SOFTWARE ENGINEERING CS-UH 2012, SPRING 2025

Lecture 4: Requirements Documentation

Dr. Mohamad Kassab, m.kassab@nyu.edu

LESSON PLAN

- Forms of Representing Requirements.
- Problems with Requirements.
- The Requirements Document
- IEEE 29148 qualities for requirements.
- Additional Best Practices to represent requirements.

HOW TO WRITE DOWN REQUIREMENTS?

- **The Three Forms of Requirements Representation**
- Various techniques can be used to describe functionality in any system. Generally, there are three forms of requirements representation
  - **Informal**
  - **Semi-Formal**
  - **Formal**
- Requirements specifications may not strictly adhere to one or another of these approaches, but usually, they contain elements of at least two of these approaches (informal and one other).

INFORMAL REPRESENTATION

- This form includes natural language (i.e., human languages), flowcharts, ad hoc diagrams, etc.
- Natural language is in fact the most common form of requirements for projects in the industry.
- In a recent survey by Kassab and Laplante , it was found that 69% of projects reported that requirements were expressed in terms of natural language, that is, informally.
- There are different ways to express requirements in natural languages. For example:
  - Shall Statement
  - Use Cases
  - User Stories
  - Quality Scenarios

Shall Statement

You are expected to utilize the Shall Statement form for your SRS document so let's take a look at this closely.

Click the words below in [ ] to show the standard requirement breakdown of the Shall Statement form.

[identifier] The [noun phrase] shall (not) [verb phrase] [constraint phrase]

As an example of a requirement that includes a measurable target, let's return to the Building Automation System example:

1. 3 The system shall generate an emergency alarm notification within 10 seconds of identification of an emergency incident.

Identifier	Noun Phrase	Verb Phrase	Constraint Phrase
Where [identifier] is a reference ID to be used to identify requirements in some way.	[Noun phrase] describes the main subject of the requirement,the shall (or shall not) statement indicates a required or prohibited behavior	[Verb phrase] indicates the actions of the requirement	[Constraint phrase] place measurable constraints on performance for functional requirements whenever possible

USE CASES

- A use case is a group of related scenarios that provide value to the user of a system where a scenario is one of many paths within a use case.

- Use cases of the system can be visualized through the UML Use Case diagram. UML diagrams are considered a semi-formal notation.

- We will discuss more the use cases later in the lecture.

- The following slide shows an example of a use case that describes how users would interact with a smart home system.

A USE CASE EXAMPLE

- **Primary Base Use Case**: Controlling home temperature.
- **Precondition**: A user has a number of smart sensors and actuators installed in their home.
- **Main Success Scenario**:
  - A user with a cell phone wants to know the temperature in their home.
  - They connect to the cloud to query their home sensors.
  - After verifying their identity, the system presents a set of options based upon the sensor/actuator package installed in their home.
  - The user selects the temperature option and received the temperature of their home from the cloud.
  - The user decides that they would like to cool the home prior to his arrival.
  - The user sends a command to the home via the cloud to turn on the air conditioner along with a specified set temperature.

User Stories

User stories are a common unit of requirement in most agile methodologies. Each user story represents a feature desired by the customer. User stories are usually written by the customer on index cards, though the process can be automated via wikis or other tools. Click on each section of the sample user story below for the Baggage Handling system to learn more.

Title: Detect Security Threat		
Acceptance Test: detSecThrt	Priority: 1	Story Points: 3
When a scanned bag has been determined to contain an instance of a banned item, the bag shall be diverted to the security checkpoint conveyor. The security manager shall be sent an email stating that a potential threat has been detected.		

SEMI-FORMAL

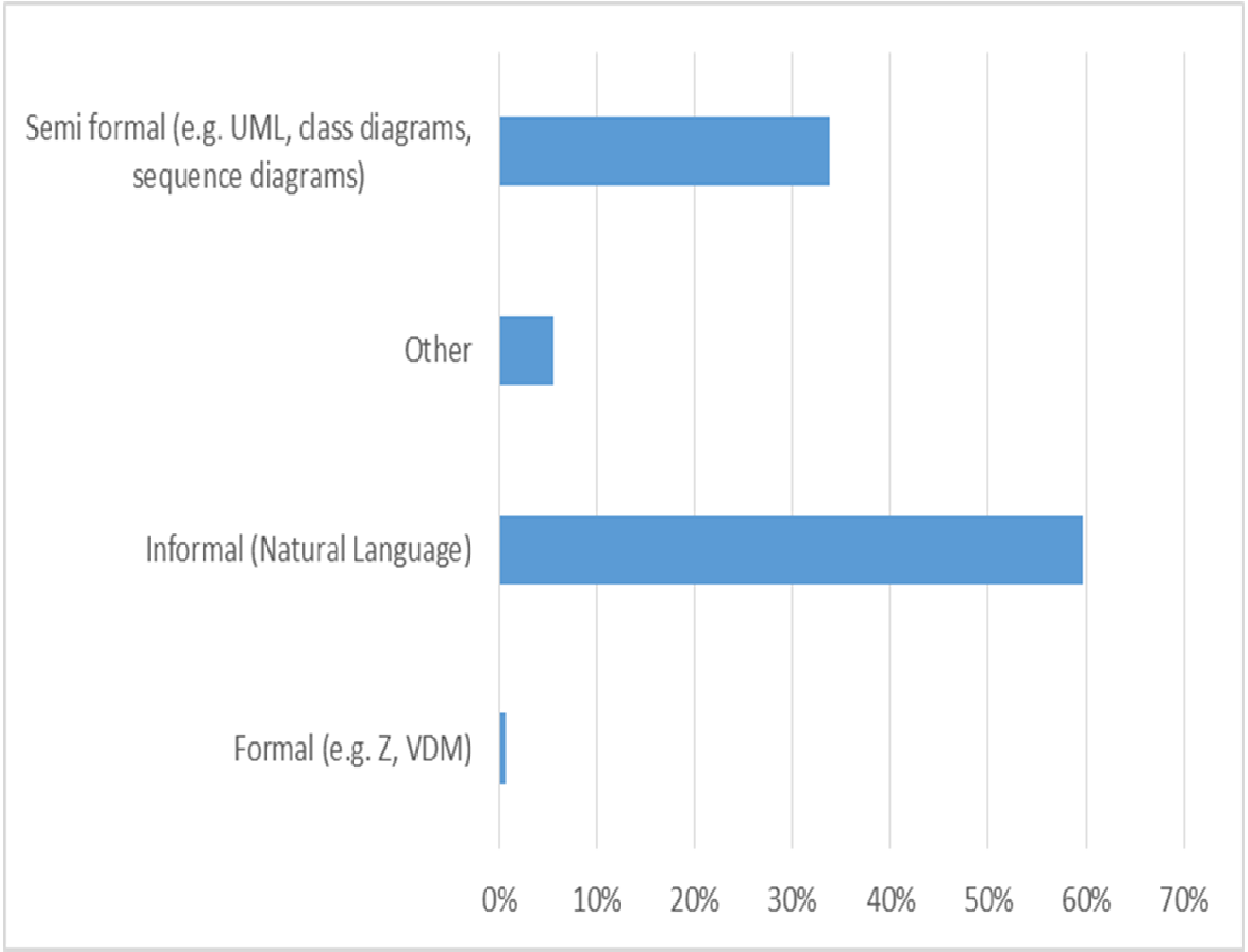
- This form includes those that, while appearing informal, have at least a partial formal basis.
- For example, many of the diagrams in the Unified Modeling Language (UML) or Systems Modeling Language (SysML) family of metamodeling languages including the use case diagram are semiformal or can be formalized.
- The UML is a set of modeling notations that are used in software and systems engineering requirements specification, design, systems analysis, and more.
- In this course, we will introduce a number UML diagrams at different phases of the development.

