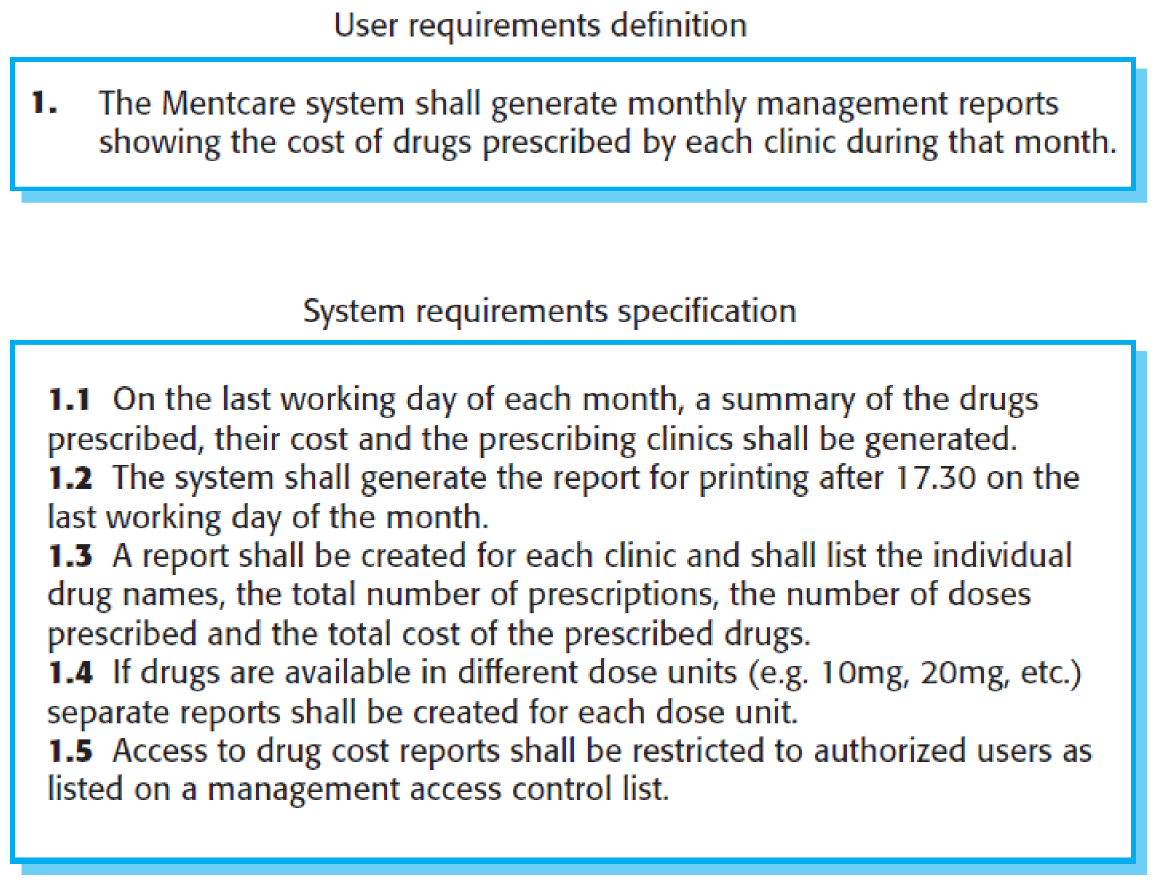
4. Requirements engineering and documentation

What is requirements engineering?

- A customer says what they want…
  - But they do not necessarily know the right terms, or the right technology
  - We need to transform that into requirements documentation through engineering
- User requirements
  - Uses <u>natural language</u>, focuses on system service/value to users
- System requirements (functional specification)
  - Detailed, technical explanations, exact feature descriptions
  - Can be part of the contract itself

What is requirements engineering?



