UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4351 Fall 2023

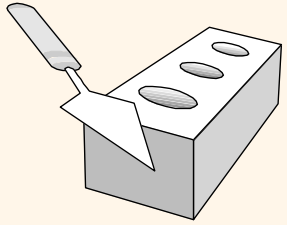
Software Engineering

M & W 4 to 5:30 PM

Prof. Victoria Hilford

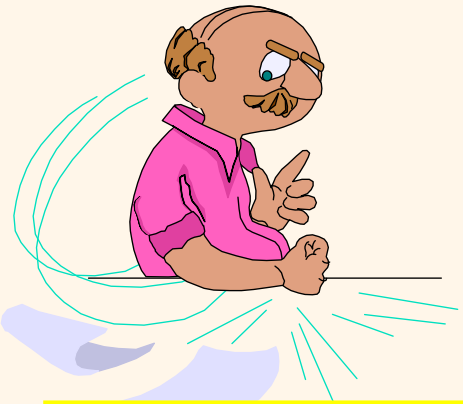
PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4351

4 to 5:30



PLEASE

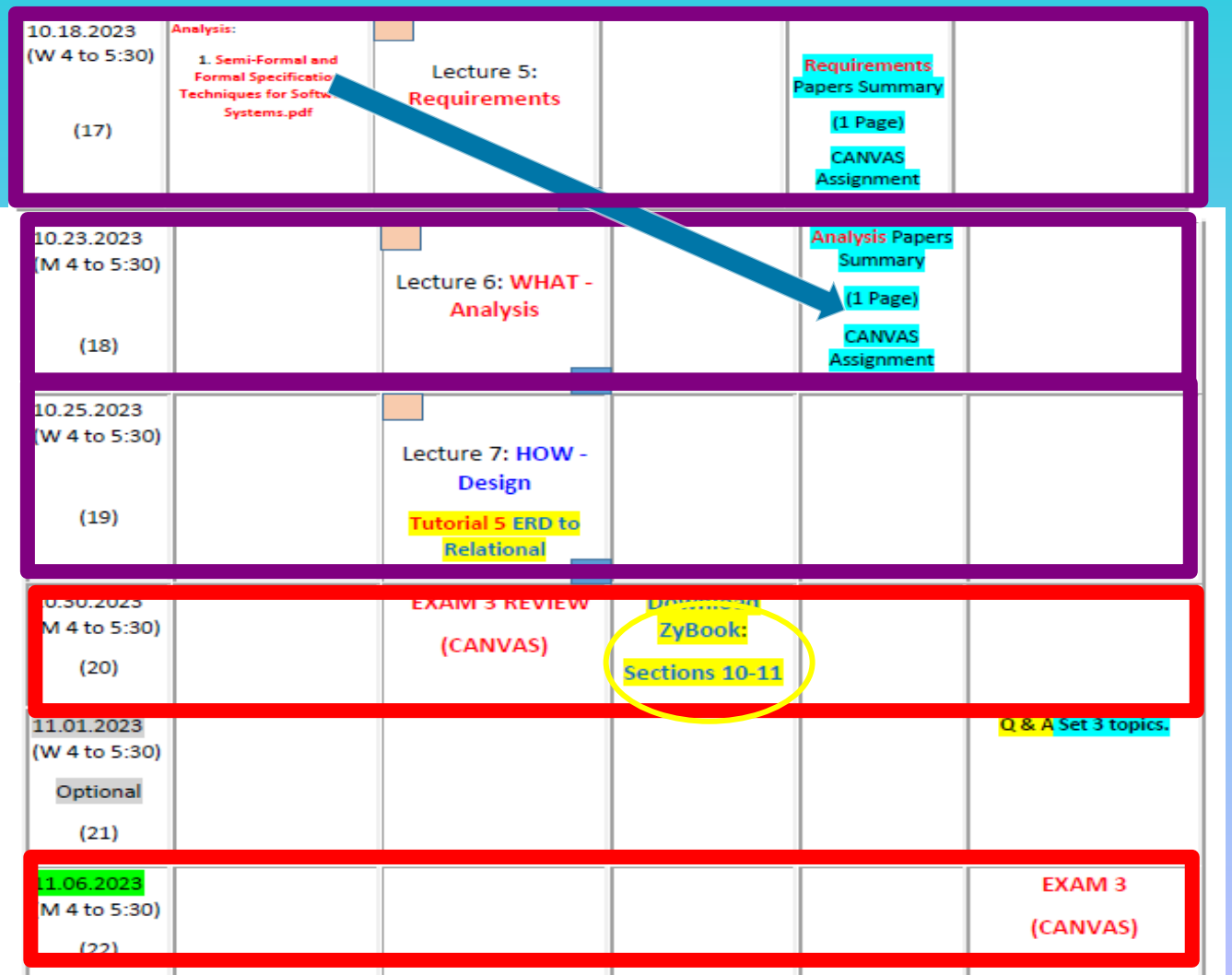
LOG IN

CANVAS

Youyi [A-L]

Kevin [M-Z]

Please close all other windows.


10.18.2023
(W 4 to 5:30)
(17)
Analysis:
1. Semi-Formal and Formal Specificatio Techniques for Softw Systems.pdf
Lecture 5: Requirements
Requirements Papers Summary
(1 Page)
CANVAS Assignment
10.23.2023
(M 4 to 5:30)
(18)
Lecture 6: WHAT - Analysis
Analysis Papers Summary
(1 Page)
CANVAS Assignment
10.25.2023
(W 4 to 5:30)
(19)
Lecture 7: HOW - Design
Tutorial 5 ERD to Relational
M 4 to 5:30)
(20)
EXAM 3 REVIEW
(CANVAS)
ZyBook:
Sections 10-11
11.01.2023
(W 4 to 5:30)
Optional
(21)
Q & A Set 3 topics.
11.06.2023
M 4 to 5:30)
(22)
EXAM 3
(CANVAS)

Class 17

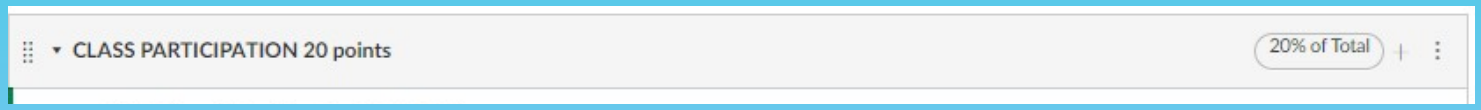
COSC 4351

Software engineering

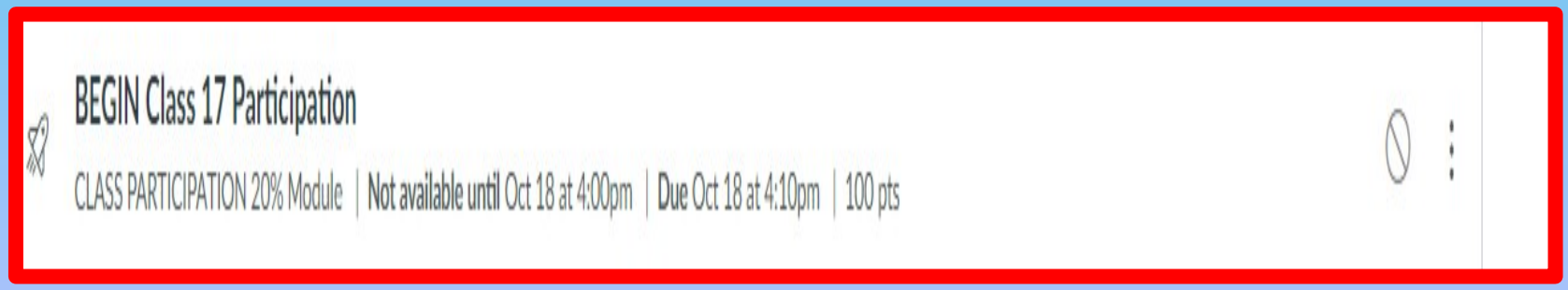
10.18.2023 (W 4 to 5:30) (17)	Analysis: 1. Semi-Formal and Formal Specification Techniques for Software Systems.pdf	Lecture 5: Requirements		Requirements Papers Summary (1 Page) CANVAS Assignment	

From 4:00 to 4:10 – 10 minutes.

10.18.2023 (W 4 to 5:30) (17)	Analysis: 1. Semi-Formal and Formal Specification Techniques for Software Systems.pdf	Lecture 5: Requirements		Requirements Papers Summary (1 Page) CANVAS Assignment	



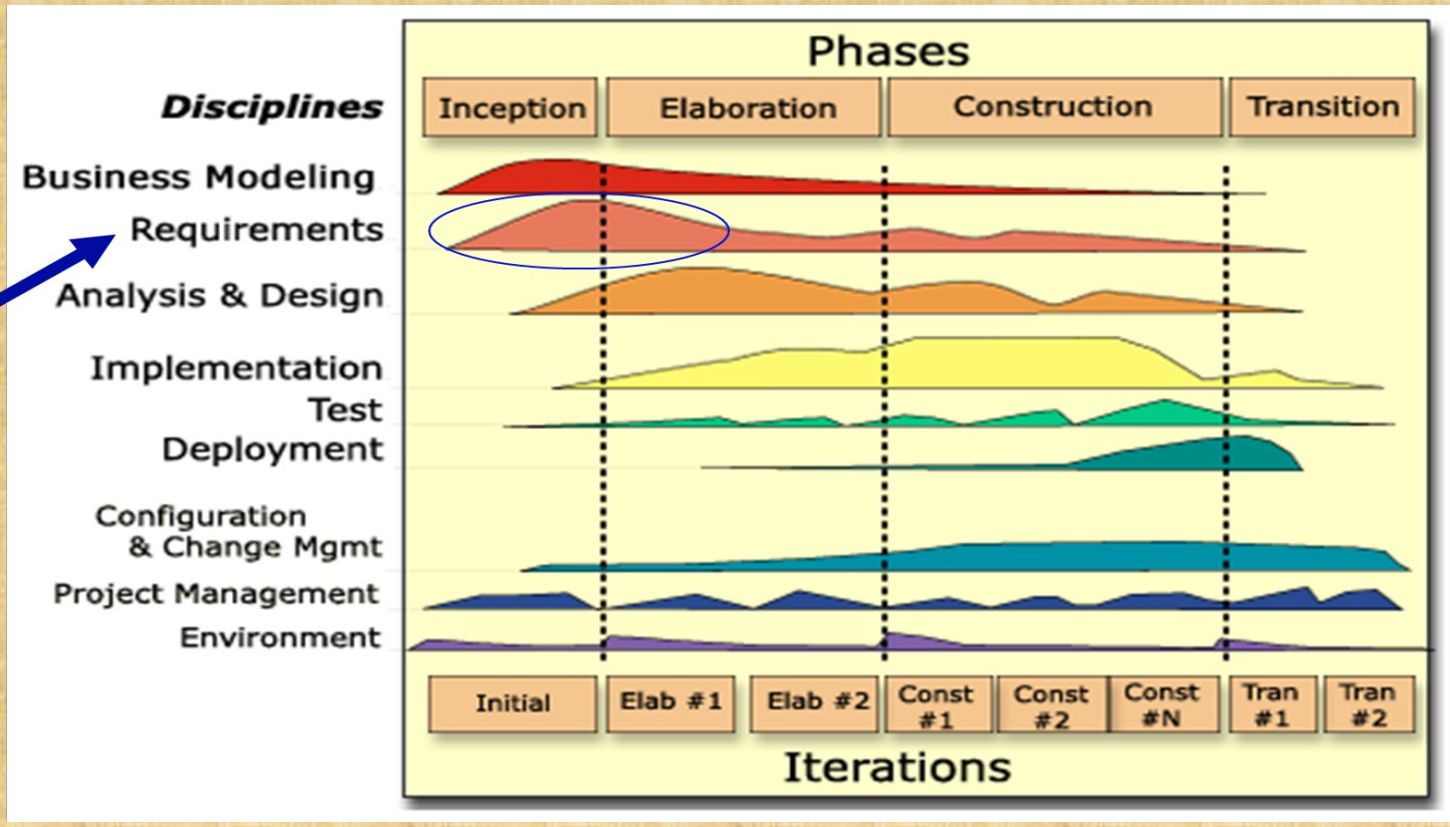
PASSWORD: I AM IN TEAMS



VH– Publish.

From 4:10 to 5:00 – 50 minutes.