FORMAL

- Formal notations such as Z and VDM rely on mathematical formalisms to represent requirements.

- We won’t explore these further in this course.

Requirements Representation in practice



Kassab, M., & Laplante, P. (2022). The current and evolving landscape of requirements engineering in practice. *IEEE Software*, *39*(5), 76-83.

OUR FOCUS TODAY…

- In this lecture, we will focus more on informal representation of requirements.
- In particular, we will focus on requirements representation in the form of “***Shall Statements***” and “***Use Cases***”.
- You will use a combination of these formats moving forward in your assignments and project deliverables.

SHALL STATEMENTS FOR MEDICAL RECORD SYSTEM

•**Patient Information Management:**
•The system **shall** allow authorized users to create, view, update, and delete patient records.
•The system **shall** store patient information, including but not limited to personal details, medical history, medications, allergies, and treatment plans.

•**Access Control:**
•The system **shall** require users to authenticate using a username and password before accessing the system.
•The system **shall** assign roles to users, with each role having specific permissions (e.g., administrator, physician, nurse).

SHALL STATEMENTS FOR MEDICAL RECORD SYSTEM

•**Audit Trail:**
•The system **shall** log all access and modifications to patient records, including the user ID, timestamp, and type of action performed.
•The system **shall** allow authorized users to view audit logs for compliance and monitoring purposes.

•**Data Security:**
•The system **shall** encrypt patient data both in transit and at rest to ensure confidentiality.
•The system **shall** automatically lock user sessions after a period of inactivity to prevent unauthorized access.

•**Interoperability:**
•The system **shall** support the exchange of patient information with other healthcare systems using HL7 or FHIR standards.
•The system **shall** allow the import and export of medical records in standard formats, such as XML or JSON.

What is wrong with these requirements?

The system shall not permit access to unauthorized users.

The system shall permit access only to authorized users.

The system shall be completely reliable.

The number of pieces of baggage lost by the system shall be minimal.