User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Sommerville

Two types of system requirements

- *Functional* requirements: What the system does
  - **Features**: Services provided, use cases, information
  - How it responds to inputs, what are its outputs
  - Can also be "what the system must not do"
- *Non-functional* requirements: What the system needs to be
  - **Constraints**: time taken, delivery, technical, security
  - Can be technical: reliability, memory use, storage, etc.
  - Can also be organizational or external requirements
  - What are some plausible non-functional requirements for the previous system?

Requirements Engineering Process

- Three steps:

1. Requirements Elicitation: Understanding what the stakeholders want
   - Talking to them, involving them in design, understanding their environment
2. Requirements Specification: Understanding what we are doing
   - Writing user requirements, system requirements, etc.
3. Requirements Validation: Understanding what we can do
   - Check our requirements: validity, consistency, completeness, realism
   - Testability is important

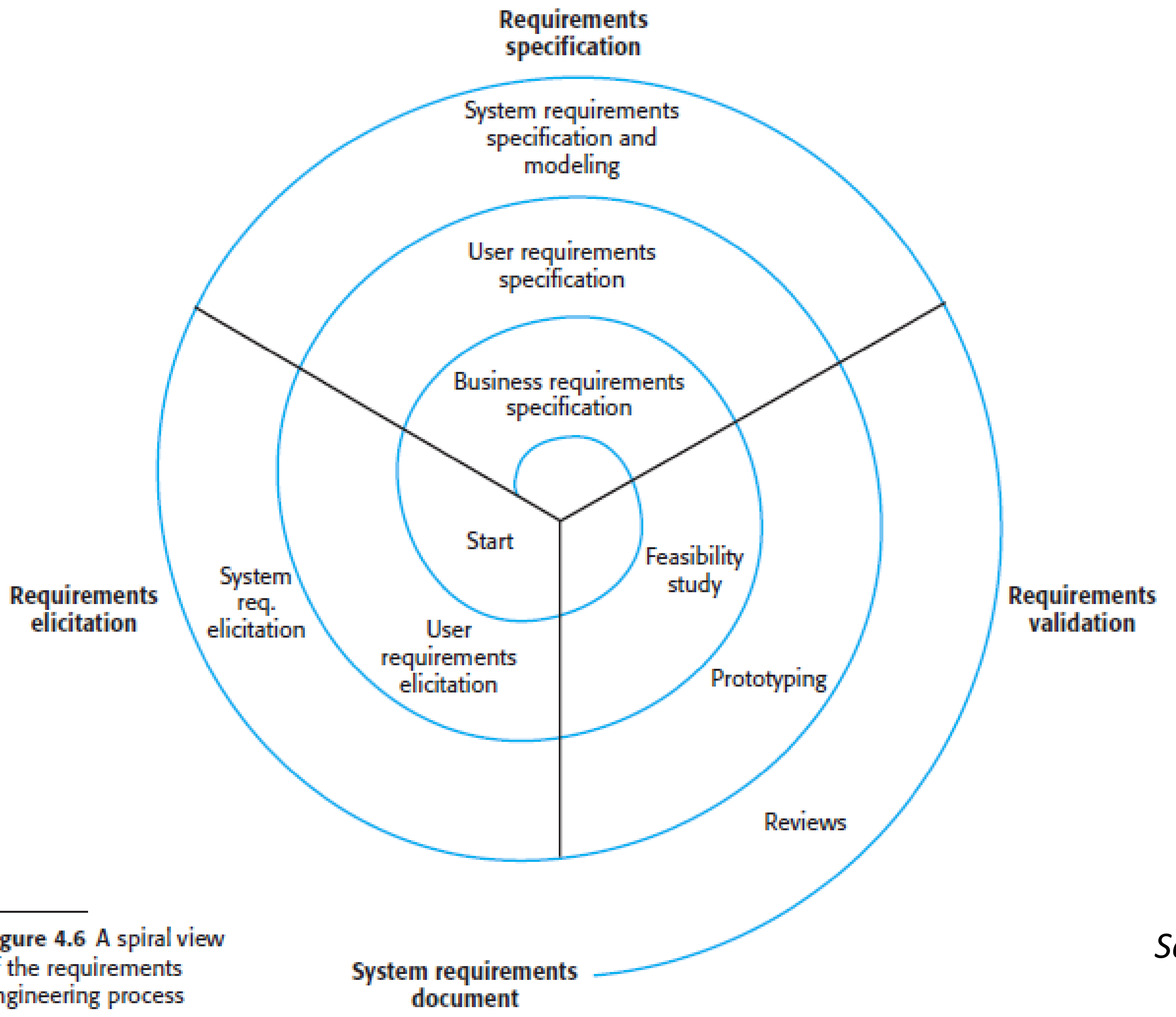

Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Requirements elicitation
System req. elicitation
User requirements elicitation
Prototyping
Requirements validation
Reviews
System requirements document

Figure 4.6 A spiral view of the requirements engineering process

Sommerville

Requirements Elicitation

- Challenges:
    - Communication gap between stakeholder and developers
        - Different knowledge sets, implicit assumptions
        - Unspoken factors, unspeakable factors
    - Conflicting motivations
        - Negotiation – convincing the stakeholder they want something different?
        - Conflict between different groups, management, developers…
    - Constantly changing environment
- Core skill: Communication!
    - Underappreciated skill in software engineering

Types of Requirements Elicitation

- Interviewing
  - Closed/open interviews
  - Use springboard questions
- Observation
  - Ethnography: systematic observation of work environment
  - Gives better understanding of how stakeholders work
  - Find out more about implicit/unspoken factors
  - Field research