Requirements Workflow

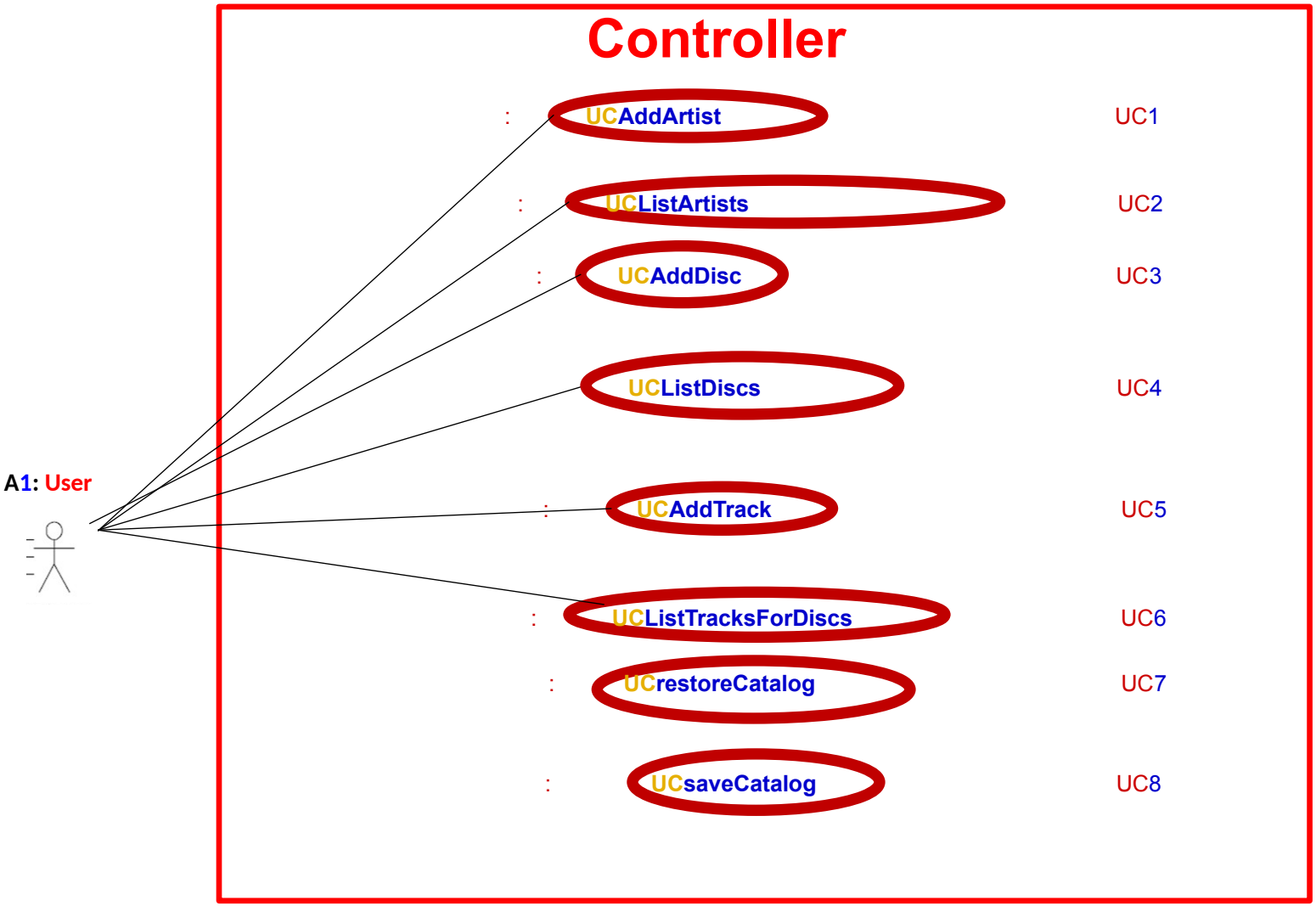


Overview

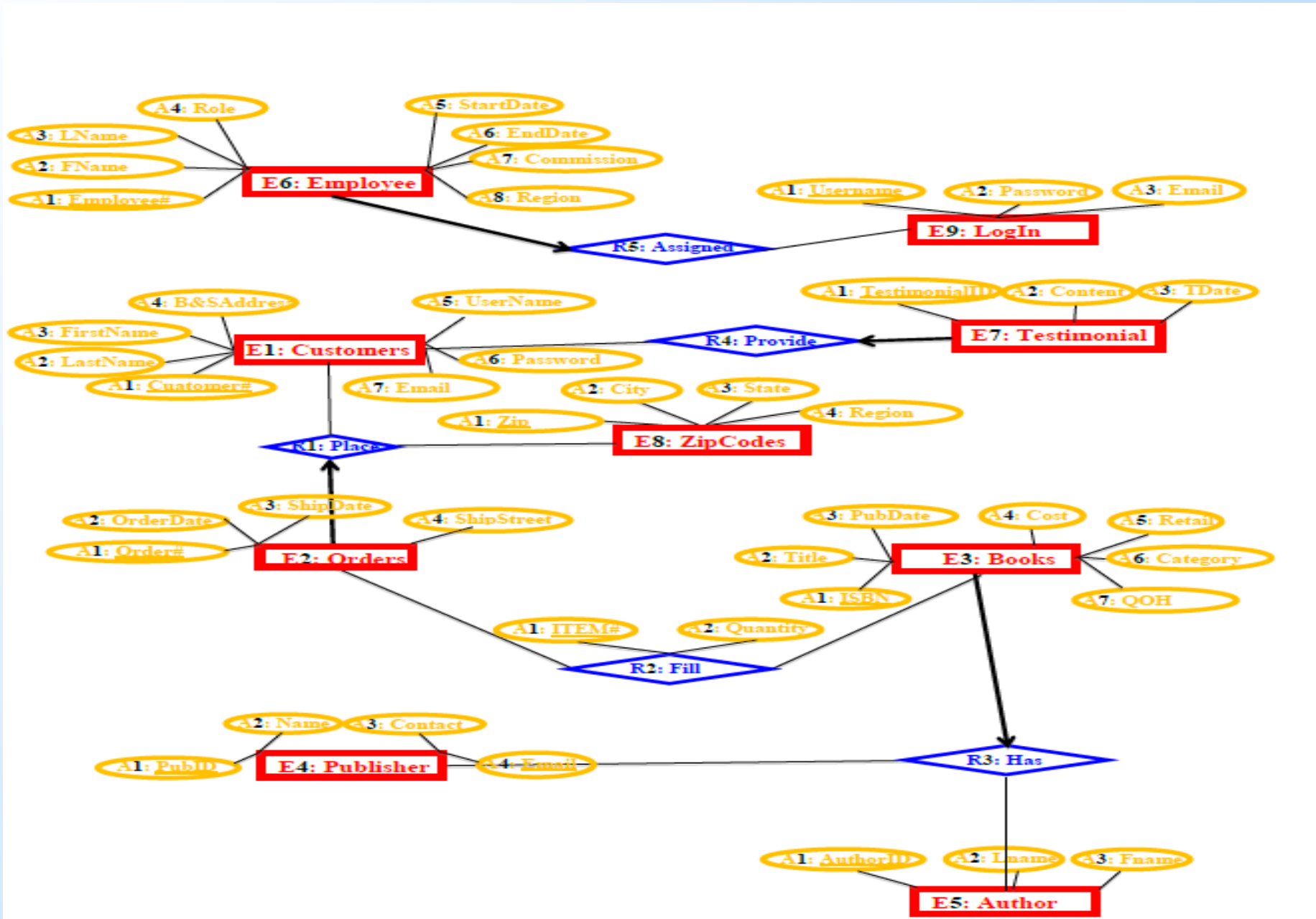
- Determining what the **Client** needs
- **Overview of the Requirements Workflow**
  - Understanding the Domain
  - The Business Model
  - Initial **Requirements**
- The **Classical Requirements** Phase
- **Rapid Prototyping**
- Human factors
- **Reusing** the rapid prototype
- CASE tools for the **Requirements** Workflow
- **Metrics** for the **Requirements** workflow
- **Challenges of the Requirements Workflow**

Movie Company System

UML USE CASE MODEL



ERD Model


A4: Role
A5: StartDate
A3: LName
A6: EndDate
A2: FName
A7: Commission
A1: Employee#
E6: Employee
A8: Region
A1: Username
A2: Password
A3: Email
E9: LogIn
R5: Assigned
A4: B&SAddress
A5: UserName
A1: TestimonialID
A2: Content
A3: TDate
A3: FirstName
E1: Customers
R4: Provide
E7: Testimonial
A2: LastName
A6: Password
A1: Customer#
A7: Email
A2: City
A3: State
A4: Region
A1: Zip
E8: ZipCodes
R1: Place
A3: ShipDate
A2: OrderDate
A4: ShipStreet
A1: Order#
E2: Orders
A3: PubDate
A4: Cost
A5: Retail
A2: Title
E3: Books
A6: Category
A1: ISBN
A7: QOH
A1: ITEM#
A2: Quantity
R2: Fill
A2: Name
A3: Contact
A1: PubID
E4: Publisher
A4: Email
R3: Has
A1: AuthorID
A2: Lname
A3: Fname
E5: Author

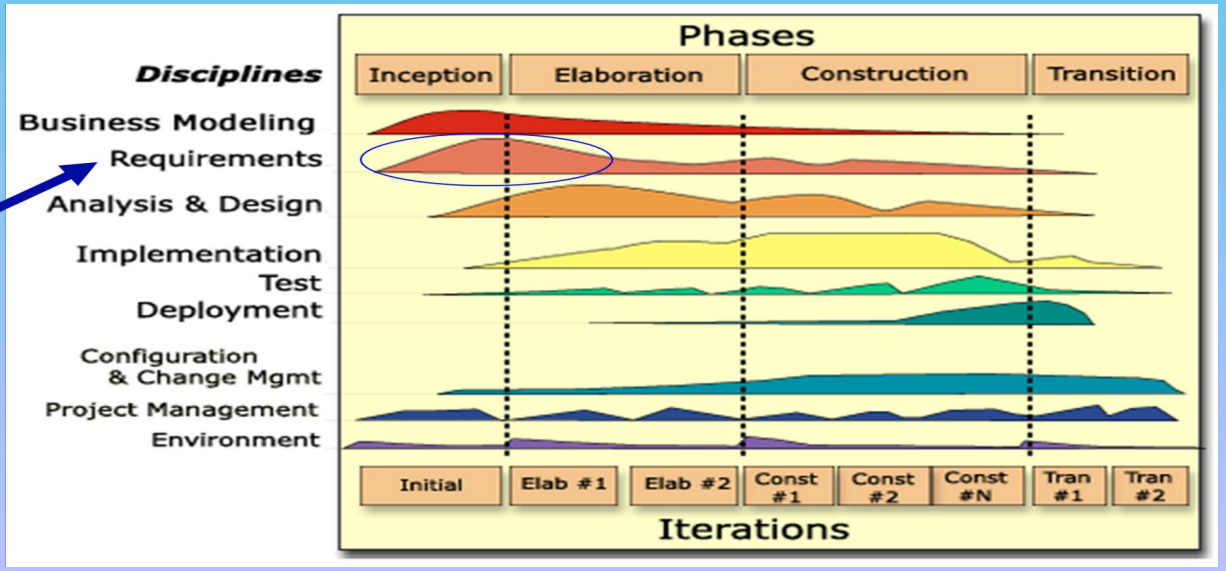
REQUIREMENTS Workflow

TA -> UML USE CASE DIAGRAM.doc

UML -> Software Requirements Specification.doc

TA -> ERD Modeling

ERD -> Data Requirements Specification



REQUIREMENTS

"The hardest single part of building a software system is deciding what to build. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

Fred Brooks

Early in a project, there is a phase in which the client and the contractor work together to create a description of what is to be built. This is called the specification and building a complex software product without a clear, fixed set of specifications is impossible.

Software Requirements Specification.doc

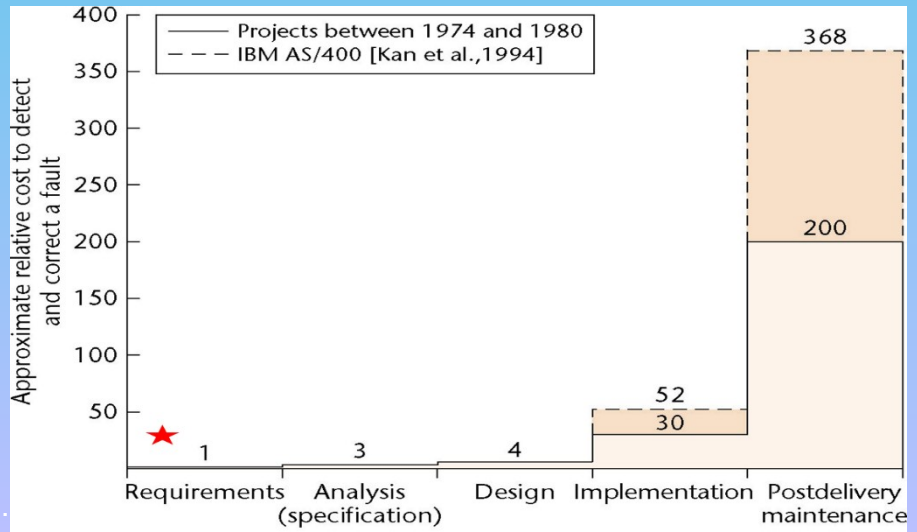
Data Requirements Specification

Requirements:

1. Why Software Projects Fail 2015.pdf
2. The Why What who When and How of Software Requirements 2015.pdf

2003

Requirements errors account for 70 percent to 85 percent of the rework costs on a software project (Wiegers 2003). If one finds a requirements defect during the requirements phase and it costs one unit to fix (for example, three engineering hours, $500), the cost of fixing that same defect will typically increase as it is found later and later in the life cycle. In fact, studies show that it can cost more than 100 times more to fix a requirements defect if it is not found until after the software is released to the field.



Laws

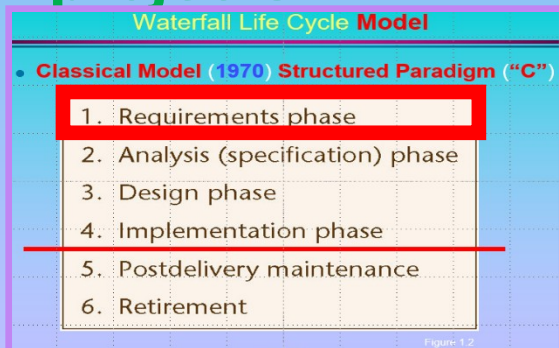
BUT, **1998**!

Requirements deficiencies are the prime source of **project** failures (L1)

- Source: Robert Glass [Glas 98] et al
- Most defects (> 50%) stem from **Requirements**
- **Requirements** defects (if not removed quickly) trigger follow-up defects in late activities

Possible solutions:

- early inspections
- **formal** specifications & validation early on
- other forms of **Prototyping** & validation early on
- **Reuse** of **Requirements** documentation from **similar projects**



Laws

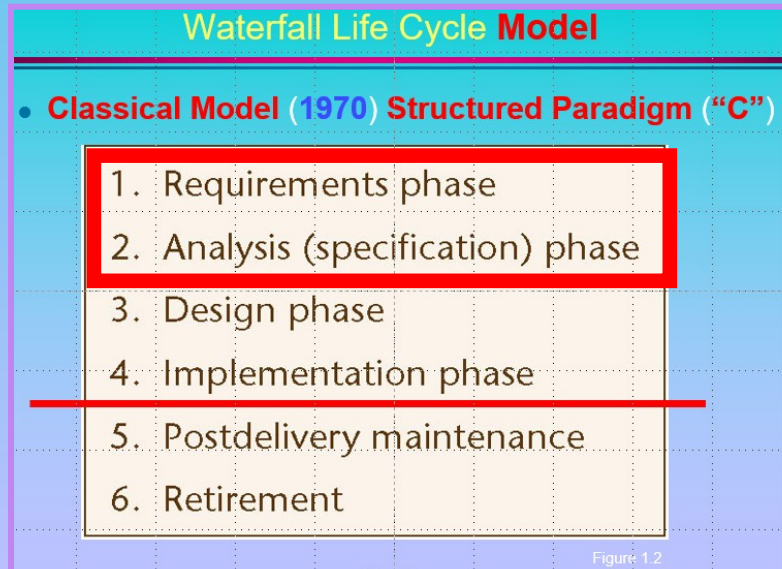
BUT, **1975**!

Defects are most frequent during **Requirements** and **Analysis** activities and are more expensive the later they are removed (L2)

- Source: Barry Boehm [Boeh 75] et al
- >80% of defects are caused up-stream (**Requirements**, **Analysis**)
- Removal delay is expensive (e.g., factor 10 per phase delay)

HealthCare.gov October 21, 2013

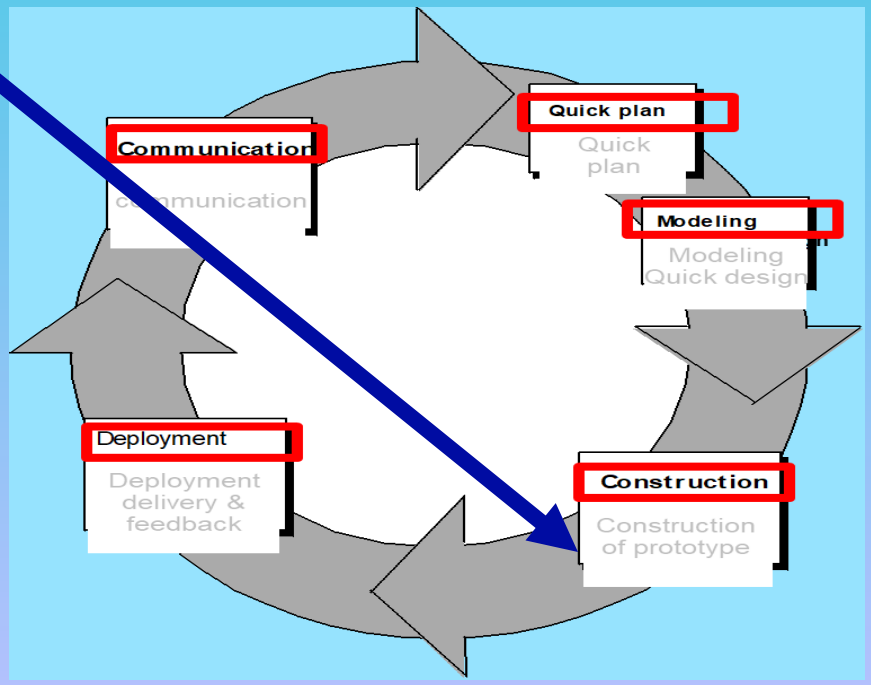
For large software projects, failure is generally determined early in the process, because failures almost exclusively have to do with planning: the failure to create a workable plan, to stick to it, or both. Healthcare.gov reportedly involved over fifty-five contractors, managed by a human-services agency that lacked deep experience in software engineering or project management. The final product had to be powerful enough to navigate any American through a complex array of different insurance offerings, secure enough to hold sensitive private data, and robust enough to withstand peak traffic in the hundreds of thousands, if not millions, of concurrent users. It also had to be simple enough so that anyone who can open a Web browser could use it. In complexity, this is a project on par with the F.B.I.'s V.C.F. or Sentinel. The number and variety of systems to be connected may not be quite as large, but the interface had to be usable by anyone, without special training. And, unlike V.C.F., Healthcare.gov was given only twenty-two months from contract award to launch—less than two years for a project similar to one that took the F.B.I. more than ten years and over twice the budget.



Laws

BUT, **1984**!

- **Prototyping** (significantly) reduces **R**equirements and **D**esign defects, especially those related to the **U**ser **I**nterface (L3)
  - Source: Barry Boehm [Boeh**84**a]
  - Prototype life-cycle Model



REQUIREMENTS ENGINEERING (RE)

QUICK LOOK

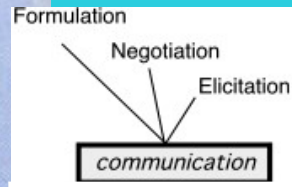
What is it? Requirements engineering helps software engineers to better understand the problem they will work to solve. It encompasses the set of tasks that lead to an understanding of what the business impact of the software will be, what the customer wants, and how end-users will interact with the software.

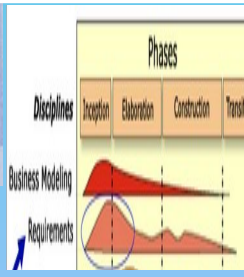
Who does it? Software engineers (sometimes referred to as *system engineers* or *analysts* in the IT world) and other project stakeholders (managers, customers, end-users) all participate in requirements engineering.

