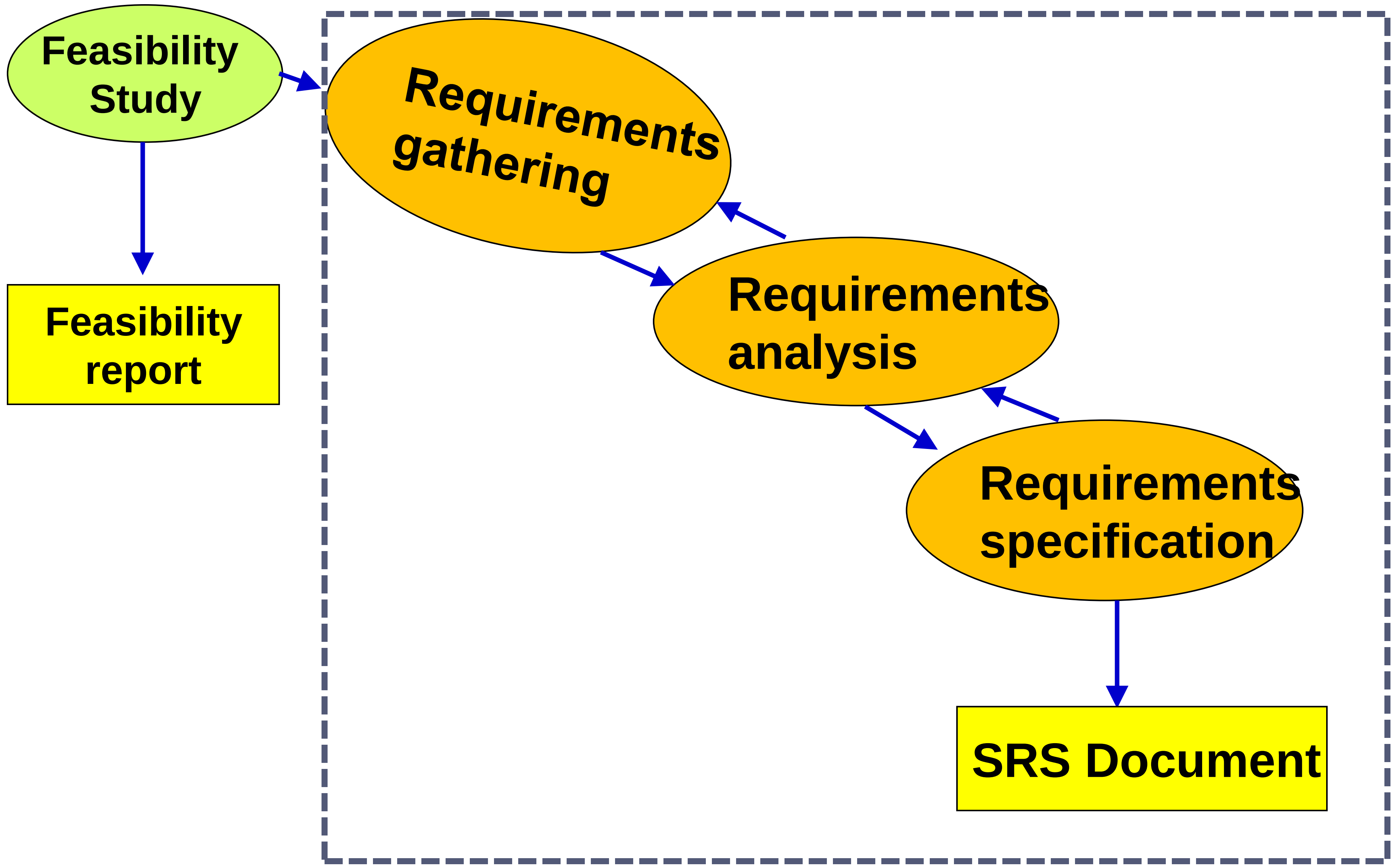
Advanced Software Engineering (CS6401)

Autumn Semester (2024-2025)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Requirement Analysis and Specification

Requirements Engineering Process



Requirements Analysis and Specification

- Requirements Gathering:
  - Fully understand the user requirements.
- Requirements Analysis:
  - Remove inconsistencies, anomalies, etc. from requirements.
- Requirements Specification:
  - Document requirements properly in an SRS document.