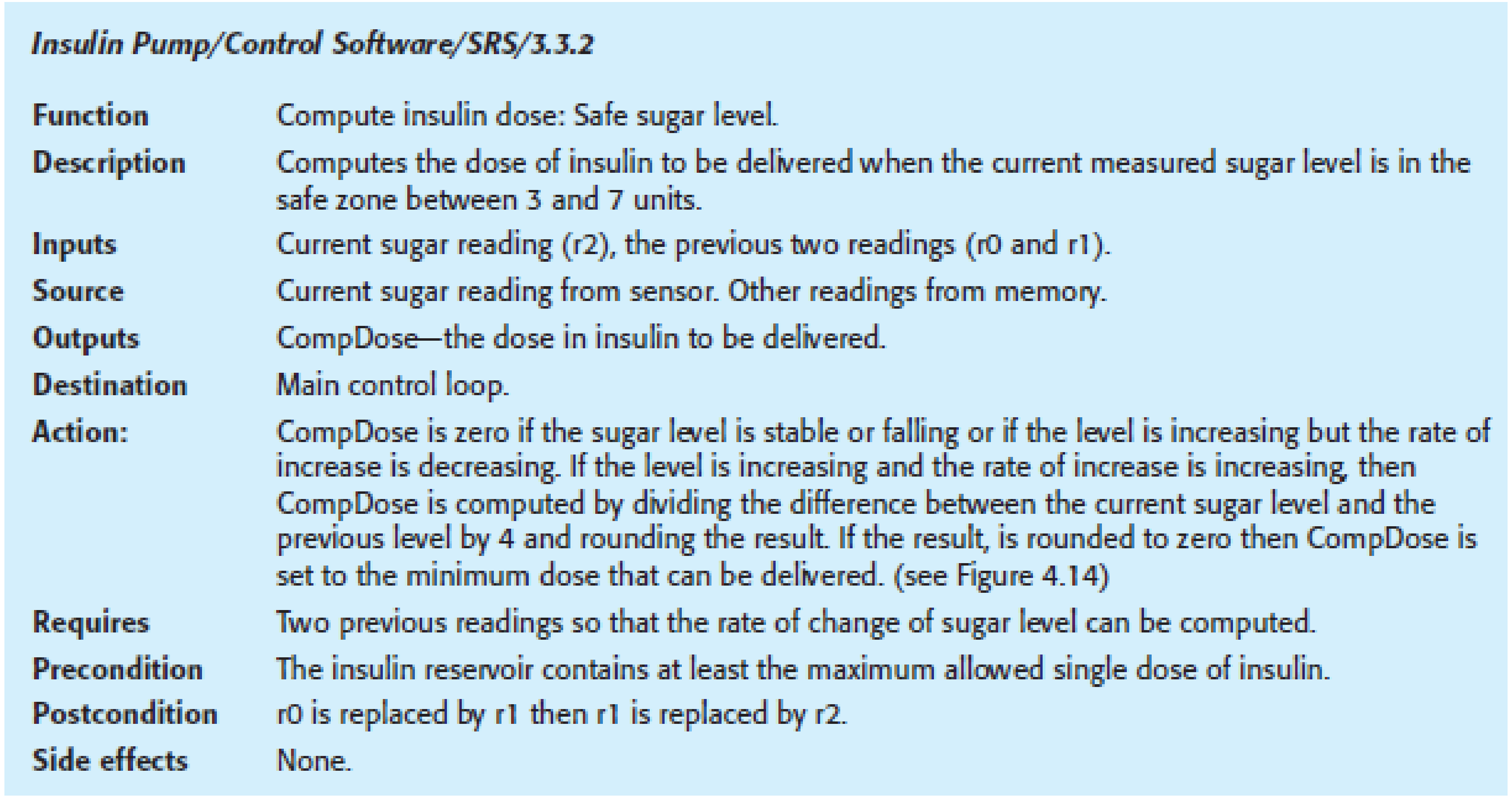
User Stories/Scenarios

- One way for stakeholders to express requirements
- Write out a story to explain what the system should do
- A story should:
  - Set up the environment, describe the characters, explain the need
  - Give certain adversarial conditions and how they are solved
  - Show how characters interact
  - Give events in order to express logic of system

Requirements document

- Introduction, glossary
- User requirements
- System requirements