Why is it important? Designing and building an elegant computer program that solves the wrong problem serves no one's needs. That's why it is important to understand what the customer wants before you begin to design and build a computer-based system.

What are the steps? Requirements engineering begins with inception—a task that defines the scope and nature of the problem to be solved. It moves onward to elicitation—a task that helps the customer to define what is required, and then elaboration—where basic requirements are refined and modified. As the customer defines the problem, negotiation occurs—what are the priorities, what is essential, when is it required? Finally, the problem is specified in some manner and then reviewed or validated to ensure that your understanding of the problem and the customers' understanding of the problem coincide.

UML USE CASE DIAGRAM.doc





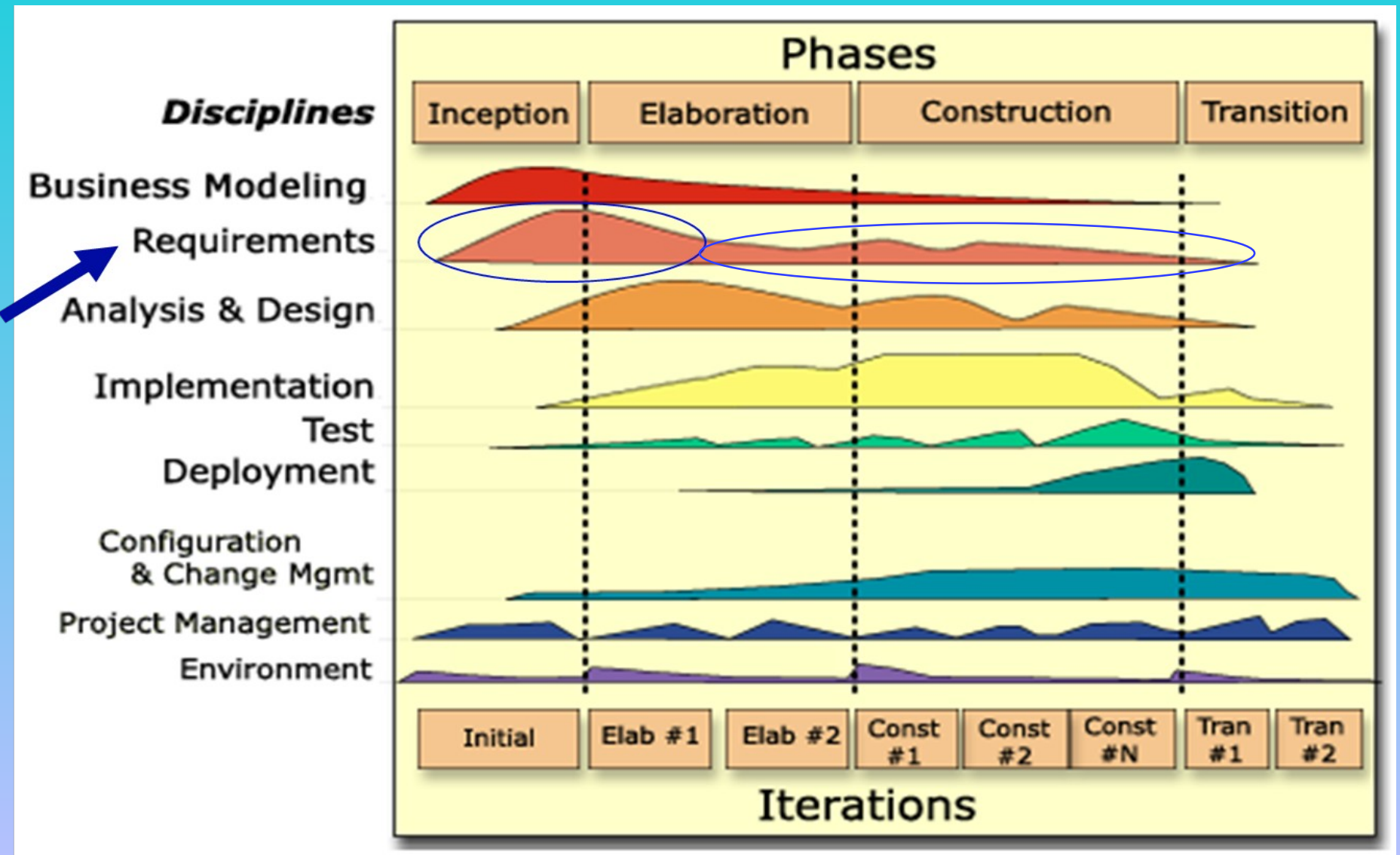
What is the work product? The intent of the requirements engineering process is to provide all parties with a written understanding of the problem. This can be achieved though a number of work products: user scenarios, functions and features lists, analysis models, or a specification.

SRS

How do I ensure that I've done it right? Requirements engineering work products are reviewed with the customer and end-users to ensure that what you have learned is what they really meant. A word of warning: even after all parties agree, things will change, and they will continue to change throughout the project.

Signed SRS

REQUIREMENTS Workflow



IEEE Standard for SRS

http://www.cse.yorku.ca/course_archive/2004-05/F/4312/

Software Requirements Specification.doc

1 Introduction
- Purpose
- Scope
- Definitions, acronyms, abbreviations
- Reference documents
- Overview

Identifies the product & Application Domain

Describes contents and structure of the remainder of the SRS

Describes all external interfaces:

parallelism, etc)

Appendices

TABLE1: Requirements

ID	Detail	Type	Priority	Line Numbers
R1	The TEAM?OPTS shall display a list of queries available to CMs	Queries Functional	MustHave	145 - 173
.....				
R26	The TEAM?OPTS shall authenticate CMs.	Authentication Non Functional	MustHave	223 - 227
.....				

TABLE 1: Actors

TABLE 2: LABELED USE CASES

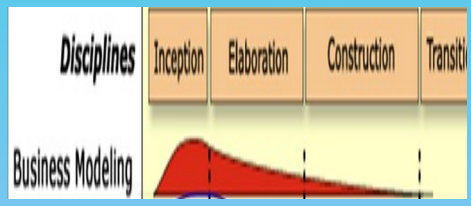
FIGURE 1: LABELED UML USE CASE DIAGRAM

Software Specification

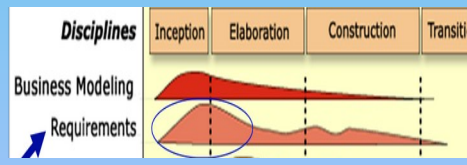
The **Process** of establishing **what** services are required and the constraints on the **system**'s **operation** and **development**.

Requirements Engineering (RE) Process

- Feasibility Study;



- Requirements Elicitation and Analysis;

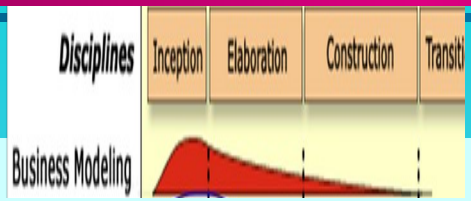


- Requirements Specification;

- Requirements Validation.

SQA Review Meeting

The Requirements Engineering(RE) *Process*

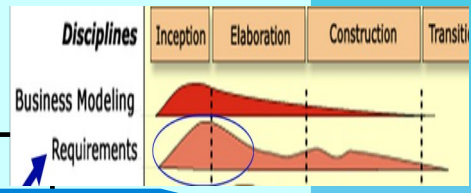


UML USE CASE DIAGRAM.doc

Feasibility study

Requirements elicitation and analysis

Requirements specification



Feasibility report

Requirements validation

SQA

System models

User and system requirements

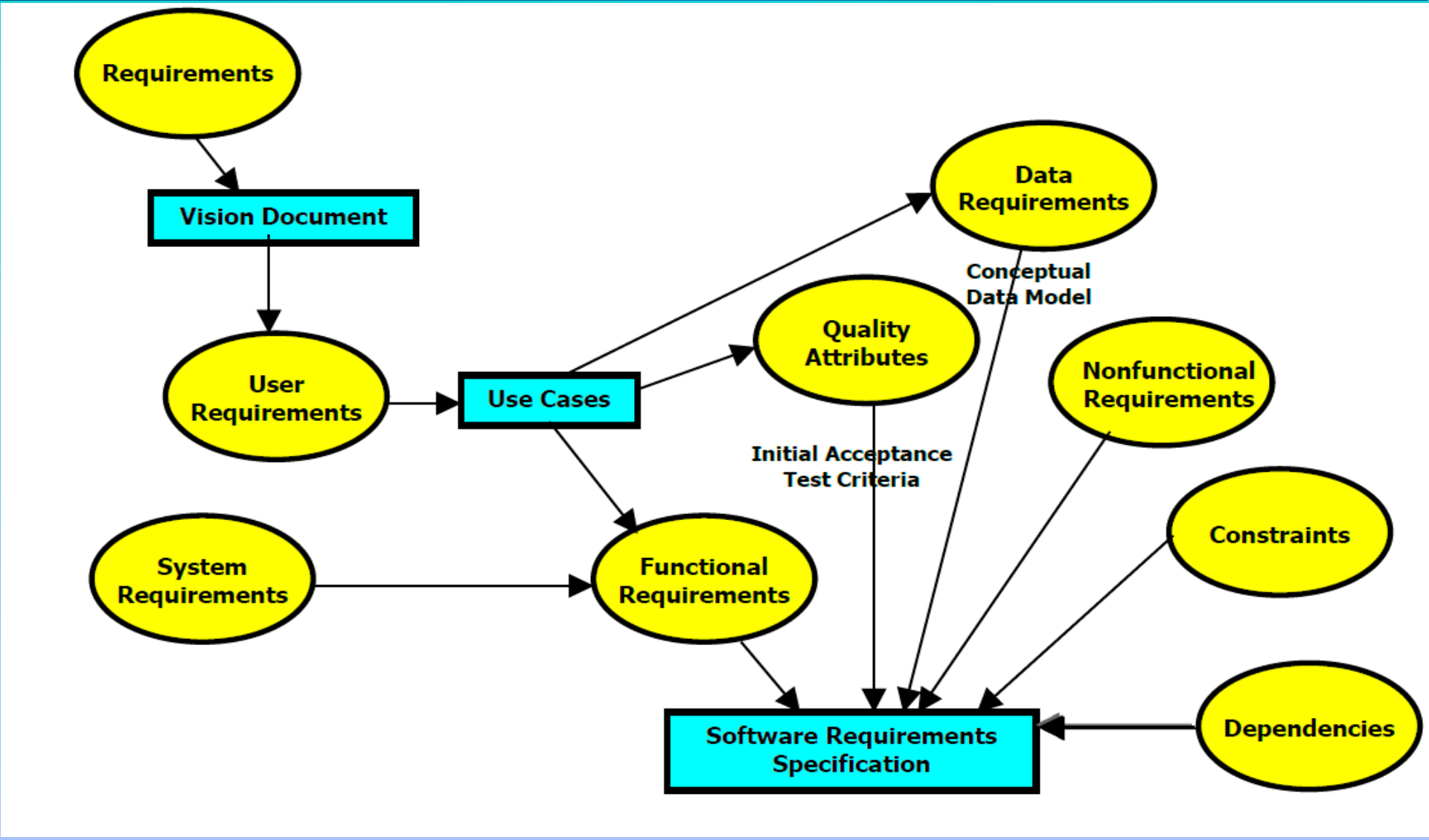
Requirements document

Feasibility Study.doc

SRS
Signed SRS

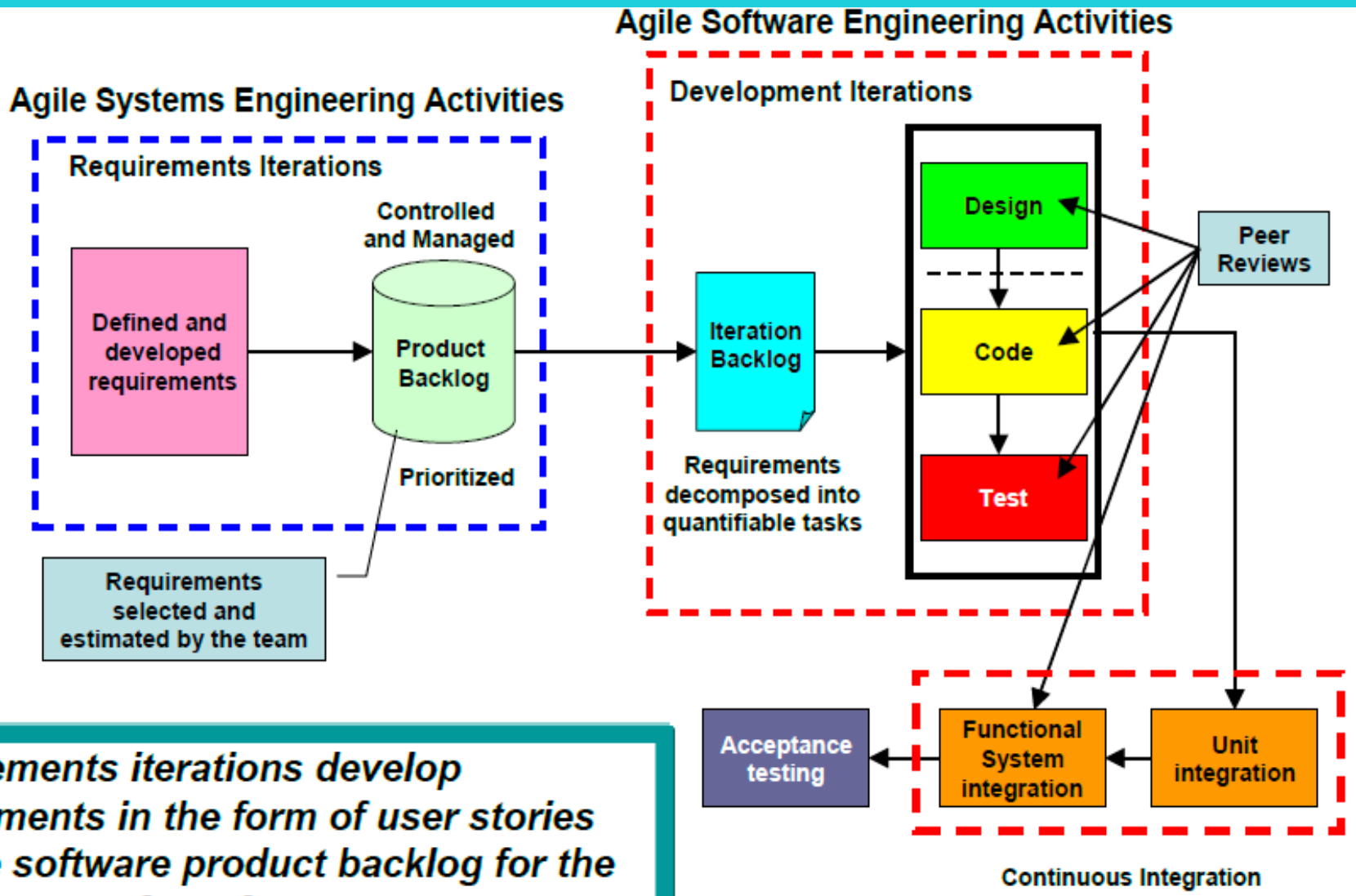
Software Requirements Specification.doc

Traditional Requirement Engineering



https://www.codeproject.com/articles/674450/agile-software-development-steps-to-work-with-requ

Agile Requirement Engineering



Requirements iterations develop requirements in the form of user stories into the software product backlog for the development of a software system

Estimation

One of the most challenging aspects of any software project is estimation -- determining how long the work will take.

Fred Brooks says it best:

“It is very difficult to make a vigorous, plausible, and job-risking **defense of an estimate** that is derived by **no quantitative method**, **supported by little data**, and certified chiefly by the hunches of the managers.”