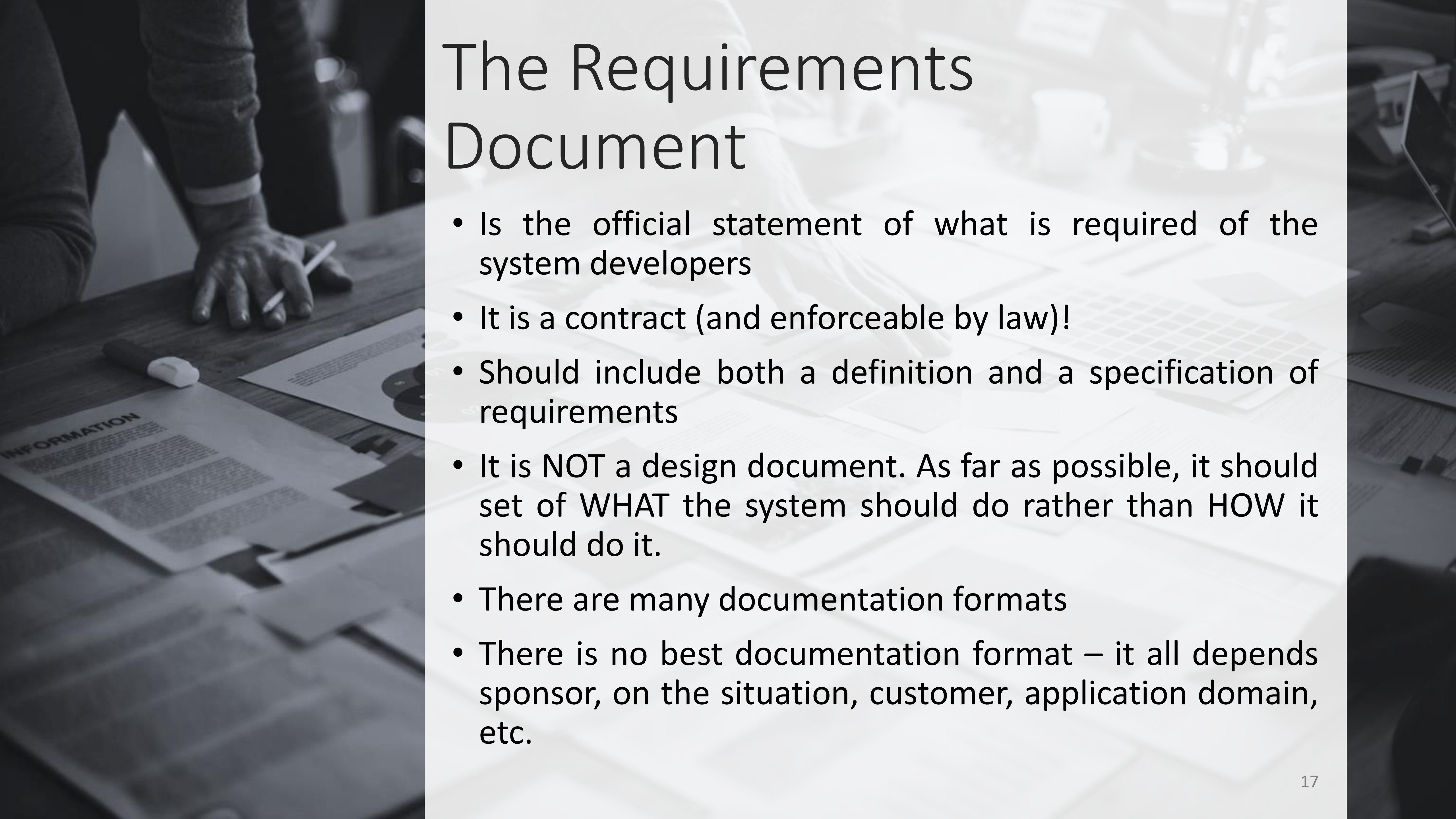
Problems with Requirements

Requirements can be

- **Confusing**
- **Extraneous**
- **Duplicated**
- **Conflicting**
- **Missing**

Requirements analysis – determine if the above

Requirements agreement – techniques to deal with these



The Requirements Document

- Is the official statement of what is required of the system developers
- It is a contract (and enforceable by law)!
- Should include both a definition and a specification of requirements
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.
- There are many documentation formats
- There is no best documentation format – it all depends sponsor, on the situation, customer, application domain, etc.

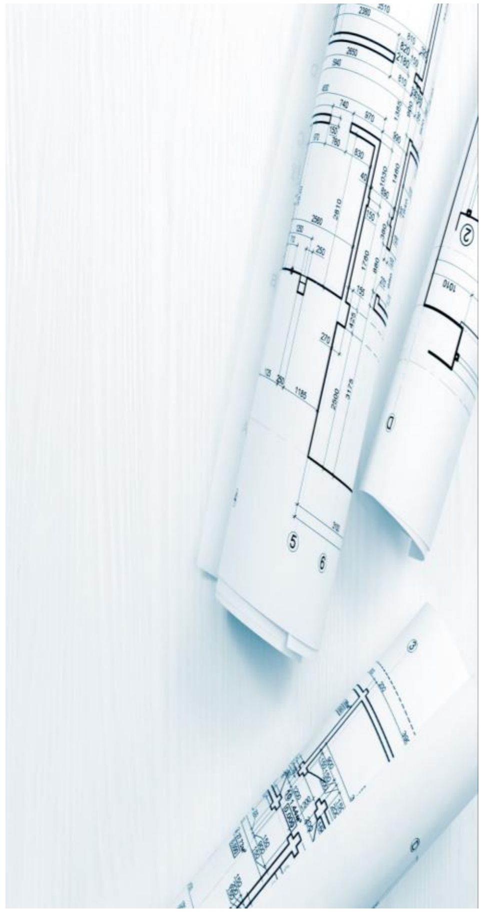
Users of a Requirements Document

Customers	Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements
Managers	Use the requirements document to plan a bid for the system and to plan the system development process
Developers	Use the requirements to understand what system is to be developed
Test engineers	Use the requirements to develop validation tests for the system
Maintenance engineers	Use the requirements to help understand the system and the relationship between its parts

Don't forget lawyers, judges, plaintiffs, juries, district attorneys, arbiters, mediators, etc., who will view the SRS as a legal document in the event of disputes.

