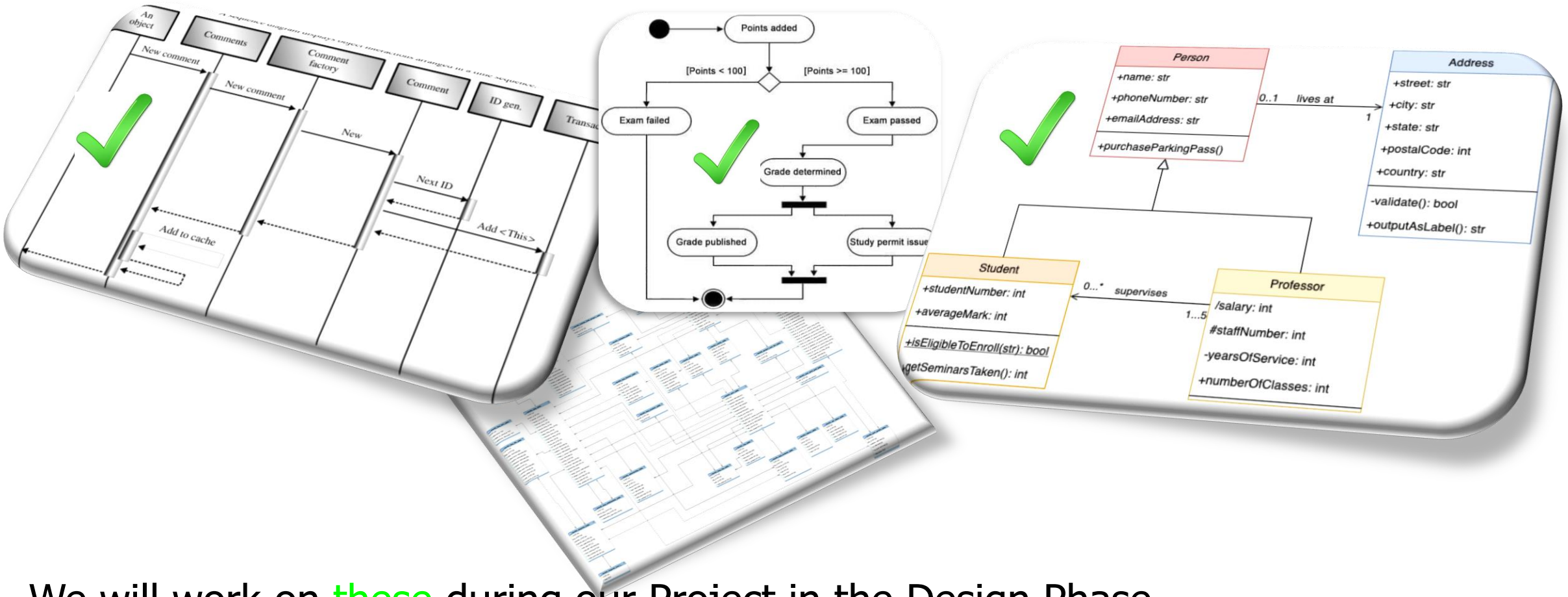
- Activities:
  1. Refinement of the Plan
  2. Model Synchronization
  3. *Analysis*
  4. *Design*
  5. *Implementation*
  6. *Tests*

- **Solution at the logical level** to meet the requirements.
- It is designed **how the system will behave** to perform the functions requested of it.
- We work on a **level very close to the software.**
- Activities:
  1. **Define Actual Use Cases**
  2. **Define Reports and User Interface**
  3. **Refining System Architecture**
  4. **Defining Interaction Diagrams**
  5. **Define the Design Class Diagram**
  6. **Defining the Database Schema**

We will see activities in detail in Theme 10

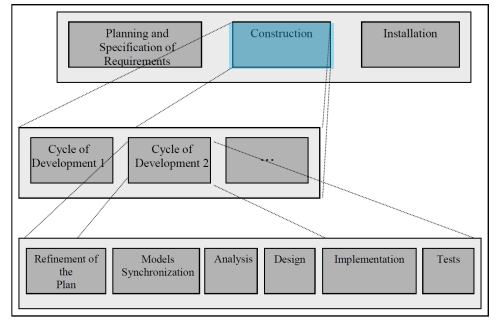
Craig Larman Method
Phase: Construction - Design





We will work on these during our Project in the Design Phase

Craig Larman Method Phase: Construction

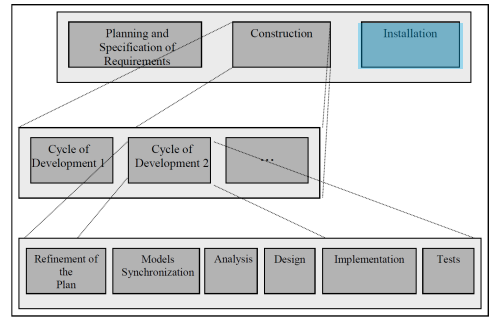


- Activities:
  1. Refinement of the Plan
  2. Model Synchronization
  3. *Analysis*
  4. *Design*
  5. *Implementation*
  6. *Tests*

- What was specified in the **Design phase is translated into a programming language.**
- The elements that make up the implementation are **shown with the diagrams:**
  - **Component diagram.**
  - **Deployment (or Implementation) Diagram.**

- The obtained program is **debugged and tested**, and you already have a working part of the system that can be tested with future users, and even **put into production if a gradual installation is planned.**
- Once you have a stable version, you **go on to the next development cycle to increase the system with the use cases assigned to that cycle.**

Craig Larman Method Phase: Installation



- Installation phase is not taken into account in this subject

- [x] Software engineering discipline
- [x] Practical work presentation
- [x] Software development process

THIS SESSION GOAL

DOUBTS?