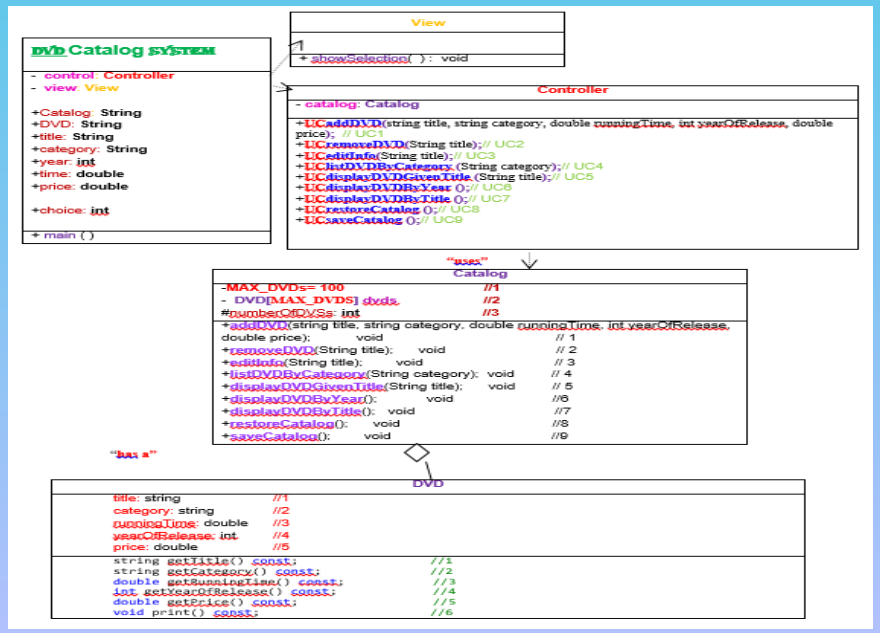
Bottom-up Approach

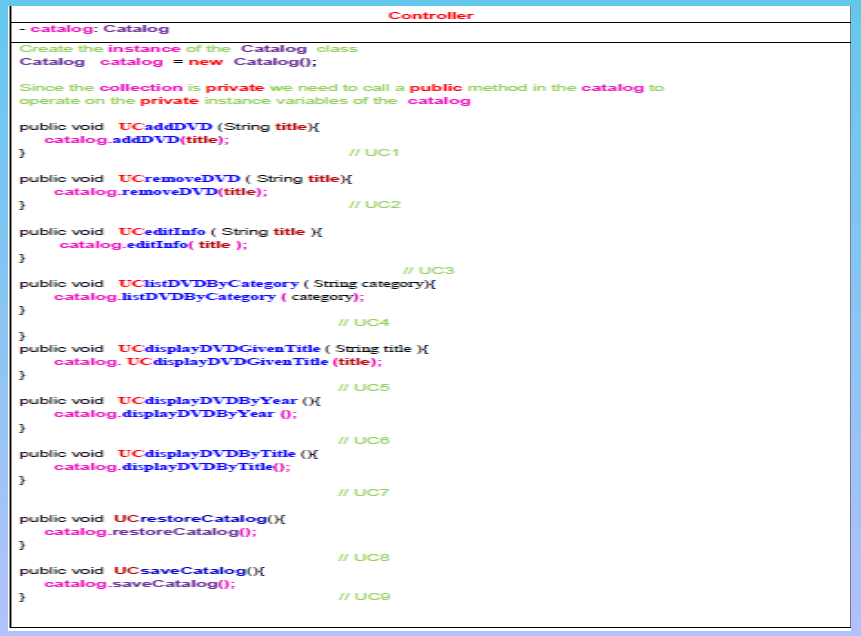
When using the **Object Oriented Paradigm**

- The independence of the **Classes** assists here
- However, the interactions among the **Classes** (coupling) **complicate** the **Estimation process**

thus, MVC

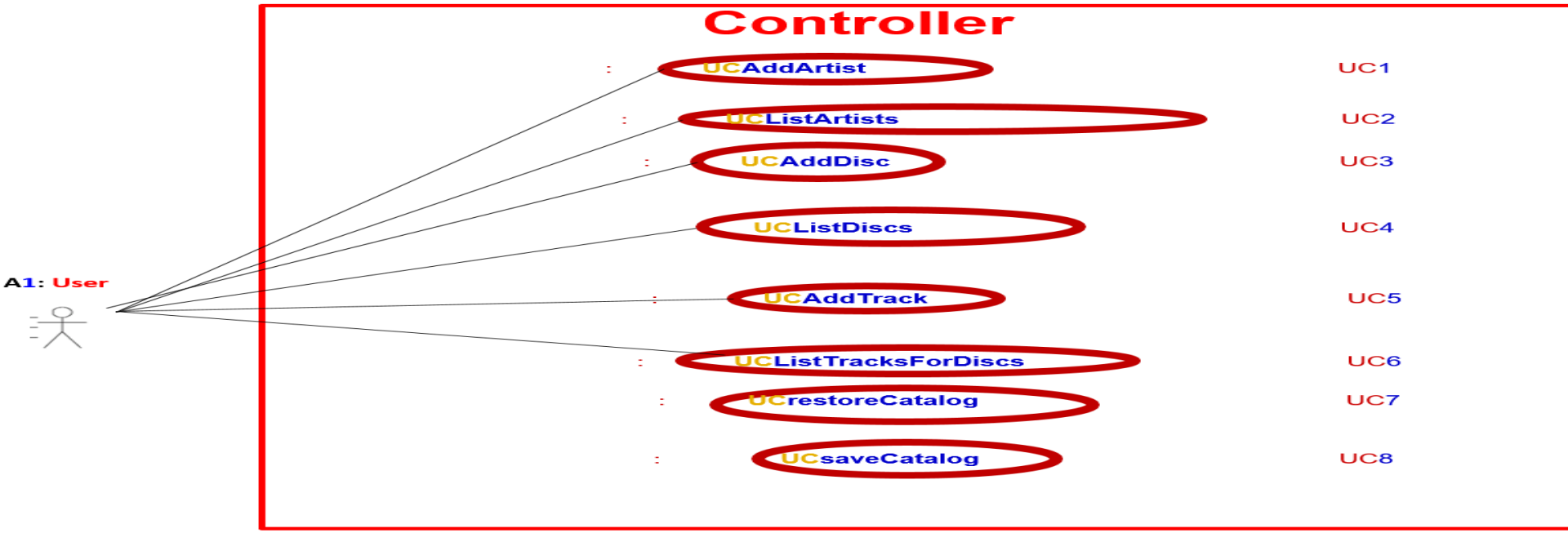
DVDCollection Application - ? Classes ? methods





Movie Company System

UML USE CASE MODEL



Movie Company System

UML USE CASE Description

UC2 UCListArtists Description

Name:	**UCListArtists**
Actor:	**User**
Description:	**This use case describes the process used by User to List all Artists**
Successful Completion:	**User requests Listing Artists** **1. Movie Company System displays all the Artists by traversing the artists array**
Alternative:	
Pre-Condition:	**User requests List Artists**
Post-Condition:	**Artists displayed**
Assumptions:	**None**

User Story

Step 4: UML Use Cases Description

The **Aim** of the **R**equirements **W**orkflow

To answer the question:

"What" and "Why" must the **product** be able to do!

But not "How"!

In practice, this separation cannot be strictly maintained. Some **Requirements** may directly *prescribe certain* ***Designs*** or ***Implementation*** aspects, such as compatibility with existing interfaces or language to be used for the implementation.

MVC

RoR C#

In order to be able to define certain **Requirements**, some basic ***Design decisions*** may have to be made first. Also, **Developers** may need an overall idea "How" *certain functions can be* ***Implemented*** before they can accept them as meaningful **Requirements**.

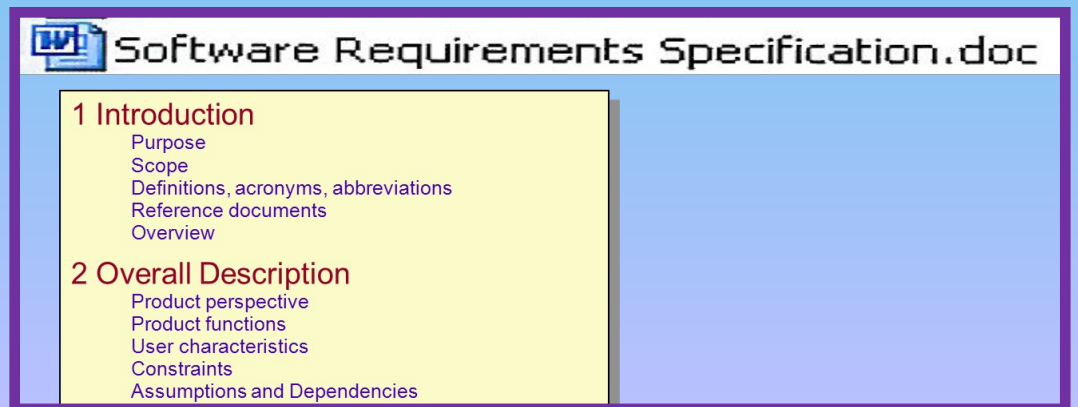
Types of Requirements

User Requirements

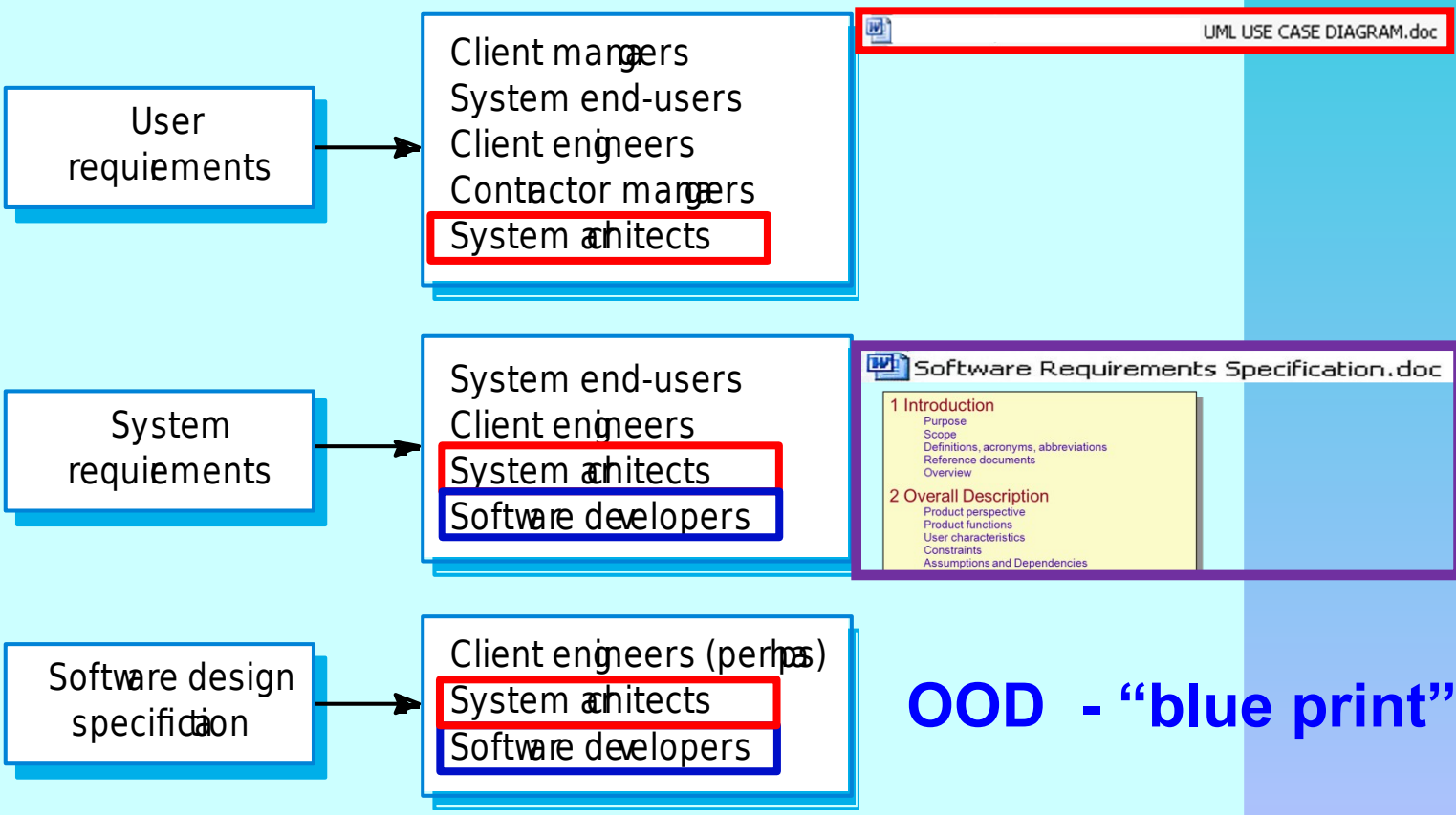
- Statements in natural language plus Diagrams of the **services** the **system** provides and its operational constraints. Written for **Customers**.

System Requirements

- A structured document setting out detailed descriptions of the **system**'s functions, services and operational constraints. Defines "What" should be **Implemented** so may be part of a ***Contract*** *between* ***Client*** *and* ***Contractor***.



Requirements Readers



Functional, Non-Functional, and Domain Requirements

Software system Requirements are classified as:

- **Functional Requirements**
  - Statements of **services** the **system** should provide, how the **system** should react to particular inputs and how the **system** should behave in particular situations.

- **Non-Functional Requirements**
  - **constraints** on the **services** or **functions** offered by the **system** such as timing **constraints**, **constraints** on the **Development Process**, **Standards**, etc.

- **Domain Requirements**
  - Requirements that come from the Application Domain of the **system** and that reflect characteristics of that Domain.

Microsoft WORD

Functional Requirements

- Describe functionality or **system** services.
- Depend on the type of **software**, expected **Users** and the type of **system** where the **software** is used.
- Functional **User Requirements** may be **high-level statements** of **what** the **system** should do but Functional **System Requirements** should **describe** the **system** services **in detail**.

Examples of **Functional Requirements**

- **R13**: The **User** *shall be able* to search either all of the initial set of **Data**bases or select a subset from it.
- **R40**: The **System** *shall provide* *appropriate viewers* for the **User** to read documents in the document store.
- **R49**: Every order *shall be* allocated a unique identifier (ORDER_ID) which the **User** shall be able to copy to the account's permanent storage area.

Requirements Imprecision

Problems arise when **Requirements** are not **precisely** stated.

Ambiguous **Requirements** may be interpreted in different ways by **Developers** and **Users**.

Consider the term '*appropriate viewers*'

- **User intention** - special purpose viewer **for each** different document type;
- **Developer interpretation** - Provide a **text viewer** that shows the contents **of each** document.

Requirements *Complete & Consistent*

In principle, Requirements should be both *Complete & Consistent* .

Complete They should include descriptions of **all** facilities required.

Consistent There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a *Complete & Consistent* **Requirements** Document.

Functional Requirements

Software system Requirements are classified as:

- **Functional Requirements**
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

Movie Company System

UML USE CASE Description

UC2 UCListArtists Description

Name:	**UCListArtists**
Actor:	**User**
Description:	**This use case describes the process used by User to List all Artists**
Successful Completion:	**User requests Listing Artists** **1. Movie Company System displays all the Artists by traversing the artists array**
Alternative:	
Pre-Condition:	**User requests List Artists**
Post-Condition:	**Artists displayed**
Assumptions:	**None**

User Story

Non-Functional Requirements

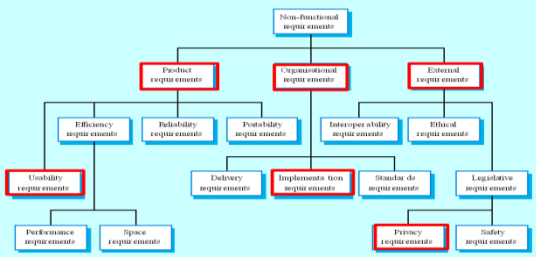
These define **system** properties and **constraints**

- properties: Reliability, Response Time and Storage Requirements,
- **constraints**: I/O device capability, **system** representations, etc.

Process Requirements may also be specified mandating a particular CASE System, **Programming Language** or **Development method**.

Non-Functional Requirements may be more critical than **Functional Requirements**. If these are not met, the **system** is useless.

Not MVC



Non-Functional Classifications

Product Requirements

- Requirements which specify that the delivered product must behave in a particular way e.g. Execution Speed, Reliability, etc.