ISO/IEC/IEEE Standard 29148

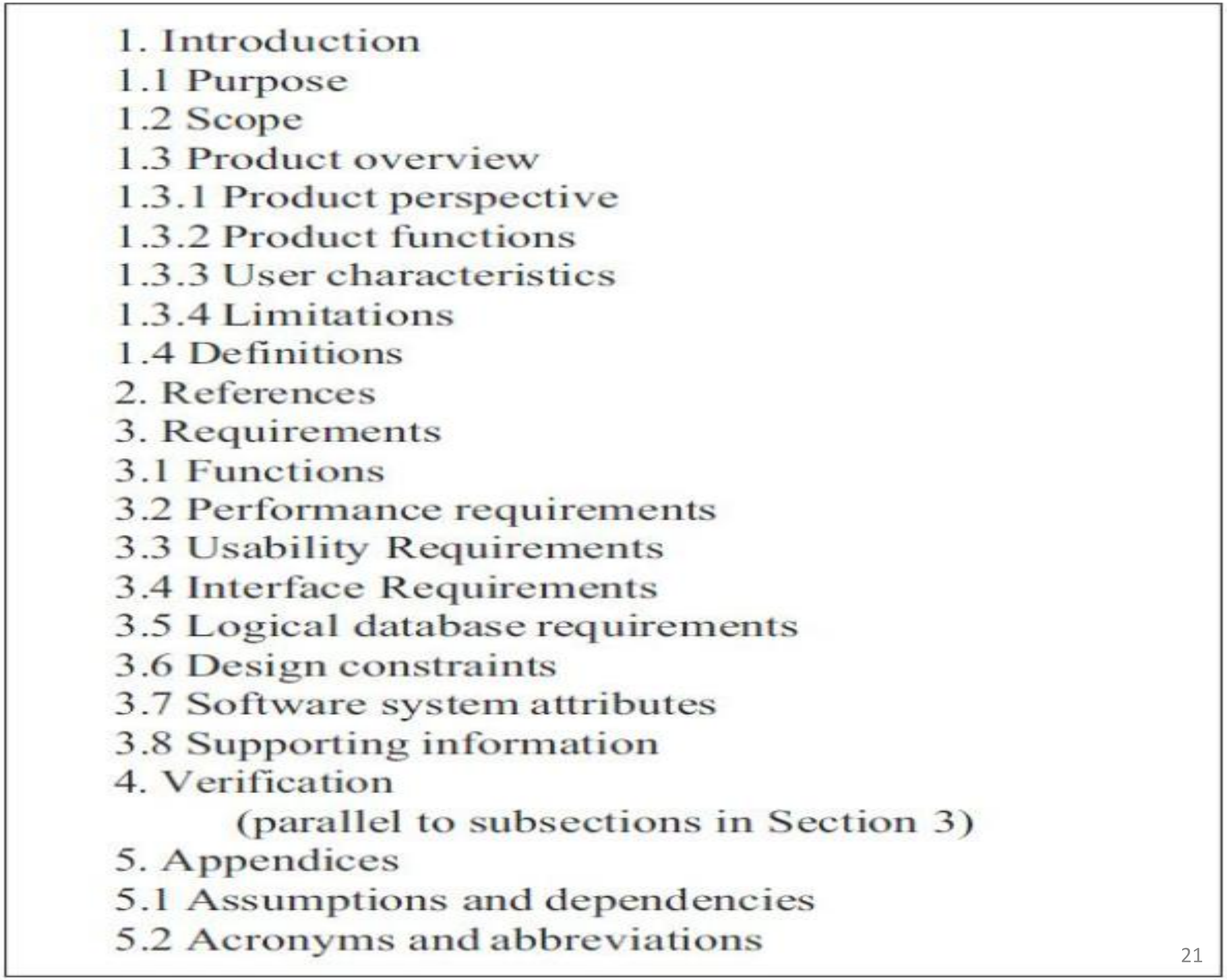
- In 2011, three international standardization bodies, the International Standards Organization (ISO), the International Electrotechnical Commission (IEC), and the Institute for Electrical and Electronics Engineers (IEEE) jointly released the ISO/IEC/IEEE Standard 29148 (2011). In 2018, the second edition of ISO/IEC/IEEE Standard was released. The standard is based on a model that produces a document that helps:
  - to establish the basis for agreement between the acquirers or suppliers on what the product is to do;
  - to force a rigorous assessment of requirements before design can begin and reduce later redesign;
  - to provide a realistic basis for estimating product costs and risks, and schedules organizations to develop validation and verification plans;
  - to provide an informed basis for deploying a product to new users or new operational environments;
  - to provide a basis for product enhancement.



ISO/IEC/IEEE Standard 29148

- The IEEE 29148 outline is particularly beneficial to the requirements engineer because it has been widely deployed across a broad range of application domains
- Because of all the above, our course has adopted the IEEE 29148 standard as a template for your team project.
- In particular, you will need to follow the template for your SRS document (Delivery 2). The template is an adaptation of IEEE 29148.

Table of Contents for an SRS



Organizational Options for section 3.1

- System mode (e.g., navigation, combat, diagnostic)
- User class (e.g., user, supervisor, diagnostic)
- Object (by defining classes/objects, attributes, functions/methods, and
- messages)
- Feature (describes what the system provides to the user)
- Stimulus (e.g., sensor 1, sensor 2, actuator 1, …)
- Functional hierarchy (e.g., using a top-down decomposition of tasks)

IEEE Std 29148 Considerations for requirements

For a single requirement

- Singular
- Feasible
- Unambiguous
- Complete
- Consistent
- Verifiable
- Traceable

IEEE Std 29148

- For a set of requirements
  - Complete
  - Consistent
  - Bounded
  - Affordable

Singular

- Each requirement specifies a single and have no conjunctions.
    - 3.1.1 The pump control unit shall start the submersible pump motors to prevent the wet well from running over and stop the pump motors before the wet well runs dry.
- versus:
    - 3.1.1.1 The pump control unit shall start the submersible pump motors to prevent the wet well from running over.
    - 3.1.1.2 The pump control unit shall stop the pump motors before the wet well runs dry.
- Conjunctions can be used if they describe a single behavior
    - 3.2.6 The light emitted by the display shall be either red or yellow.

Feasible

- A requirement that can be satisfied with current technology and cost constraints,
- Not a ridiculous requirement.
- Examples
  - The system shall consume no power.
  - The length of the beam shall be less than 0 feet.

Unambiguous

- A requirement is unambiguous if it can have only one interpretation.
- Example
  - If the valve is open and the drain is open, then it should be closed.
- Conjunctions can cause ambiguity and that is why they should be avoided.

Complete

- A requirement is complete “to the extent that all of its parts are present and each part is fully developed.”
    - Switch 1 shall have two positions, one position is “on”.
- Same property for a set of requirements
    - E.g. no “TBD”

Consistent

- A requirement is consistent if it does not contradict itself (internal) or a prevailing standard or law (external).
  - If the lever is in position 1 then it shall open and close the valve.
- Same for a set of requirements (e.g.)
  - 3.1 If the lever is in position 1, then valve 1 is opened.
  - 3.2 If the lever is in position 1, then valve 1 is closed.