Requirements Analysis and Specification

- **Aim of this phase:**
  - understand the exact requirements of the customer,
  - document them properly.

- Consists of two distinct activities:
  - requirements gathering and analysis
  - requirements specification.

Goals of Requirements Analysis

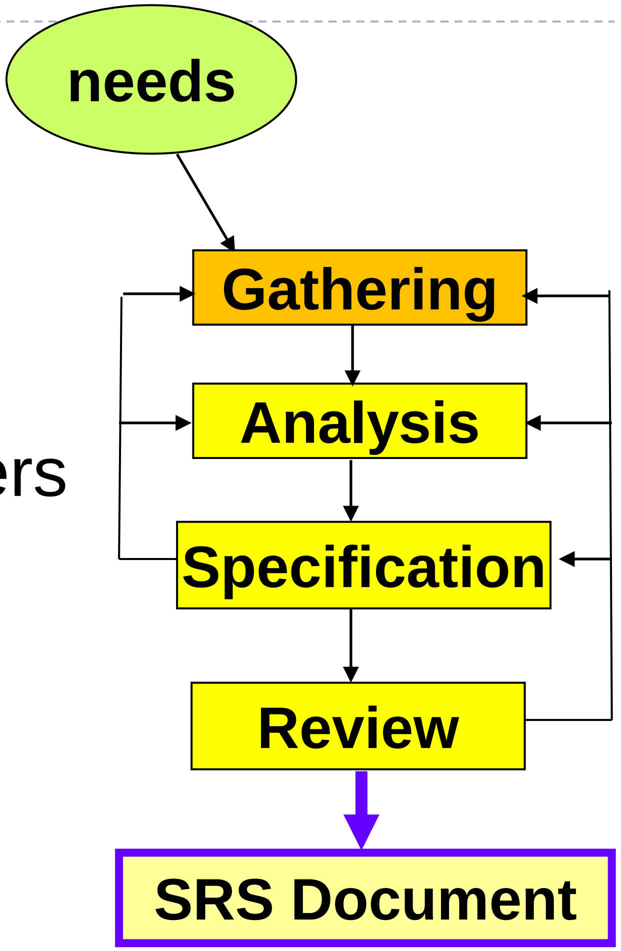
- Collect all related data from the customer:
  - Analyse the collected data to clearly understand what the customer wants,
  - Find out any inconsistencies and incompleteness in the requirements,
  - Resolve all inconsistencies (requirements contradicts) and incompleteness (parts of requirement omitted).

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through interviews and discussions.
  - For example, for a business accounting software:
    - interview all the accountants of the organization to find out their requirements.

How to Gather Requirements?

- Observe existing (manual) systems
- Study existing procedures
- Discuss with customer and end-users
- Input and Output analysis
- Analyse what needs to be done



Requirements Gathering Activities

- 1. Study existing documentation
- 2. Interview
- 3. Task analysis
- 4. Scenario analysis
- 5. Form analysis

Requirements Gathering (CONT.)

- In the absence of a working system,
  - Lot of imagination and creativity are required.
- Interacting with the customer to gather relevant data:
  - Requires a lot of experience.

Requirements Gathering (CONT.)

- Some desirable attributes of a good requirements analyst:
  - Good interaction skills,
  - Imagination and creativity,
  - Experience…

Case Study: Automation of Office Work at CSE Dept.

- The academic, inventory, and financial information at the CSE department:
  - At present carried though manual processing by two office clerks, a store keeper, and two attendants.
- Considering the low budget he had at his disposal:
  - The HoD entrusted the work to a team of student volunteers.

Case Study: Automation of Office Work at CSE Dept.

Interview

- The team was first briefed by the HoD:
  - Concerning the specific activities to be automated.
- The analysts first discussed with the two office clerks:
  - Regarding their specific responsibilities (tasks) that were to be automated.
- The analyst also interviewed student and faculty representatives who would also use the software.

Case Study: Automation of Office Work at CSE Dept.

Task and Scenario Analysis

- For each task that a user needs the software to perform, they asked:
  - The steps through which these are to be performed.
  - The various scenarios that might arise for each task.