Organisational Requirements

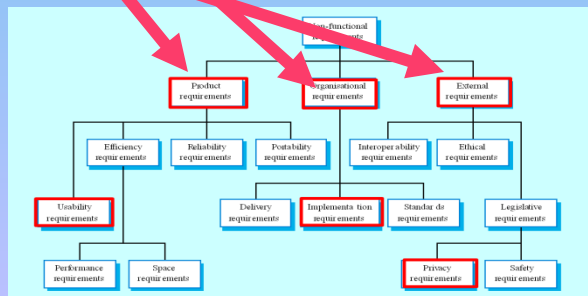
- Requirements which are a consequence of organisational **policies** and **procedures** e.g. Process Standards used, Implementation **Requirements**, etc

UML **MVC**

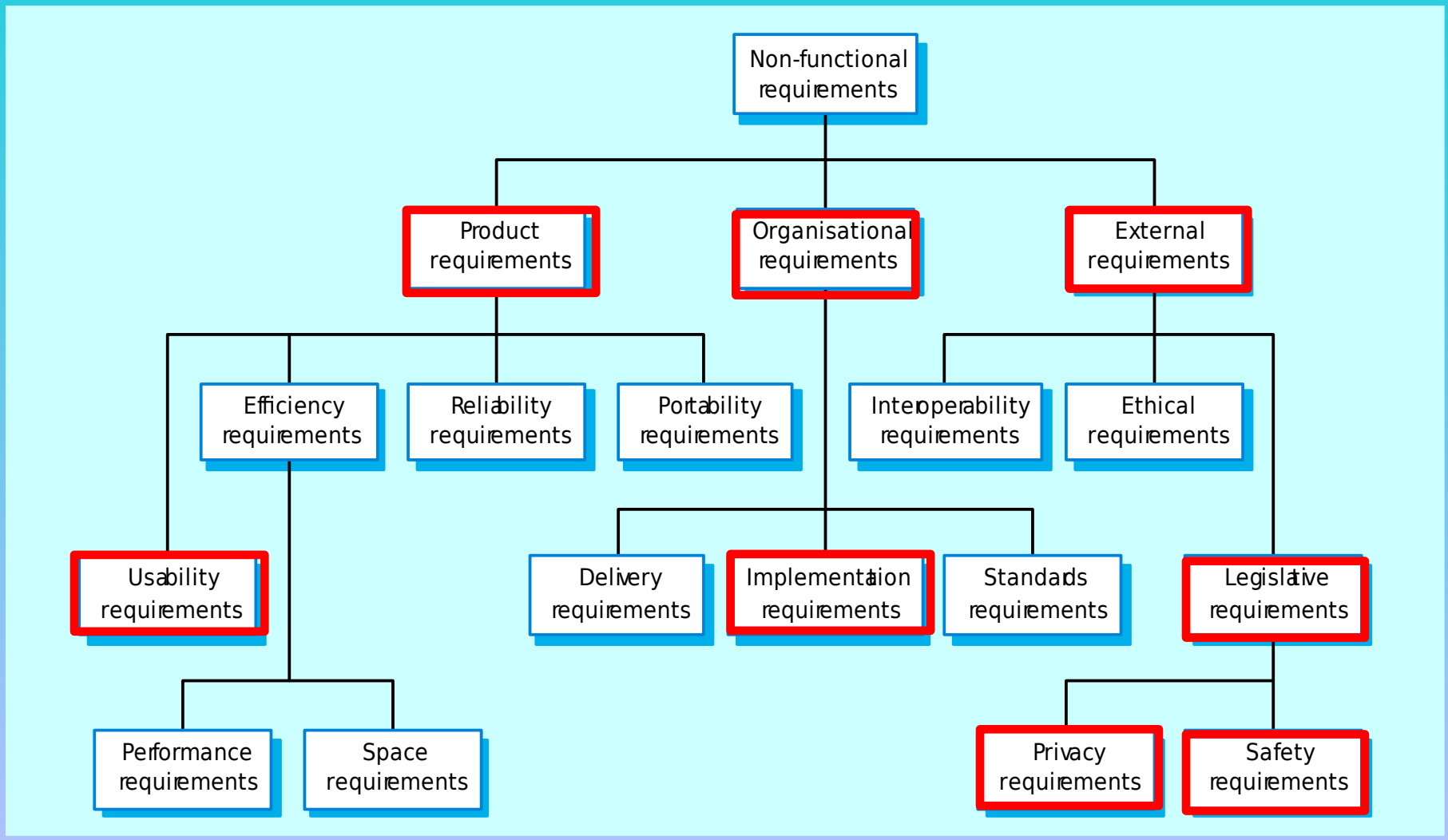
External Requirements

- Requirements which arise from factors which are external to the system and its **Development Process** e.g. Interoperability **Requirements**, Legislative **Requirements**, etc.

HIPPA



Non-Functional Requirement Types



Non-Functional Requirements examples

Product Requirements

R.8.1 The User Interface for LIBSYS *shall be* implemented as simple HTML without frames or Java applets.

Organisational Requirements

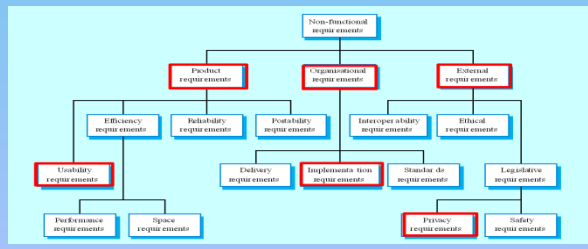
R.9.3.2 The system Development **Process** and deliverable documents *shall conform* to the **Process** and Deliverables defined in XYZCo-SP-STAN-95.

Unified Process **SCRUM**

External Requirements

R.7.6.5 The system *shall not disclose* any personal information about **Customers** apart from their name and reference number to the **Operators** of the system.

HIPPA



Goals and Requirements

Non-Functional Requirements may be **very difficult to state** precisely and imprecise Requirements may be difficult to **Verify**.

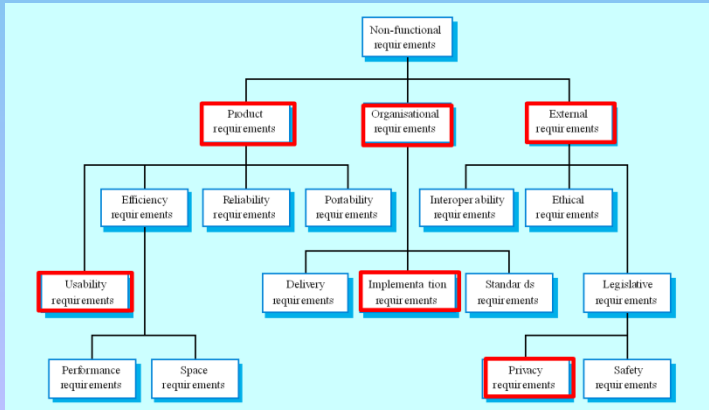
Goal

- A **general intention** of the **User** such as ease of use.

Verifiable Non-Functional **R**equirement

- A statement using **some measure** that can be objectively **tested**.

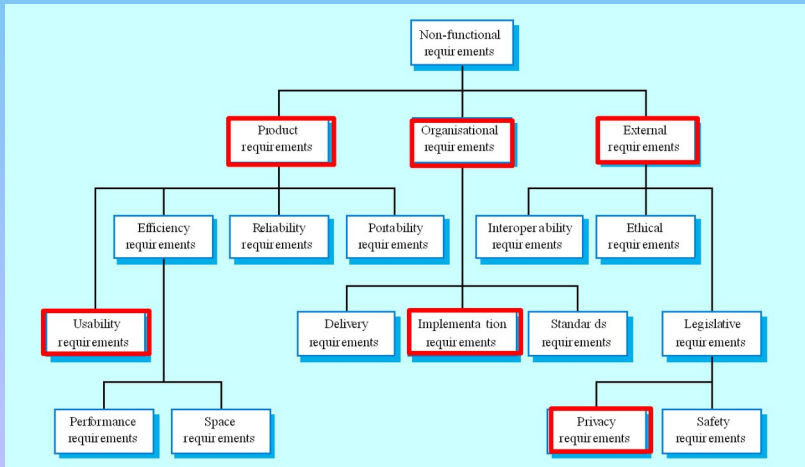
Goals are helpful to **Developers** as they convey the intentions **of** the **system Users**.



Examples

- **A System Goal**
  - The **system** *should be* easy to use by experienced **Controllers** and should be organised in such a way that **User** errors are minimized.

- **A Verifiable Non-Functional Requirement**
  - Experienced **Controllers** *shall be* able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced **Users** shall not exceed **two per day**.



Examples

Boeing 737 Max 8

A Verifiable Non-Functional Requirement

- Experienced **Controllers** *shall be* able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced **Users** shall not exceed **two per day**.

US pilots got an extra hour or two of training to prepare to use Boeing's new 737 Max 8 aircraft, which has been involved in two deadly crashes in less than six months.

The first crash involved a Lion Air flight in Indonesia that plunged into the sea soon after takeoff in October last year, killing all 189 on board. A week ago, an Ethiopian Airlines flight crashed six minutes after taking off, killing 157 people. After the second crash, China grounded the aircraft, and was quickly followed by other nations. The US Federal Aviation Administration initially allowed airlines in the US to keep flying the plane, but on Wednesday (March 13) grounded both the Max 8 and Max 9 models, citing refined satellite data and "physical evidence" found at the scene that showed similarities between the two crashes.

- The only metric of success that really matters is the one we ignore

The aircraft was developed to include changes to the flight control system that would sharply pitch the plane's nose down if the onboard computer sensed an imminent stall. The system, known as the Maneuvering Characteristics Augmentation System, or MCAS, was incorporated because the plane has larger engines placed further forward on the craft, and so has a chance of facing a stall at lower speeds than earlier 737 variations. In the Lion Air flight pilots appeared to be fighting against the system before the aircraft crashed, according to the preliminary accident report.

Requirements Measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Quality of a Software Product

- ***Software Product*** *Quality – is usually defined as the degree to which it* ***meets*** ***Customer*** **R**equirements. *This view emphasizes* ***one side*** *of the Quality: the* ***User****'s perspective. A more comprehensive view also includes the* ***Developer****'s side.*

View	Criterion	Definition
User	Availability	High degree of access
	Reliability	Low failure rate
	Efficiency	Economic resource consumption
	Installability	Easy and fast bring-up in user environment
	Usability	Well adapted to skills and preferences of user
	Robustness	Safe reaction to user errors and hardware failures
	Safety/Security	Low damage in case of negligent/malicious use
Developer	Testability	Good documentation and structure
	Maintainability	High readability and modifiability
	Portability	Low dependency on technical environment
	Localizability	Adaptable to national and regional requirements
	Reusability	High modularity and completeness

Quality of a Software Product

- *Availability – the **unit** is % (e.g. **99.9%**)*
- *MTTF – Mean Time To Failure and*
- *MTTR – Mean Time To Repair*

$$\text{Availability} = \text{MTTF}/(\text{MTTF} + \text{MTTR})$$

View	Criterion	Definition
User	Availability	High degree of access
	Reliability	Low failure rate
	Efficiency	Economic resource consumption
	Installability	Easy and fast bring-up in user environment
	Usability	Well adapted to skills and preferences of user
	Robustness	Safe reaction to user errors and hardware failures
	Safety/Security	Low damage in case of negligent/malicious use
Developer	Testability	Good documentation and structure
	Maintainability	High readability and modifiability
	Portability	Low dependency on technical environment
	Localizability	Adaptable to national and regional requirements
	Reusability	High modularity and completeness

Requirements Interaction

Conflicts between *different* *Non-Functional* Requirements are common in complex **systems**.

Spacecraft **system**

- To minimise weight, the **number of separate chips** in the **system** should be **minimised**.
- To minimise power consumption, **lower power chips** should be **used**.
- However, using **low power chips** may mean that **more chips** have to be used. Which is the most critical Requirement?

Non-Functional Requirements

Software system Requirements are classified as:

- **Functional** Requirements
  - Statements of services the **system** should provide, how the **system** should react to particular inputs and how the **system** should behave in particular situations.

TABLE1: Requirements

.....				
R26	The TEAM?OPTS shall authenticate CMs.	Authentication Non Functional	MustHave	223 - 227
.....				

Domain Requirements

- Derived from the *Application Domain* and describe **system** characteristics and features that reflect the *Domain*.
- **Domain Requirements** may be new **Functional Requirements**, constraints on existing **Requirements** or define specific computations.
- If **Domain Requirements** are not satisfied, the **system** may be unworkable.

Domain Requirements Problems

Understandability

- **Requirements** are expressed in the language of the Application **Domain**;
- This is often not **understood** by **Software Engineers** developing the **system**.

Implicitness

- ***Domain*** **Specialists** **understand** the area so well that they do not think of making the **Domain Requirements** explicit.

IEEE Standard for SRS

http://www.cse.yorku.ca/course_archive/2004-05/F/4312/

1 Introduction
- Purpose
- Scope
- Definitions, acronyms, abbreviations
- Reference documents
- Overview

2 Overall Description
- Product perspective
- Product functions
- User characteristics
- Constraints
- Assumptions and Dependencies

3 **Specific Requirements**

Appendices

A.1 Outline for SRS Section 3
Organized by **Use Case**

Index

Identifies the product & application domain

Describes contents and structure of the remainder of the SRS

Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints

Summary of major functions

Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)

All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section

Problems with Natural Languages

Lack of clarity

- Precision is difficult without making the document difficult to read

Requirements confusion

- Functional and Non-Functional Requirements tend to be mixed-up.

Requirements amalgamation

- Several different Requirements may be expressed together.

Example - Naur's **Requirements** Specification

Given an input text consisting of words separated by blanks or carriage return (CR) characters. Transform the text into a line-by-line format where a line should start only where a blank or CR character was in the input text. Each output line should be as full as possible, and no line should have more than a given number (n) of characters.

Peter Naur [Naur69a]

The following **Requirements errors** found in the proposed **SRS**:

r1: No condition specified for the normal end of the program.
r2: No information on the role of the CR character in the output file.
r3: No information on how to handle errors, e.g. words longer than *n* characters.
r4: Unspecified assumption on the sequence and non-changeability of words.
r5: No information on storage formats or medium of input and output.
r6: Open whether multiple consecutive blanks are allowed in the output.

Determining **What** the Client Needs

- Misconception
  - We must determine **what** the Client Wants

- "I know you believe you understood what you think I said, but I am not sure you realize that what you heard **is not** what I meant!"

- We must determine **what** the Client *Needs*
  - *What Client should be saying*
  - *It is **easy to misunderstand what** the Client is saying*

Determining **What** the **Client** Needs

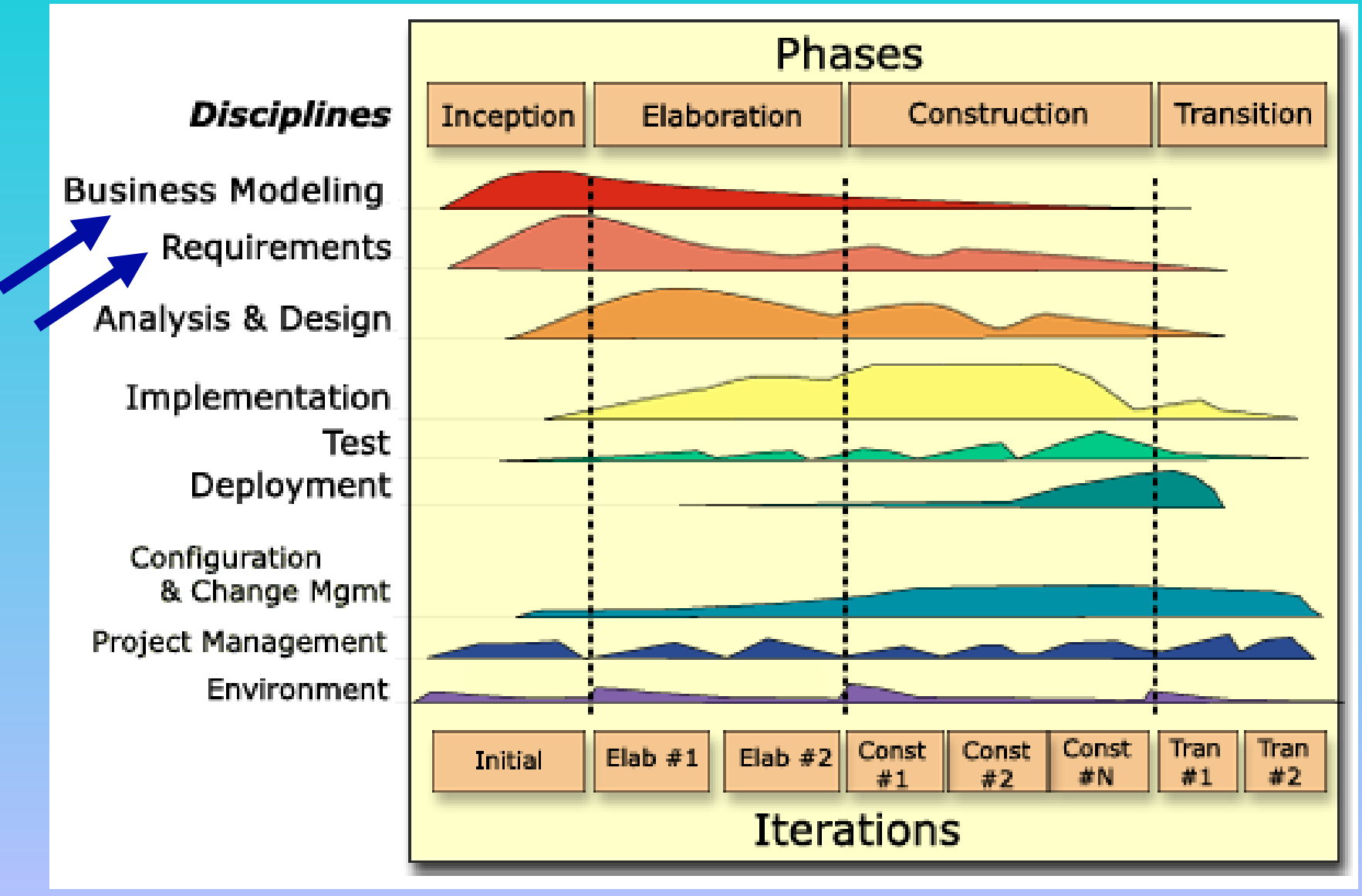
- It is hard for a **Systems Analyst** to visualize a **software product** and its functionality
  - The problem is far worse for the **Client**
- A skilled **Systems Analyst** is needed to **Elicit** the appropriate information from the **Client**
- The **Client** is the **only** source of this information