Verifiable

- Satisfaction of the requirement can be demonstrated
    - The system shall be user friendly
- Usually needs metrics associated with the requirement
- Important to always envision the acceptance test along with the requirement
- Goal-question-metric paradigm useful

Traceable

- Numbered or indexed somehow
- Essential quality for effective communications ease of modification, and legal considerations.
- Stored in a requirements management tool

Bounded

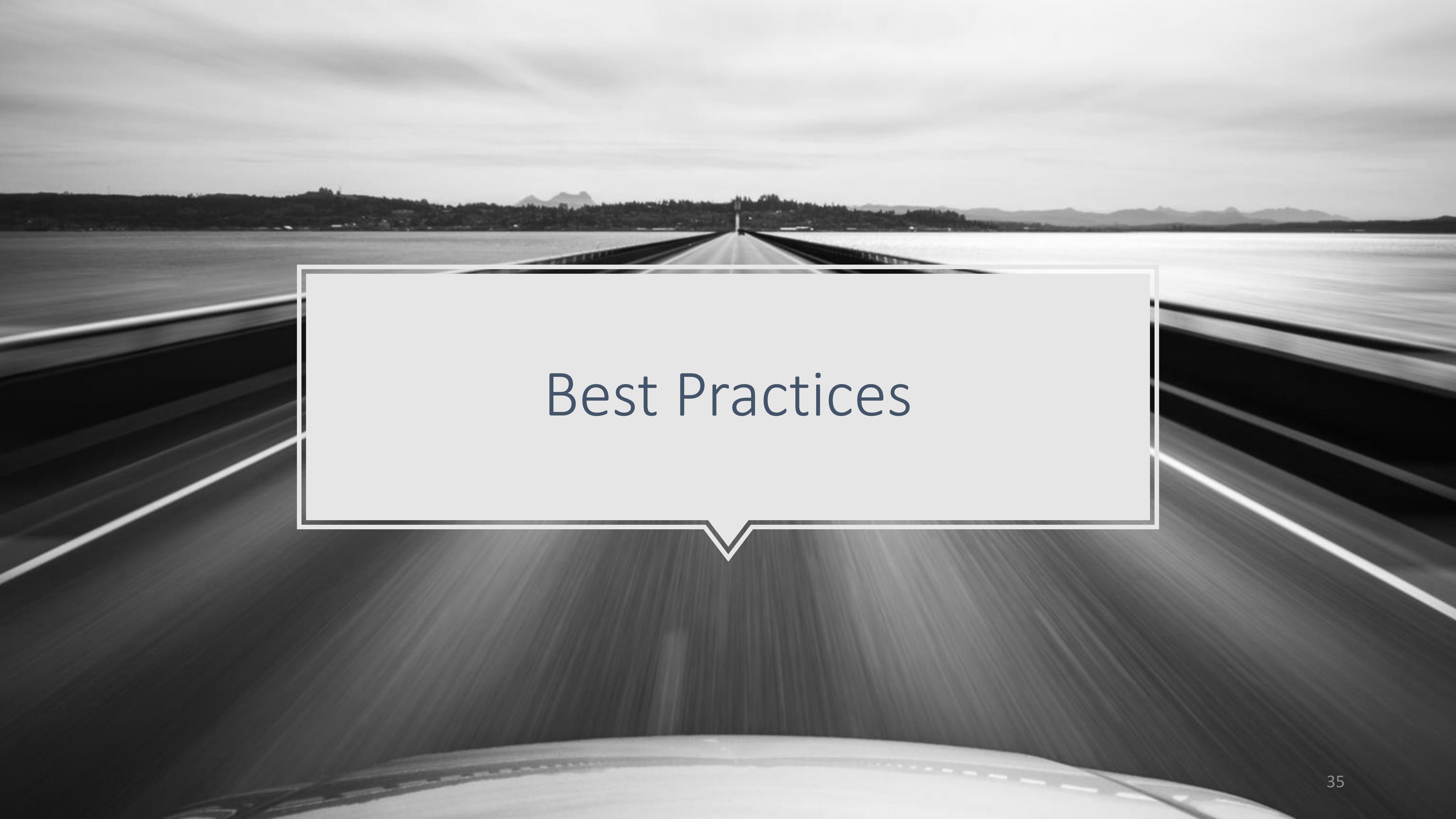
- A set of requirements is bounded if it does not exceed the scope of the system intent.
- Important to avoid scope creep.
- Refer to mission statement (or conops) to avoid scope creep;
  - e.g. does this requirement fit the mission?

Affordable

- For a set of requirements meets budgetary constraints.
- Affordability analysis discussed later