  - Structured requirements
  - Tabular specification of requirements
- System architecture and models (next!)
- System evolution

Structured requirements

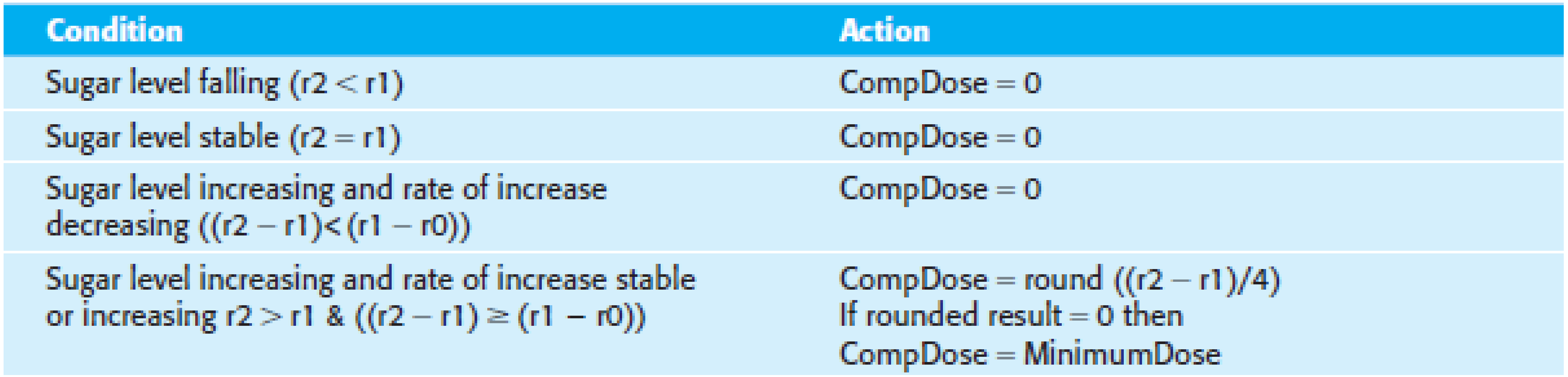


Insulin Pump/Control Software/SRS/3.3.2

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. (see Figure 4.14)
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Precondition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Postcondition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Sommerville