Determining **What** the **Client** Needs

The solution:

- Obtain initial information from the **Client**
- Use this initial information as input to the **Unified Process**
- Follow the steps of the **U**nified **P**rocess to determine the **Client**'s real Needs

Unified Process

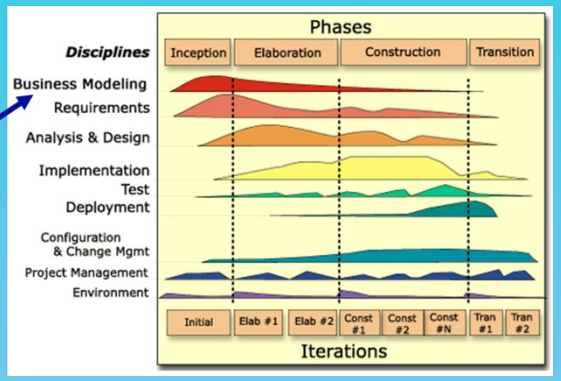


UP Requirements Workflow Steps

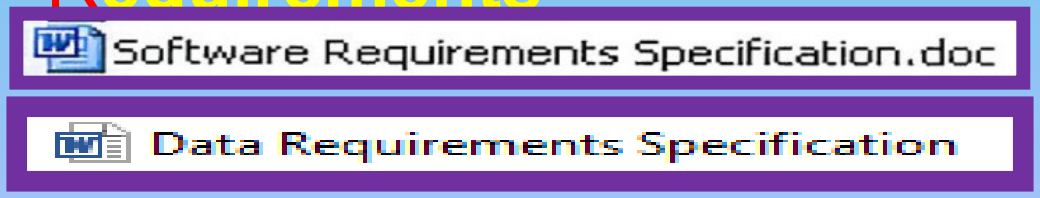
- First, gain an understanding of the *Application Domain* (or *Domain*, for short)
  - The specific environment in which the target product is to operate
- Second, build a Business Model
  - Model the Client's Business Processes

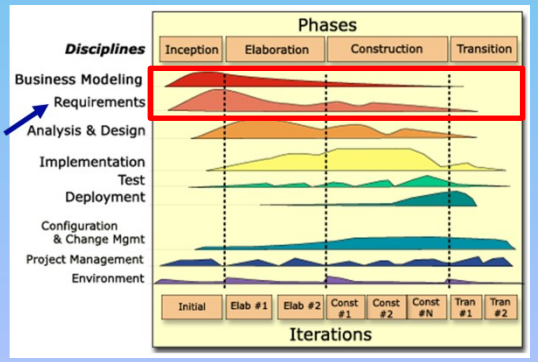
UML

ERD



- Third, use the Business Model to determine the Client's Requirements





- Iterate the above Steps

Definitions

- **Discovering** the **Client**'s **Requirements**
  - **Requirements *Elicitation*** (or **Requirements *Capture***)
  - **Methods** include **Interviews** and **Surveys**

- **Refining** and **Extending** the initial **Requirements**
  - **Requirements *Analysis***

Overview of the Requirements Workflow

- First, gain an understanding of the *Application Domain* (or *Domain*, for short)
  - The specific environment in which the target product is to operate
- Second, build a business model
  - Model the client's business processes
- Third, use the business model to determine the client's requirements
- Iterate the above steps

Step 1: Understanding the Domain

Every Member of the **Requirements Team** must become fully familiar with the **Application Domain**

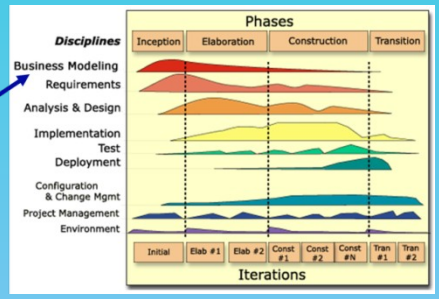
- Correct terminology is essential

Construct a **Glossary**

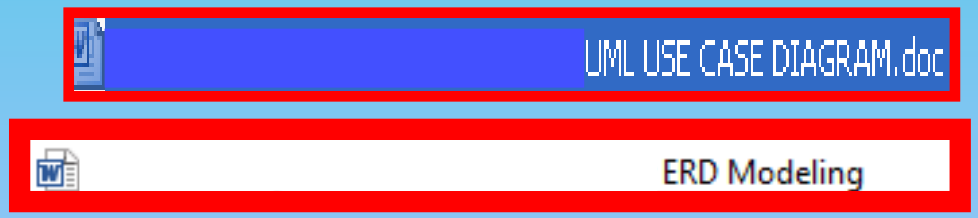
- A list of technical words used in the **Domain**, and their meanings

Overview of the Requirements Workflow

- First, gain an understanding of the *application domain* (or *domain*, for short)
  - The specific environment in which the target product is to operate

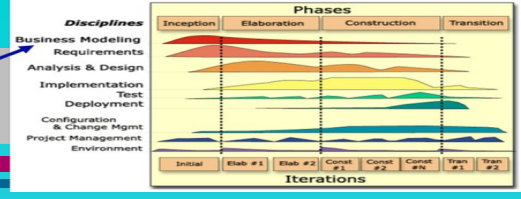


- Second, build a Business Model
  - Model the Client's Business Processes



- Third, use the business model to determine the client's requirements
- Iterate the above steps

Step 2: Business Model



UML USE CASE DIAGRAM.doc

ERD Modeling

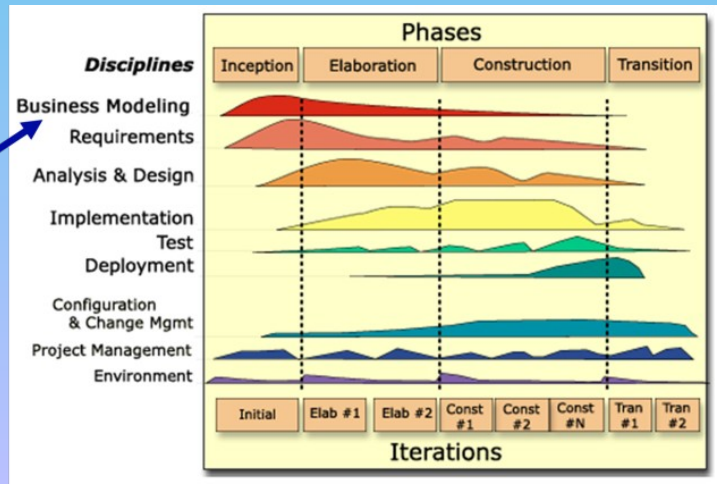
- A ***Business Model*** is a description of the Business **Processes** of an **Organization**

- The ***Business Model*** **gives an understanding** of the **Client**'s **Business** as a whole
  - This knowledge is essential for advising the **Client** regarding computerization

- The **Systems Analyst** needs to **obtain a detailed understanding** of the various **Business Processes**
  - Different techniques are used, primarily Interviewing

Step 2: Interviewing

- The **Requirements Team** meet with the **Client** and **Users** to **extract** all relevant **information**



Interviewing

- There are two types of questions
  - *Close-ended* questions require a specific answer
  - *Open-ended* questions are posed to encourage the **Person** being **Interviewed** to speak out

- There are two types of **Interviews**
  - In a *structured* **Interview**, specific preplanned questions are asked, frequently Close-ended
  - In an *unstructured* **Interview**, questions are posed in response to the answers received, frequently Open-ended

Interviewing

Interviewing is **not easy**

- An **Interview** that is too unstructured will **not yield** much relevant **information**
- The **Interviewer** must be fully familiar with the **Application Domain**
- The **Interviewer** must remain open-minded at all times

After the **Interview**, the **Interviewer** must prepare a written **Report**

- It is strongly advisable to give a copy of the **Report** to the **Person** who was **Interviewed**

Other Techniques

- **Interviewing** is the primary technique
- A **Questionnaire** is useful when the **opinions of hundreds of Individuals** need to be determined
- **Examination** of **Business Forms** shows how the **Client** currently does **Business**

Other Techniques

Direct observation of the **Employees** while they perform their duties can be useful

- Videotape **Cameras** are a modern version of this technique
- But, it can take a long time to analyze the tapes
- **Employees** may view the **Cameras** as an unwarranted invasion of privacy

Overview of the Requirements Workflow

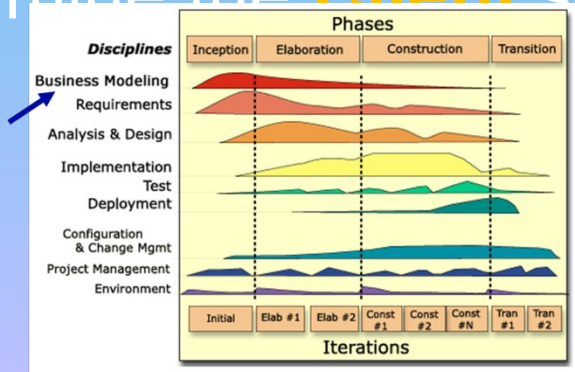
First, gain an understanding of the *application domain* (or *domain*, for short)

- The specific environment in which the target product is to operate

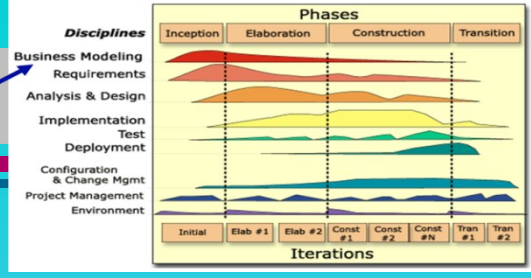
Second, build a business model

- Model the client's business processes

Third, use the **Business Model** to determine the **Client's Requirements**



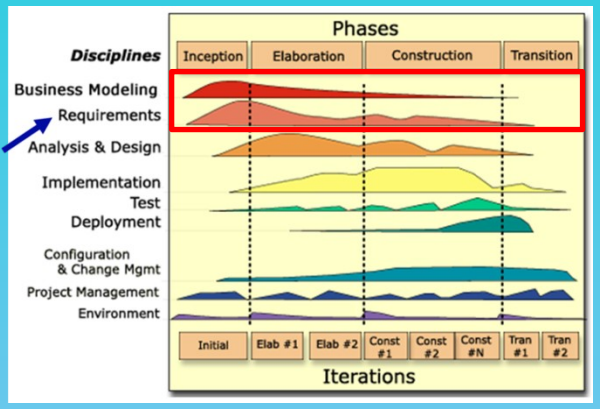
Step 3: Initial Requirements



UML USE CASE DIAGRAM.doc

ERD Modeling

The **Initial Requirements** are based on the **Initial Business Model**



Software Requirements Specification.doc

Data Requirements Specification

Then they are **Refined**

The **Requirements** are **dynamic** — there are frequent changes

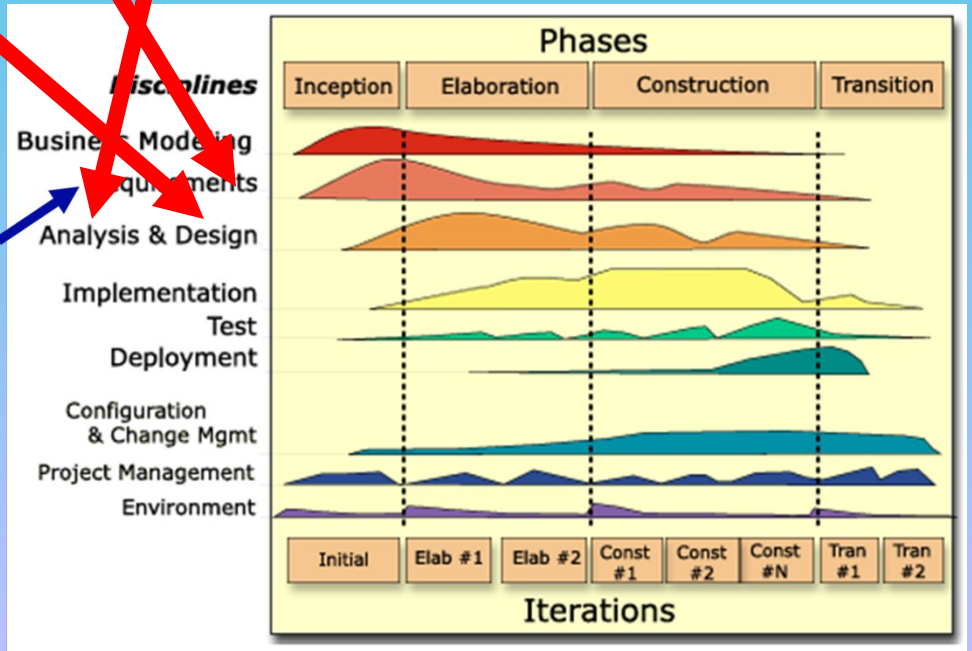
- Maintain a list of likely **Requirements**, together with **Use Cases** of **Requirements** approved by the **Client**

Iterate the above steps

Requirements

- **Functional Requirements** are handled as part of the ***Requirements and Analysis Workflows***

- Some **Non-Functional Requirements** have to wait until the ***Design Workflow***



CASE Tools for the Requirements Workflow

We need **Graphical Tools** for **UML** Diagrams

- To make it easy to change **UML** Diagrams
- The documentation is stored in the **Tool** and therefore is always available

MS WORD

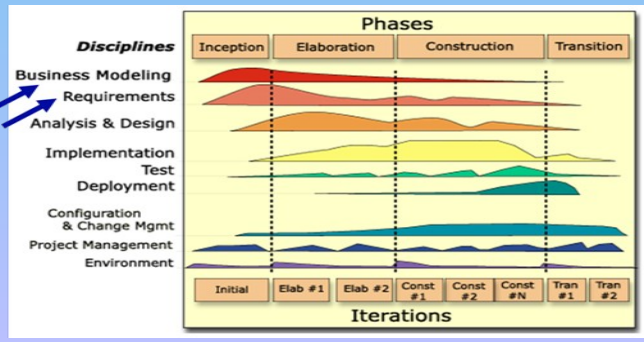
SVN

Such **Tools** are sometimes hard to use

The Diagrams may need considerable "tweaking"

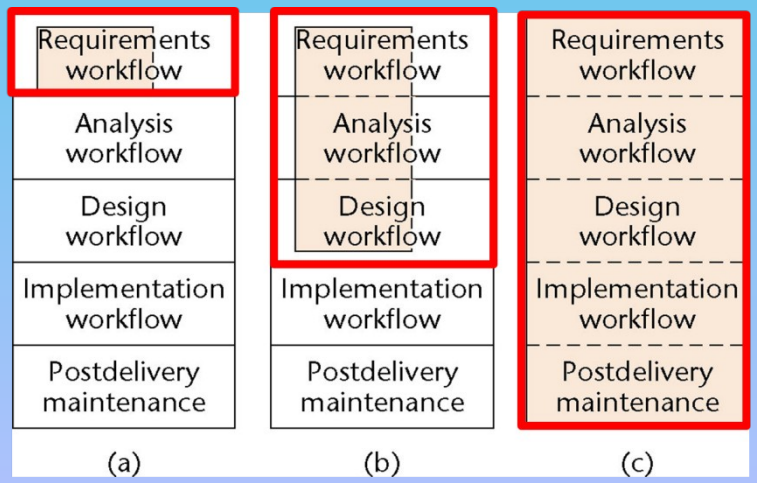
Overall, the strengths outweigh the weaknesses

(**Cost Benefit Analysis**)



CASE Tools for the Requirements Workflow

- Graphical **CASE Environments** support **UML**
  - System Architect (IBM)
  - Software through Pictures
- **O**bject **O**riented **CASE Environments** include
  - **IBM** Rational Rose
  - Together
  - ArgoUML (open source)
  - MagicDraw



(a) **Tool** versus (b) **Workbench** versus (c) **Environment**

CASE Tools for the Requirements Workflow

CASE Tools for the Requirements Workflow

Use Cases and Requirements Management - CaseComplete - Mozilla Firefox

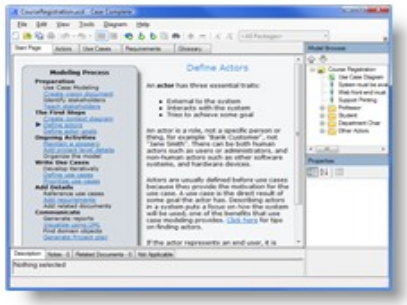
File Edit View History Bookmarks Tools Help

http://www.casecomplete.com/

chipware.com | Use Cases and Requirements Ma...

casecomplete

Download Free Trial | Watch Online Demos | Learn More



Requirements and Use Cases: Simple, Clear, and Effective

CaseComplete is a tool to help you author, manage and share **use cases** and **requirements**. Whether you are part of a diverse team or working solo, an expert author or a novice, CaseComplete will help you write effective use cases and requirements **faster** and **easier**.