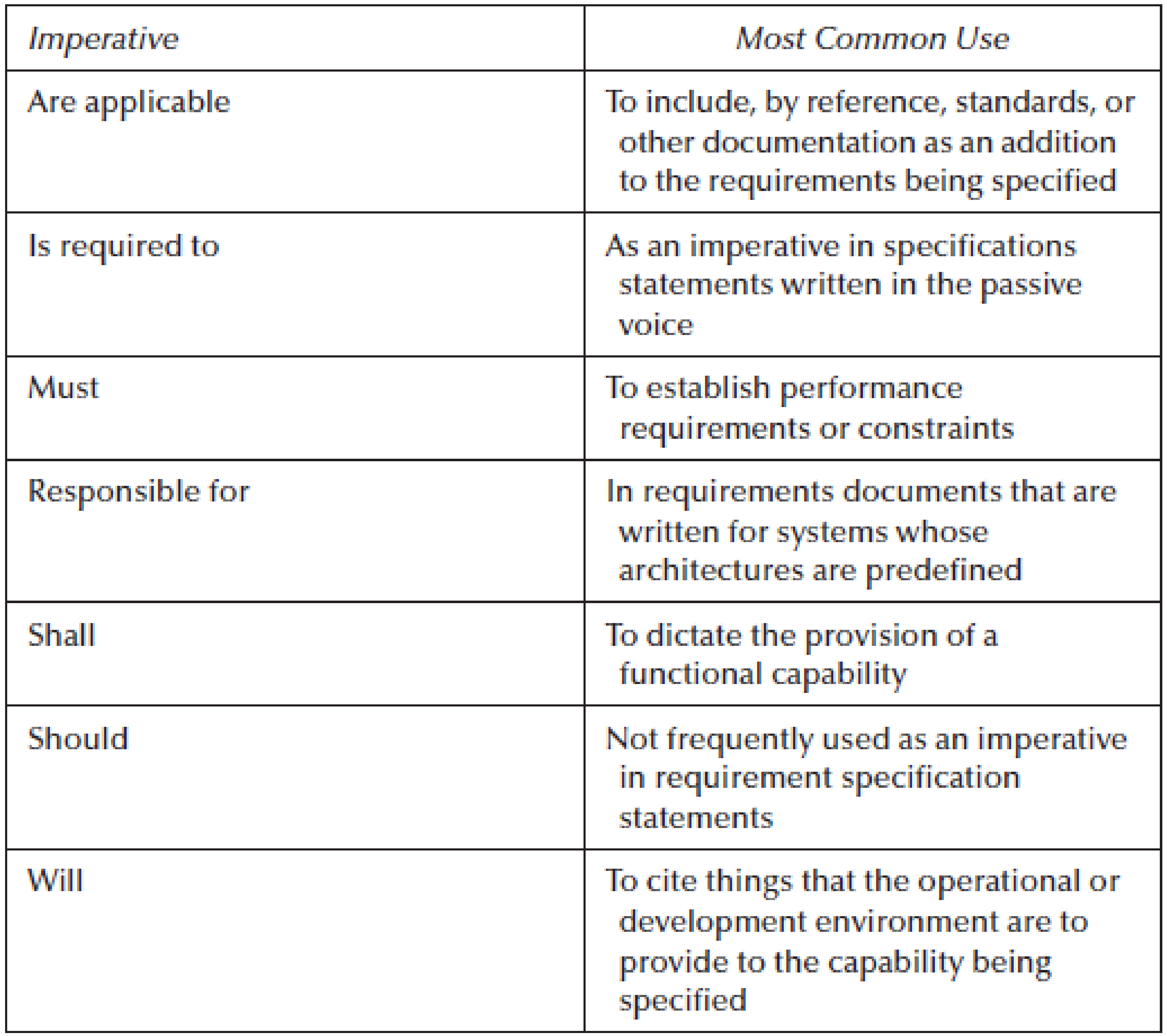
Ranked

- Not a 29148 mandated quality, but…
- Important for design decisions, tradeoff analysis, budget negotiations and more
- Rank can be ordinal (1, 2, 3, …) or subjective (high, medium, low), or based on need (mandatory, desirable, optional)
- Two requirements can hold the same rank
- Ranking is part of the requirements agreement process



Best Practices

Shall or Must?



Imperative	*Most Common Use*
Are applicable	To include, by reference, standards, or other documentation as an addition to the requirements being specified
Is required to	As an imperative in specifications statements written in the passive voice
Must	To establish performance requirements or constraints
Responsible for	In requirements documents that are written for systems whose architectures are predefined
Shall	To dictate the provision of a functional capability
Should	Not frequently used as an imperative in requirement specification statements
Will	To cite things that the operational or development environment are to provide to the capability being specified

Best Practices

- Use language in a consistent way.
- Use “shall” for mandatory requirements,
- Use “should” for desirable requirements
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of technical language unless it is warranted.

Shall Not Requirements

Can be rewritten in positive form. Ex

- The system shall not harm humans.

Alternative

- The system shall cause humans no harm.

Avoid The Following

Hedging words/clauses:	Conjunctions:	Speculative terms:	Vague terms
• If necessary • Almost • Probably • Maybe • Might be	• And • Or • But	• Perhaps • sometimes	• User friendly • generally

Bad Habits in Requirements Specification

- Mixing of operational and descriptive specifications.
- Combining low-level hardware functionality and high level systems and software functionality in the same functional level.
- Omission of timing information.

Best Practices

Bad:

- The system shall be completely reliable.
- The system shall be modular.
- The system will be fast.
- Errors shall be less than 99%.

Better:

- Response times for all level one actions shall be less than 100 ms.
- The cyclomatic complexity of each module shall be in the range of 10 to 40.
- 95% of the transactions shall be processed in less than 1 s.
- MTBF shall be 100 hours of continuous operation.

Weak Phrases

Lead to uncertainty due to multiple interpretations:

- Adequate
- As a minimum
- As applicable
- Be able to
- Be capable
- But not limited to
- Capability of
- Capability to
- Effective
- If practical
- Normal
- Provide for
- Timely
- TBD

Source: Rosenberg

Use Cases

Let’s focus now on how to write requirements in **Use Cases**.

Online Checkout (Main Success Scenario)

1. **Customer** navigates to the **Shopping Cart** page and clicks **“Proceed to Checkout.”**
2. **System** displays the **Order Review** page, showing items, quantities, and pricing.
3. **Customer** confirms or edits **delivery address** and **payment details** (credit card, etc.).
4. **System** processes the payment by calling a **Payment Service**.
5. **Payment Service** returns **approval** to the System.
6. **System** finalizes the order, **generates an order ID**, and **sends a confirmation** email to the Customer.
7. **Customer** sees a **confirmation message** on the screen with the **order details**

Online Checkout (Alternative Scenarios (Extensions))

(1a) **Customer Not Logged In:**
1a.1. The **System** detects the user is not logged in.
1a.2. The **System** prompts the user to **log in** or **create an account**.
1a.3. Upon successful login, the **use case** continues from **Main Success Scenario step**

(3a) **Invalid Payment Details:**
3a.1. The **System** detects invalid or **incomplete** payment details.
3a.2. The **System** displays an **error message** asking the Customer to re-enter correct payment details.
3a.3. After re-entry, the **use case** resumes from **Main Success Scenario step 3**.