Tabular requirements



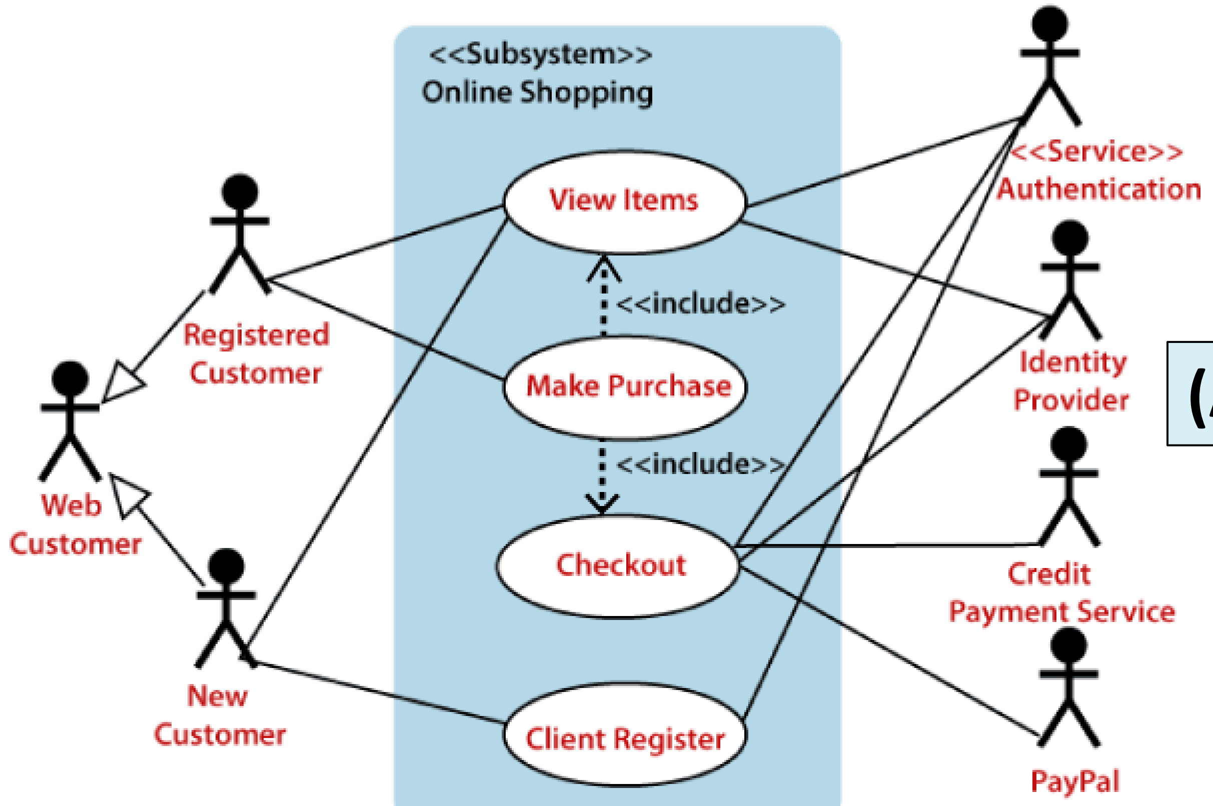
Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing ($(r2 - r1) < (r1 - r0)$)	CompDose = 0

Sugar level increasing and rate of increase stable or increasing $r2 > r1$ & $((r2 - r1) \geq (r1 - r0))$	CompDose = round ($(r2 - r1)/4$)
	If rounded result = 0 then
	CompDose = MinimumDose

System modeling

- Unified Modeling Language
  - Generalized set of rules on how to present a model
    - What are the components of the system, how do they interact, how do entities interact with the system
  - Many different types of models are supported:
    - Behavioral diagrams
      - Use-case diagram, etc.
    - Structural diagrams
      - Class diagram, etc.