If you have ever been frustrated maintaining your use cases and requirements in a word processor or spreadsheet, or in a difficult-to-use requirements management tool, then CaseComplete is for you!

MS WORD

Single Source Project Documentation

CaseComplete produces project documentation for all phases of your project:

- High quality Microsoft Word and HTML reports
- Reports for different stakeholders
- Create Use Case diagrams, GUI mockups/wireframes and flow charts
- Export to UML tools
- Generate Test Plans and Test Cases

Since CaseComplete is easy to install, you can literally *get started today* creating great use cases, requirements and test plans that are customizable to fit your organization's needs. See how easy it is: watch the online demos and then download your 30-day free trial.

Done

http://www.casecomplete.com/

Metrics for the Requirements Workflow

- Volatility and Speed of Convergence are **measures** of **how rapidly** the **Client**'s needs are determined

Software Requirements Specification.doc

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
View Week	24	25 LECTURE 9	26	27 LECTURE 10 & BLUE TEAMS FORMATIONS BEGIN	28	1	2
View Week	3	4 PURPLE TEAMS DELIVERABLES- WHAT & BLUE TEAMS [illegible] MRTs 11	5	6	7	8	9
View Week	10	11 HOLIDAY	12	13 HOLIDAY	14	15	16
View Week	17	18 LECTURE 11	19 Today	20 LECTURE 12 & MRTs 12, 13	21	22	23
View Week	24	25 LECTURE 13 & MRTs 14	26	27 HOW SPMP, [illegible]	28	29	30

March 9, 2015

First iteration SRS

March 30, 2015

> 2 weeks later SRS not signed!

Metrics for the Requirements Workflow

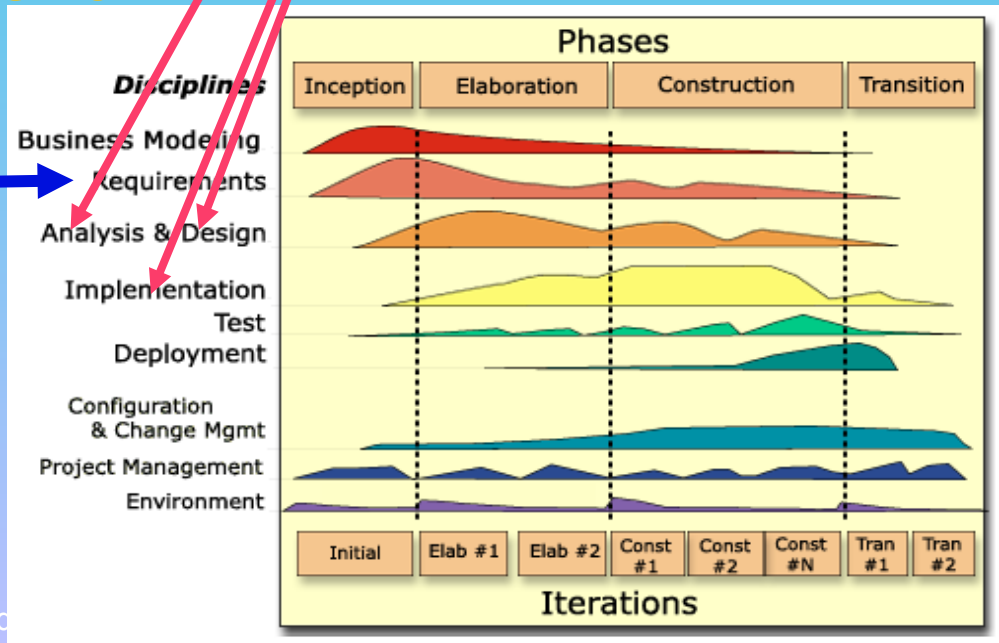
The Number of Changes made *during subsequent Phases*

Changes initiated by the **Developers**

- Too many changes can mean the **Process** is flawed

Changes initiated by the **Client**

- **Moving target problem**



Challenges of the Requirements Phase

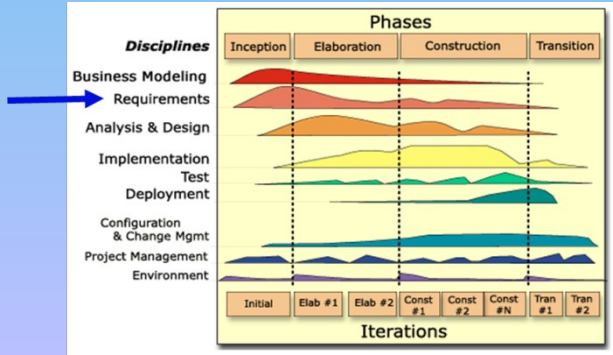
Employees of the **Client** organization often feel **threatened** by computerization

Requirements Team Members must be able to **negotiate**

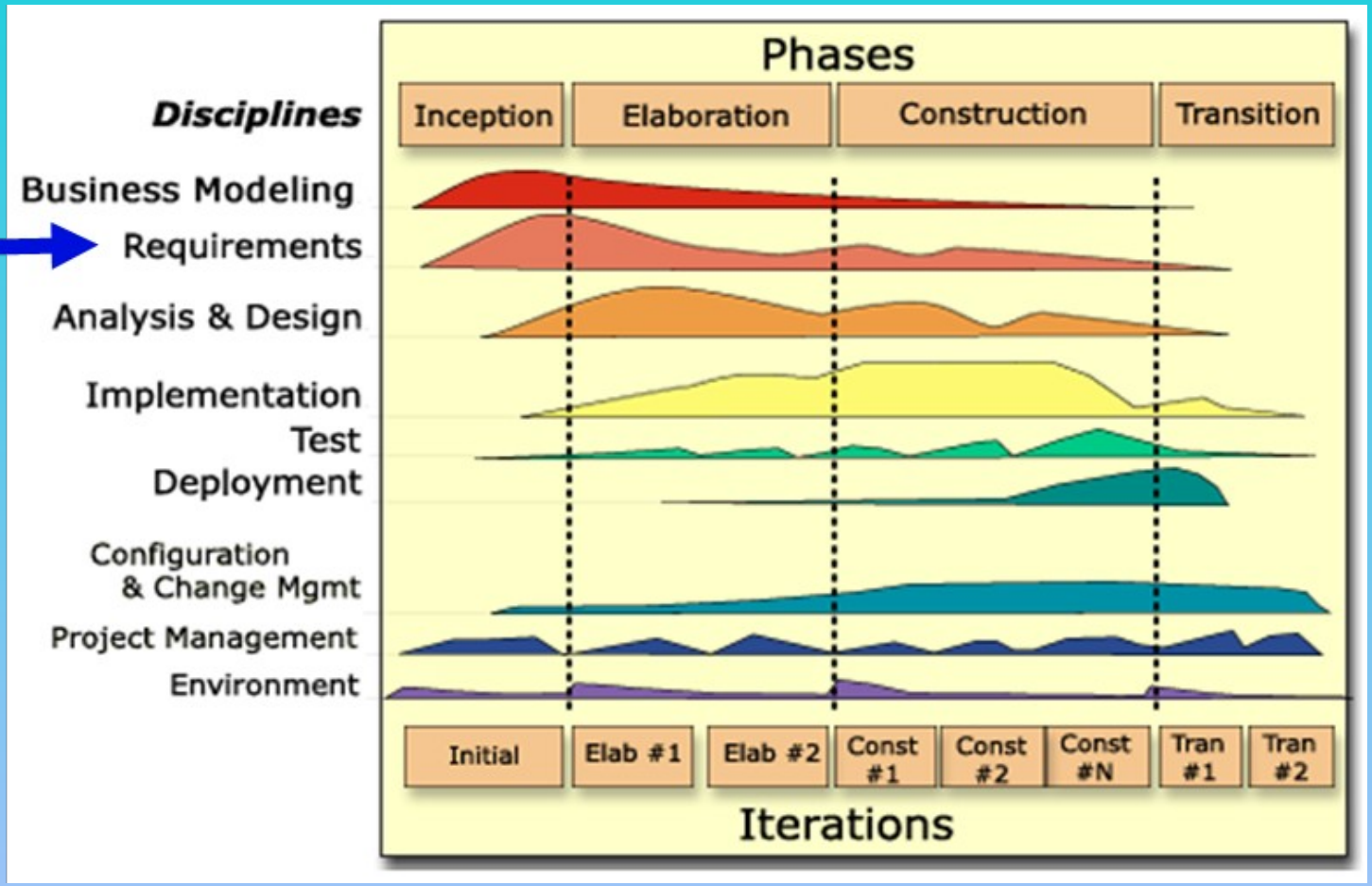
- The **Client**'s needs may have to be scaled down

Key **Employees** of the **Client** organization **may not** have the time for essential in-depth discussions

Flexibility and Objectivity are essential



Requirements Workflow



Overview of the Requirements Workflow

Fill in the _____ and the _____ with words that are appropriate (each word is worth 1 point):

The overall aim of the Requirements Workflow is for the development organization to determine the Client's needs. The first step toward this goal is to gain an understanding of the application domain (or domain, for short), that is, the specific environment in which the target product is to operate. The domain could be banking, space exploration, automobile manufacturing, or telemetry. Once the members of the development team understand the domain to a sufficient depth, they can build a business MODEL that is, UML USE CASE Diagram to describe the Client's business processes. The business MODEL is used to determine WHAT the Client's initial Requirements are. Then iteration is applied. In other words, the starting point is an initial understanding of the domain. This information is used to build the initial business MODEL. The initial business MODEL is utilized to draw up an initial set of the Client's Requirements. Then, in the light of what has been learned about the Client's Requirements, a deeper understanding of the domain is gained; and this knowledge is utilized in turn to refine the business MODEL and hence the Client's Requirements. This iteration continues until the team is satisfied with the set of Requirements. At this point, the iteration stops.

The term Requirements Engineering is sometimes used to describe what is performed during the Requirements Workflow. The process of discovering the Client's Requirements is termed Requirements Elicitation (or Requirements Capture). Once the initial set of Requirements has been drawn up, the process of refining and extending them is termed Requirements Analysis.

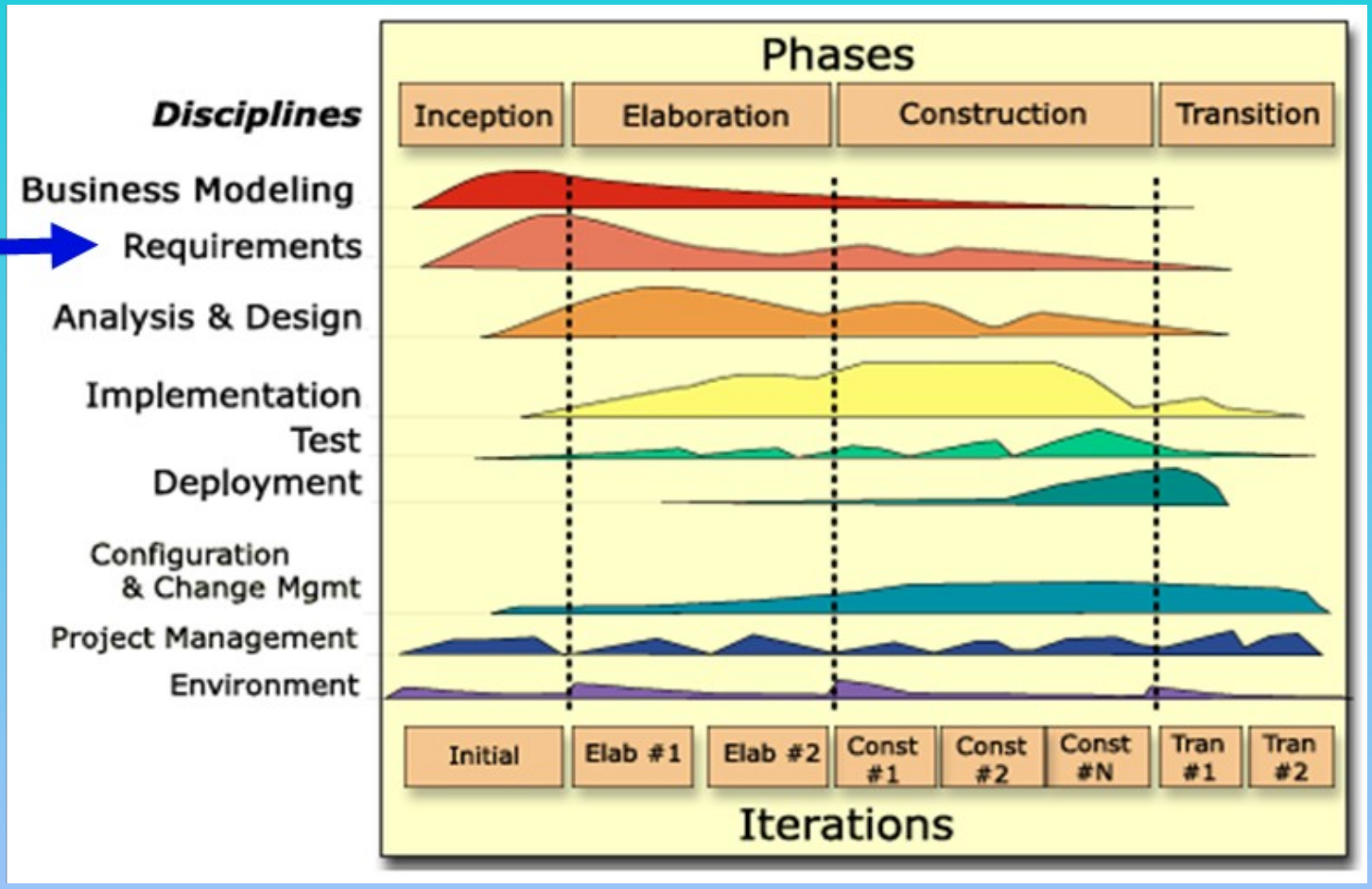
What would you say?

Is there such a thing as "Object Oriented Requirements"?

The **Requirements Workflow** has nothing to do with how the **product** is to be built.