Form Analysis

- Also collected the different types of forms that were being used.

Case Study: Automation of Office Work at CSE Dept.

Requirements Analysis

- The analysts understood the requirements for the system from various user groups:
  - Identified inconsistencies, ambiguities, incompleteness.
- Resolved the requirements problems through discussions with users:
  - Resolved a few issues which the users were unable to resolve through discussion with the HoD.

Case Study: Automation of Office Work at CSE Dept.

Requirements Specification

- Documented the requirements in the form of an SRS document.

Properties of a Good SRS Document

- **It should be concise**
  - at the same time should not be ambiguous.
  - Verbose and irrelevant description reduces readability and increases the possibility of errors.
- **It should be implementation-independent.**
  - It should specify what the system must do and not say how to do it.
  - Specify externally visible behaviour of the system and not discuss the implementation issue.

Properties of a Good SRS Document

- **It should be modifiable.**
  - **Easy to change,**
    - i.e., it should be well-structured.
    - Well-structured document is easy to understand and modify.
- **It should be consistent.**
- **It should be complete.**

Properties of a Good SRS Document (cont...)

- **It should be traceable**
    - You should be able to trace which part of the specification corresponds to which part of the design, code, etc and vice versa.
- **It should be verifiable**
    - e.g. “system should be user friendly” is not verifiable.
    - e.g. “A requirement that is checking of availability of books in the library” is verifiable.

Attributes of a Bad SRS Document (cont...)

- **Over-specification**
  - It occurs when analyst tries to address “how to” aspects in the specification.
  - Example, library membership record need to be stored indexed on the member’s first name or library membership identification number.
- **Forward referencing**
  - One should not refer to aspects that are discussed much later in the SRS document.
  - Reduces readability.

Properties of a Bad SRS Document (cont...)

- **Wishful thinking**
  - Description of aspects which would be difficult to implement.
- **Noise**
  - e.g., register customer function , suppose analyst writes that customer registration department is manned by clerks who report for work between 8 am and 5 pm, 7 days a week.

SRS should not include...

- **Project development plans**
  - E.g. cost, staffing, schedules, methods, tools, etc
    - Lifetime of SRS is until the software is made obsolete
    - Lifetime of development plans is much shorter
- **Product assurance plans**
  - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc
    - Different audiences
    - Different lifetimes

SRS should not include (cont...)

- **Designs**
  - Requirements and designs have different audiences
  - Analysis and design are different areas of expertise

Alternatives to Natural Language(NL) specification

- Structured Natural Language (modules)
  - Usage of forms or templates
- Design Description Languages with graphical notations
  - UML Use cases, message sequence charts etc.
- Mathematical specifications
  - ADTs

Structured language specifications

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Indication of other entities required.
- Pre and Post conditions (if appropriate).
- The side effects (if any) of the function.

what input to the module and output going as a input to other module and any sort of pre conditions on the inputs need to be validated by the module. For example, having a name then the pre condition could be not having any digital characters.

Form-based specification Example

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose – the dose in insulin to be delivered
Destination	Main control loop
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

Tabular specification

Condition	**Action**
Sugar level falling ($r_2 < r_1$)	CompDose = 0
Sugar level stable ($r_2 = r_1$)	CompDose = 0
Sugar level increasing and rate of Increase decreasing ($(r_2-r_1) < (r_1-r_0)$)	CompDose = 0
Sugar level increasing and rate of Increase stable or increasing ($(r_2-r_1) >= (r_1-r_0)$)	CompDose = round($(r_2-r_1)/4$) if rounded result = 0 then CompDose=MinimumDose

Graphical models (primarily UML derivatives)