(4a) **Payment Declined:**
4a.1. The **Payment Service** returns a **declined** status (e.g., insufficient funds).
4a.2. The **System** notifies the Customer and requests an alternate **payment method**.
4a.3. Customer either **cancels** the order or **enters** new payment details.
4a.4. If new payment details are valid, the **use case** returns to step 4 in the **Main Success Scenario**. If the user cancels, the order is not placed, and the **use case** ends in failure.

(6a) **Email Service Unavailable:**
6a.1. The **System** fails to send the confirmation email.
6a.2. The **System** logs the error and **queues** the email for later sending.
6a.3. The **use case** otherwise completes normally (the order is still placed and the Customer sees the on-screen confirmation).

What is a use case?

- A group of related scenarios that provide value to the user of a system
  - A scenario is one of many paths within a use case
    - Main Success Scenario
    - Alternate Scenarios

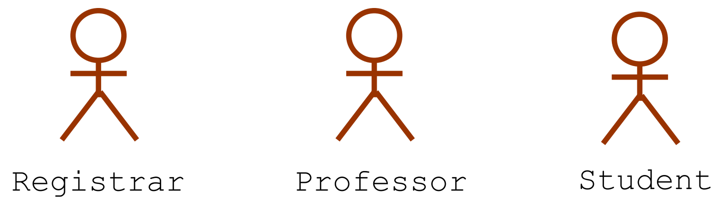
- Mechanism for projects to organize requirements in a way that makes them easy to find, understand, change, critique, test, verify and deliver incrementally.

Use case modeling elements

- Define Actors
- Define top level use case diagram
- Define use case diagrams
- Define use cases
- Example
  - A University wants to create an online registration system. The University registrar sets up the curriculum, students register for courses and the professors request course rosters.

Actors

- External entities that interact with a system
- They define a role played by entities when they interact with a system
  - many entities may play the same role or an entity may play many roles
- Example



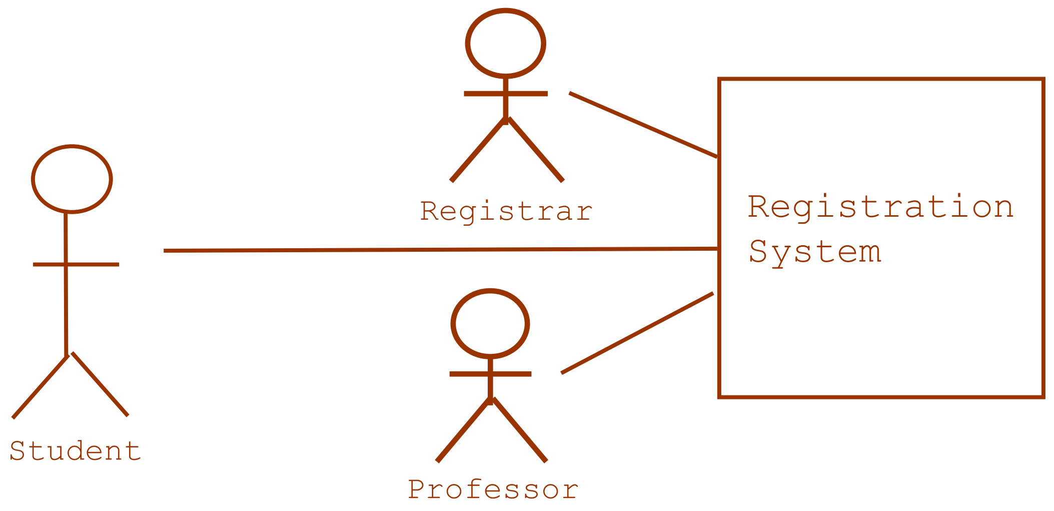
Top Level Use Case Diagram

Shows interface between actors and the system

Determines the scope of the system by clearly defining its boundary

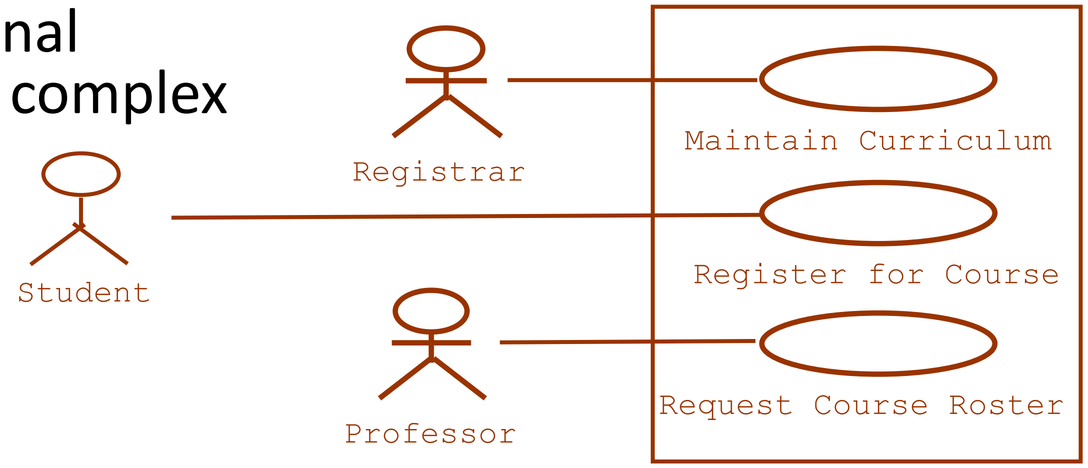
Also referred to as the system context diagram

Example



Use Case Diagrams

- A Use Case Diagram shows a set of external actors and the use cases each actor participates in
- Use the top level use case diagram to identify use cases
  - For each primary actor,
    - identify their goals (Elementary Business Processes)
    - Identify how they use the system (Scenarios)
- Consistently size use cases and scenarios
  - Too many: increased modularity, reduced functional complexity, complex configuration management, complex coordination, complex schedule
- Example



How to document Use Cases?