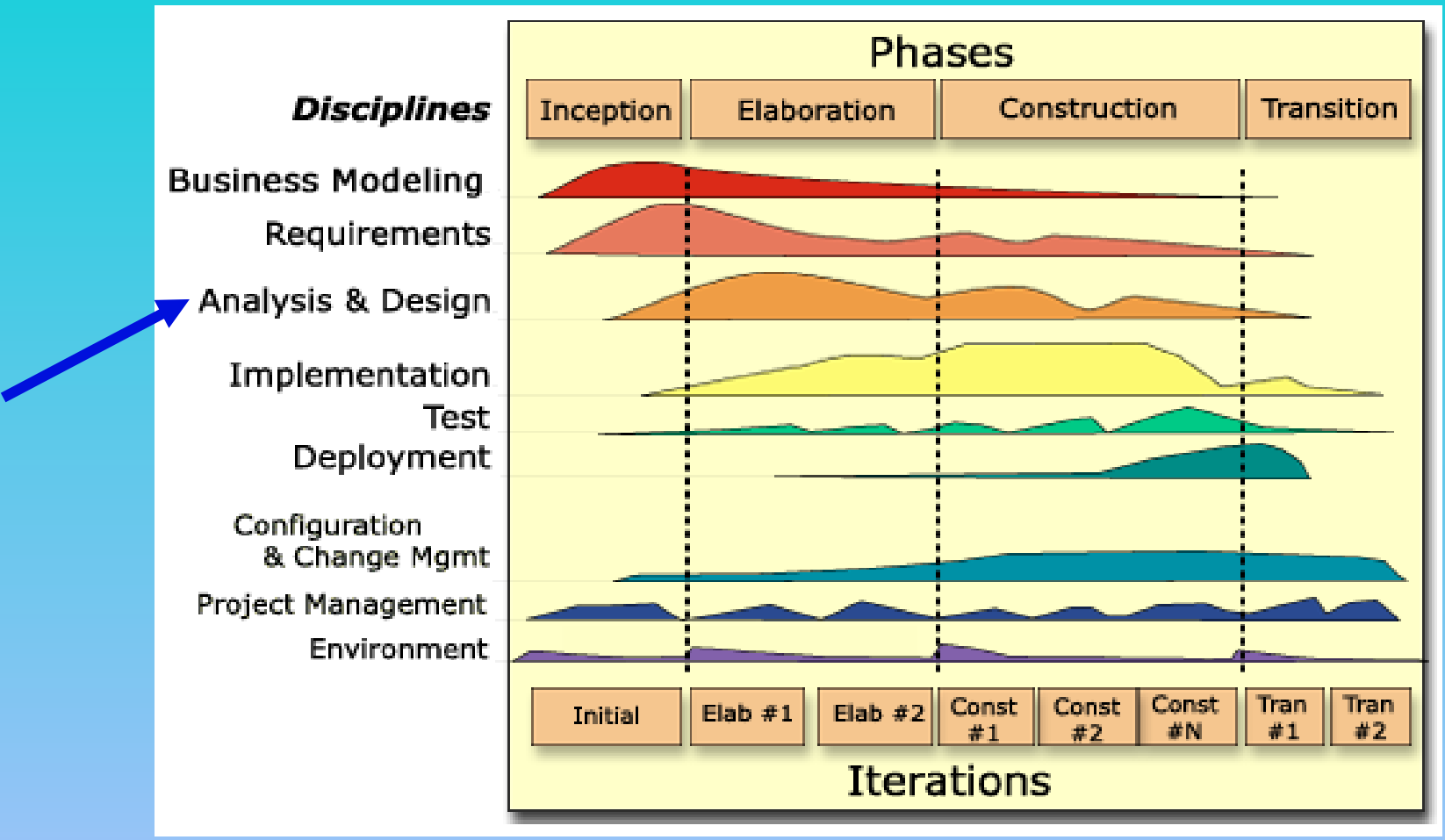
What would you say?

Requirements Workflow



COMPLETED! NEXT???

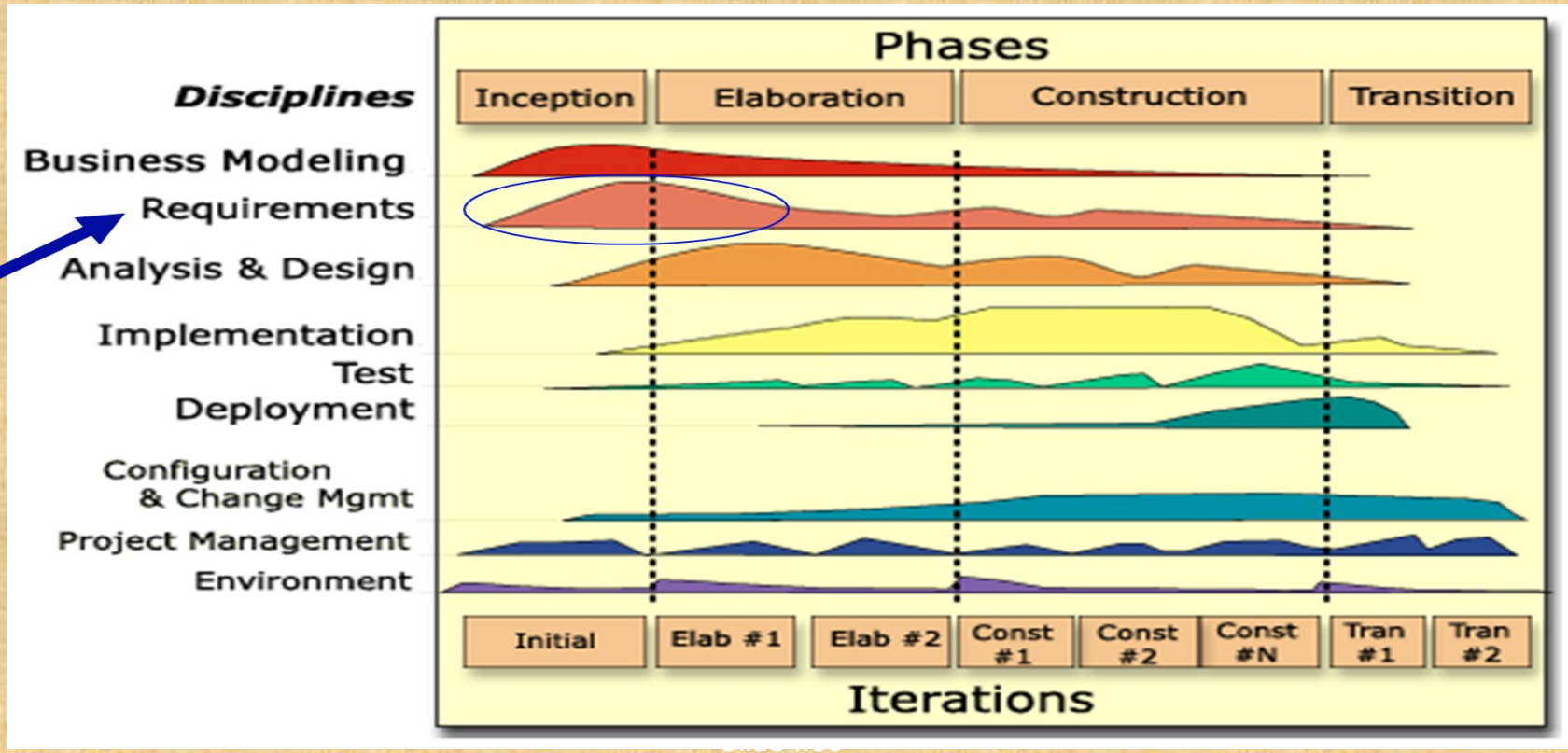
OO Analysis Workflow



From 4:10 to 5:00 – 50 minutes.

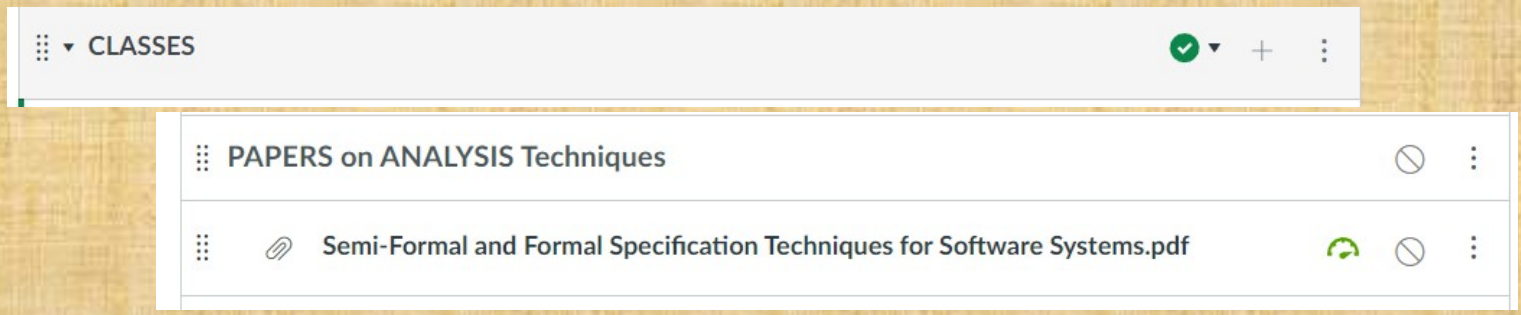
END

Requirements Workflow



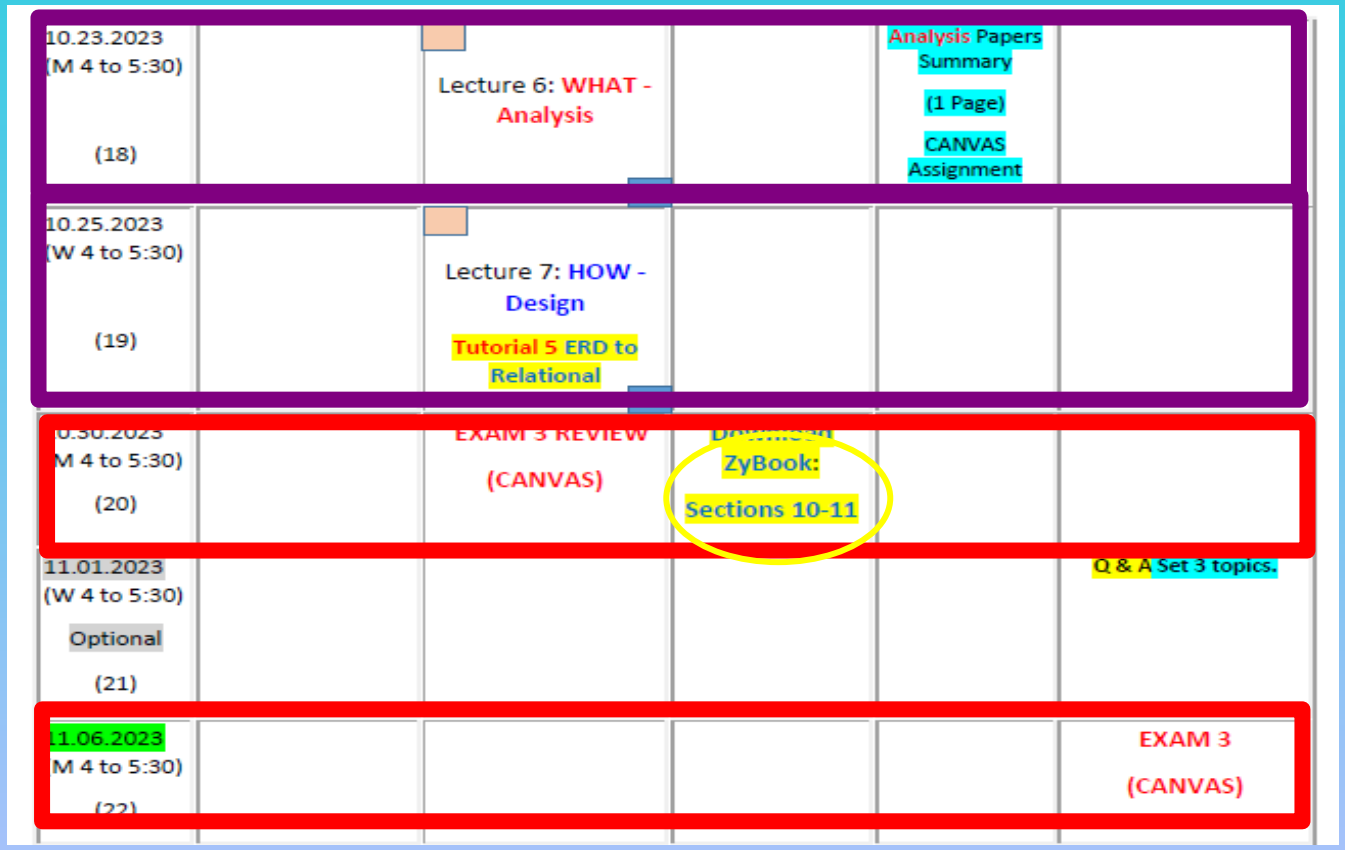
At 5:00.

10.18.2023 (W 4 to 5:30) (17)	Analysis: 1. Semi-Formal and Formal Specification Techniques for Software Systems.pdf	Lecture 5: Requirements		Requirements Papers Summary (1 Page) CANVAS Assignment	

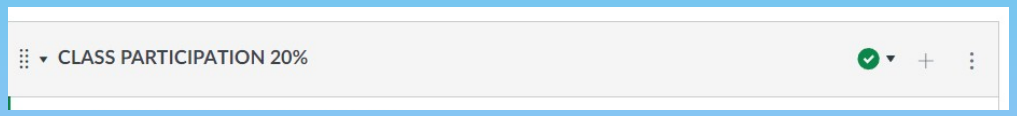


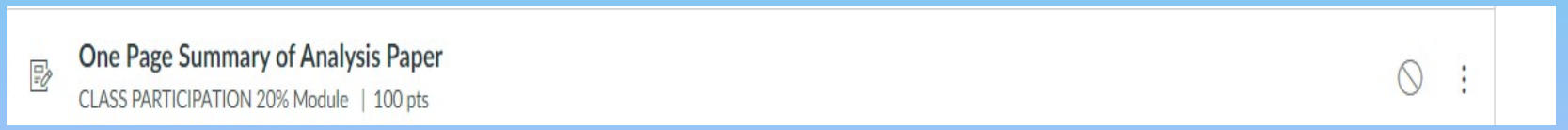
VH– Publish.

NEXT


10.23.2023
(M 4 to 5:30)
(18)
Lecture 6: WHAT - Analysis
Analysis Papers Summary
(1 Page)
CANVAS Assignment
10.25.2023
(W 4 to 5:30)
(19)
Lecture 7: HOW - Design
Tutorial 5 ERD to Relational
M 4 to 5:30)
(20)
EXAM 3 REVIEW
(CANVAS)
ZyBook:
Sections 10-11
11.01.2023
(W 4 to 5:30)
Optional
(21)
Q & A Set 3 topics.
11.06.2023
M 4 to 5:30)
(22)
EXAM 3
(CANVAS)

10.18.2023 (W 4 to 5:30) (17)	Analysis: 1. Semi-Formal and Formal Specification Techniques for Software Systems.pdf	Lecture 5: Requirements		Requirements Papers Summary (1 Page) CANVAS Assignment	
10.23.2023 (M 4 to 5:30) (18)		Lecture 6: WHAT - Analysis		Analysis Papers Summary (1 Page) CANVAS Assignment	

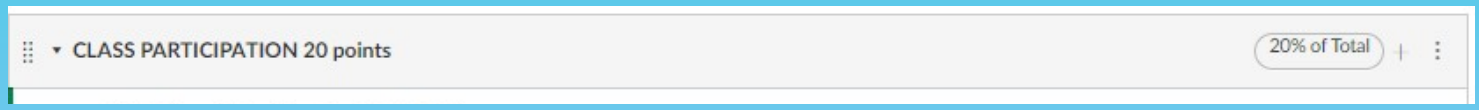




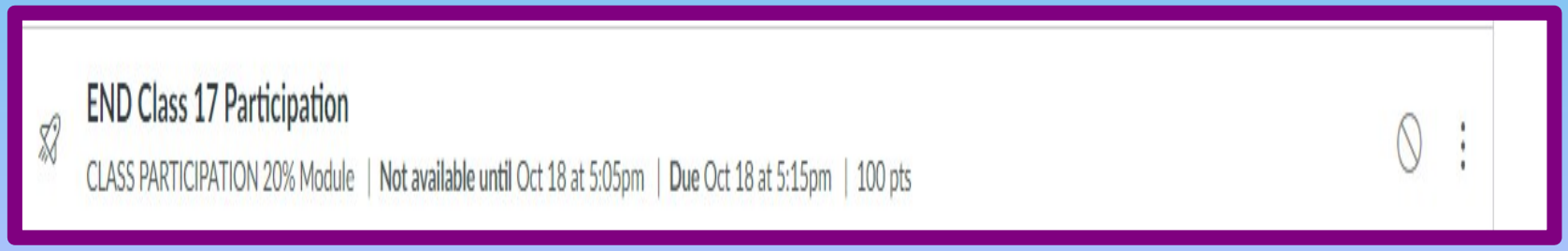
VH– Publish.

From 5:05 to 5:15 – 10 minutes.

10.18.2023 (W 4 to 5:30) (17)	Analysis: 1. Semi-Formal and Formal Specification Techniques for Software Systems.pdf	Lecture 5: Requirements		Requirements Papers Summary (1 Page) CANVAS Assignment	



PASSWORD: I AM IN TEAMS



VH– Publish.

At 5:15.

End Class 17

VH, Download Attendance Report
Rename it:
10.18.2023 Attendance Report FINAL

VH, upload Class 17 to CANVAS