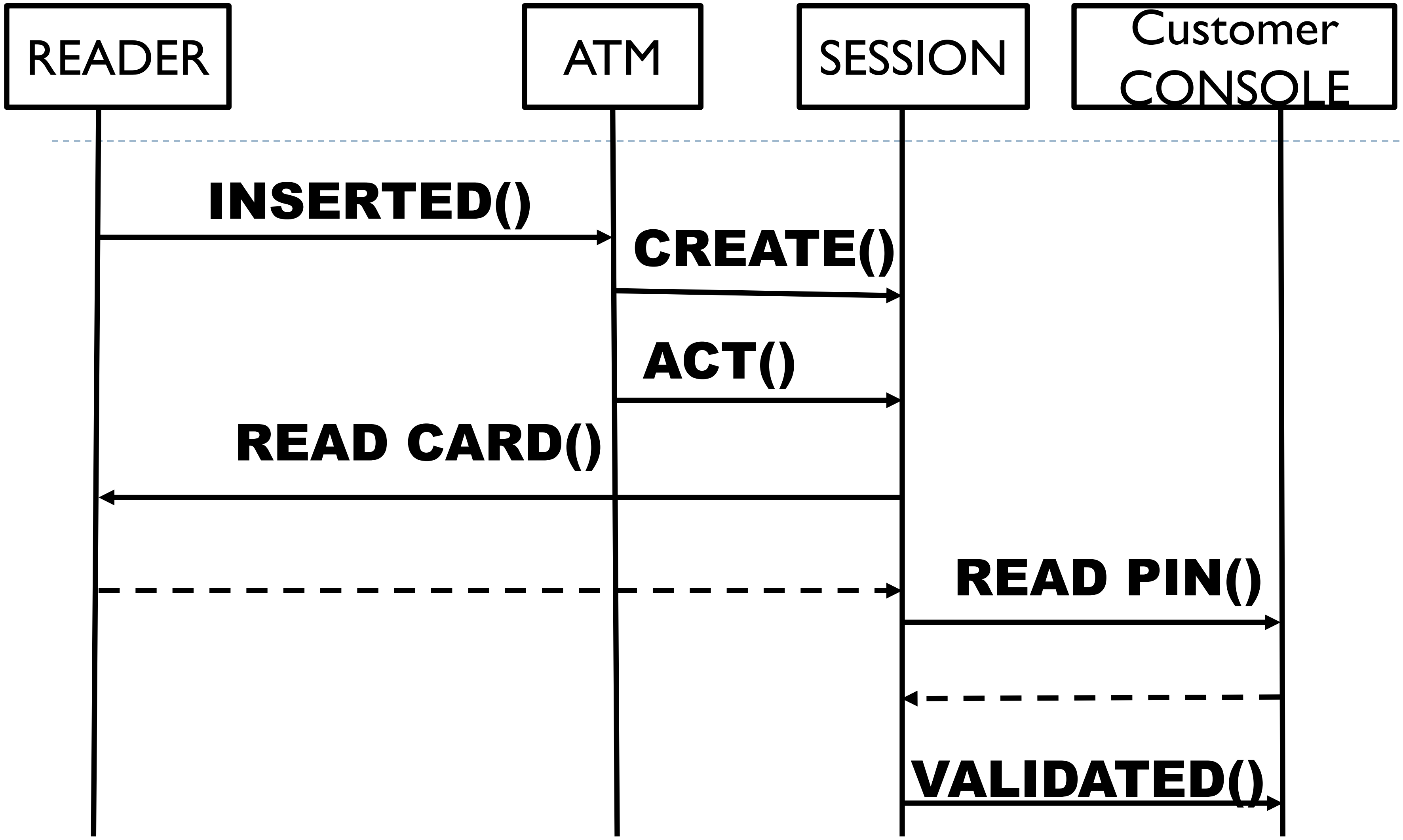
- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.

Sequence diagrams

- These show the sequence of events that take place during some user interaction with a system.

- You read them from top to bottom to see the order of the actions that take place.

- Cash withdrawal from an ATM
  - Validate card;
  - Handle request;
  - Complete transaction.



- This is how it used to formally specify the user requirement

Interface specification

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.

- Three types of interface may have to be defined
  - Procedural interfaces;
  - Data structures that are exchanged;
  - Data representations.

- Formal notations are an effective technique for interface specification.

PDL interface description

```
Interface PrintServer{
        //Defines an abstract Printer Server
    void initialize(Printer p);
    void print(Printer p, Document d);
    void displayPrintQueue(Printer p);
    void cancelPrintJob(Printer p, Document d);
 } //Print Server
```

The requirement document

- The requirement document is the official statement of what is required of the system developers.

- Should include both a definition of user requirements and a specification of the system 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.