- Use Cases can be :
- Fully Dressed : Highly structured, covering all sections (e.g., scope, level, pre-conditions, main success scenario, extensions, special requirements, etc.).
- Partially Dressed: A simpler structure—often bullet points summarizing the main success path followed by alternative flows.

Fully Dressed Use Case Template (Larman’s Template)

- Use Case Name
- Scope
- Level
- Primary Actor
- Stakeholders and Interests
- Pre-Conditions
- Success Guarantee (success end condition)
- Main Success Scenario
- Extensions
- Special Requirements
- Technology & Data Variation List
- Frequency of Occurrence
- Open Issues/miscellaneous

Link to the fully dressed template

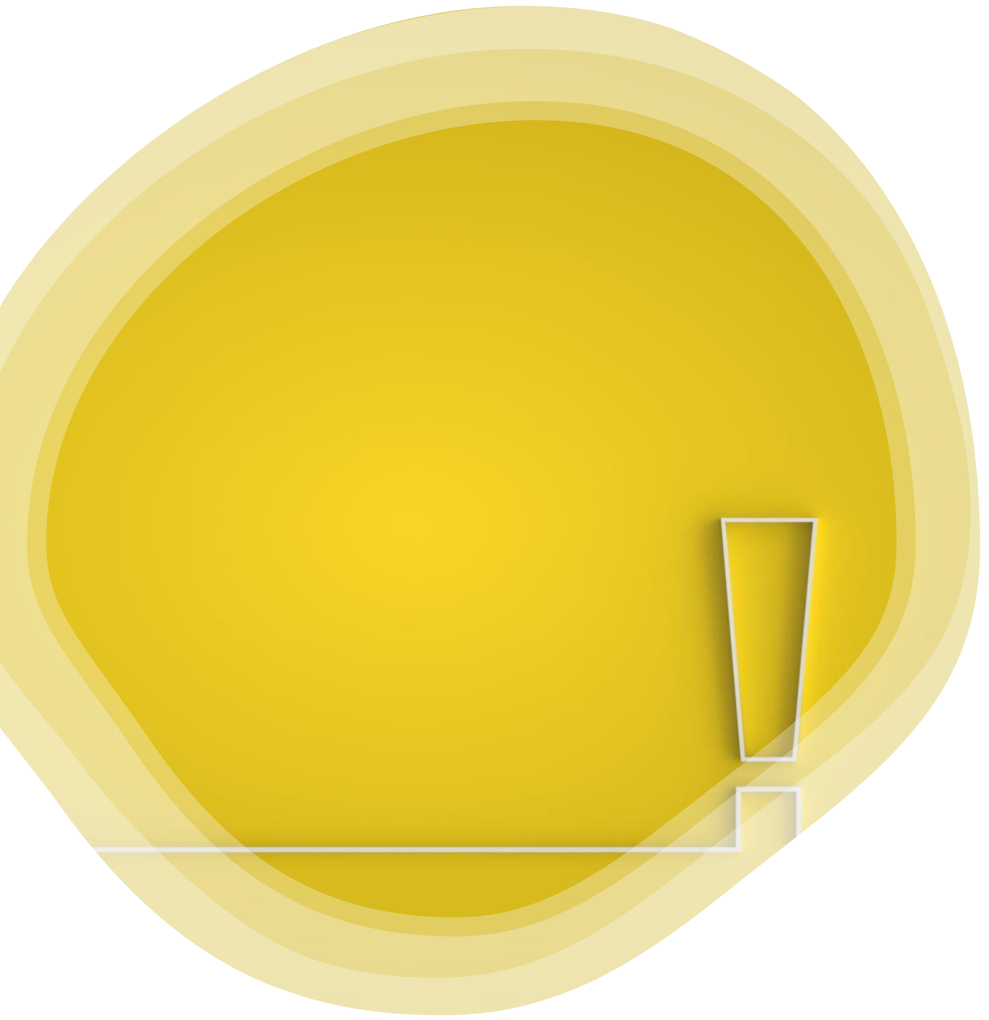
Main Success Scenario

- The main success scenario
  - Focus on what not on the how
  - Do not repeat pre-conditions
- Example

  1. Registrar enters the semester
  2. System presents course catalog
  3. Registrar selects a course
  4. System prompts for course offerings
  5. Registrar assigns sections and times

  Registrar repeats steps 3-5 until done

  6. System saves the semester curriculum



Extensions

- Alternate scenario paths such as
  - Exception conditions (failure conditions)
  - Alternate scenarios of success
- Example

  *a. At any time system fails

  1. Registrar starts again from the point of failure

  2a. Incorrect semester

  1. System signals error and rejects entry

For your Assignment 1, Deliverable 2:

- For your **Assignment 1**:
  - You will write requirements in “Shall Statements” format in section 5
  - For Section 6, You will then select five major requirements and rewrite them in use cases format. we recommend at least the following sections to be included for every use case:

  1. Use Case Title
  2. Main Success Scenario
  3. Extensions (Alternative or Failure Scenarios)

- For your **Project Deliverable 2**, you will write all your requirements in “Shall Statements” format.

READINGS , ASSIGNMENTS, PROJECT TASKS

Read:

- Chapters 7, 8 of the textbook.
- Read IEEE 29148 Template in the supplemental reading. (This is your template for project delivery 2)

Assignments & Project Tasks:

- Project Delivery 1 is due Friday.
- Assignment 1 is open (Due on Friday 7/1). This is an Individual Submission.

Questions?