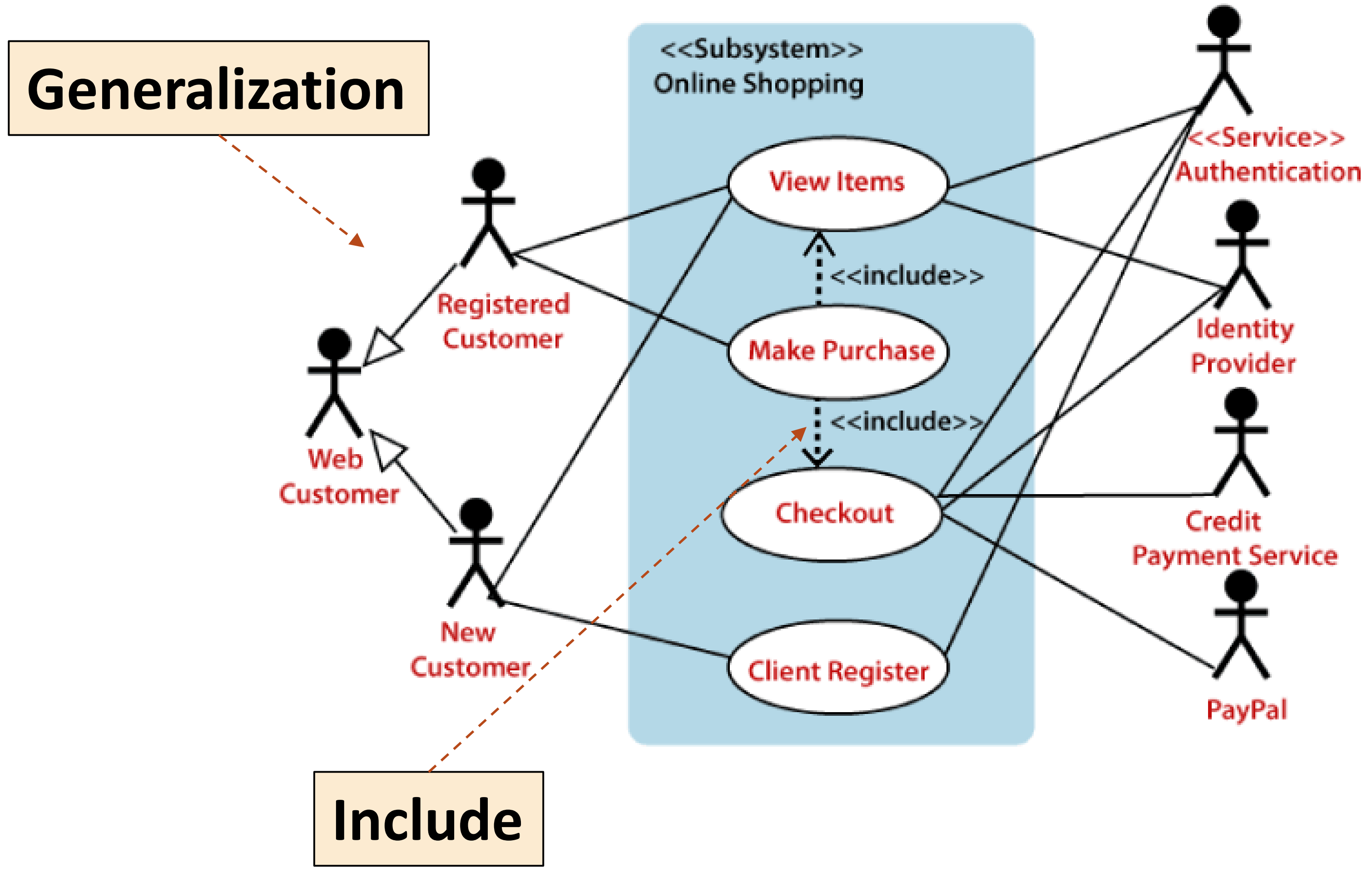
UML: Use-case diagram



(Actors)

(Use cases)

UML: Use-case diagram (relationships)



UML Class Diagram

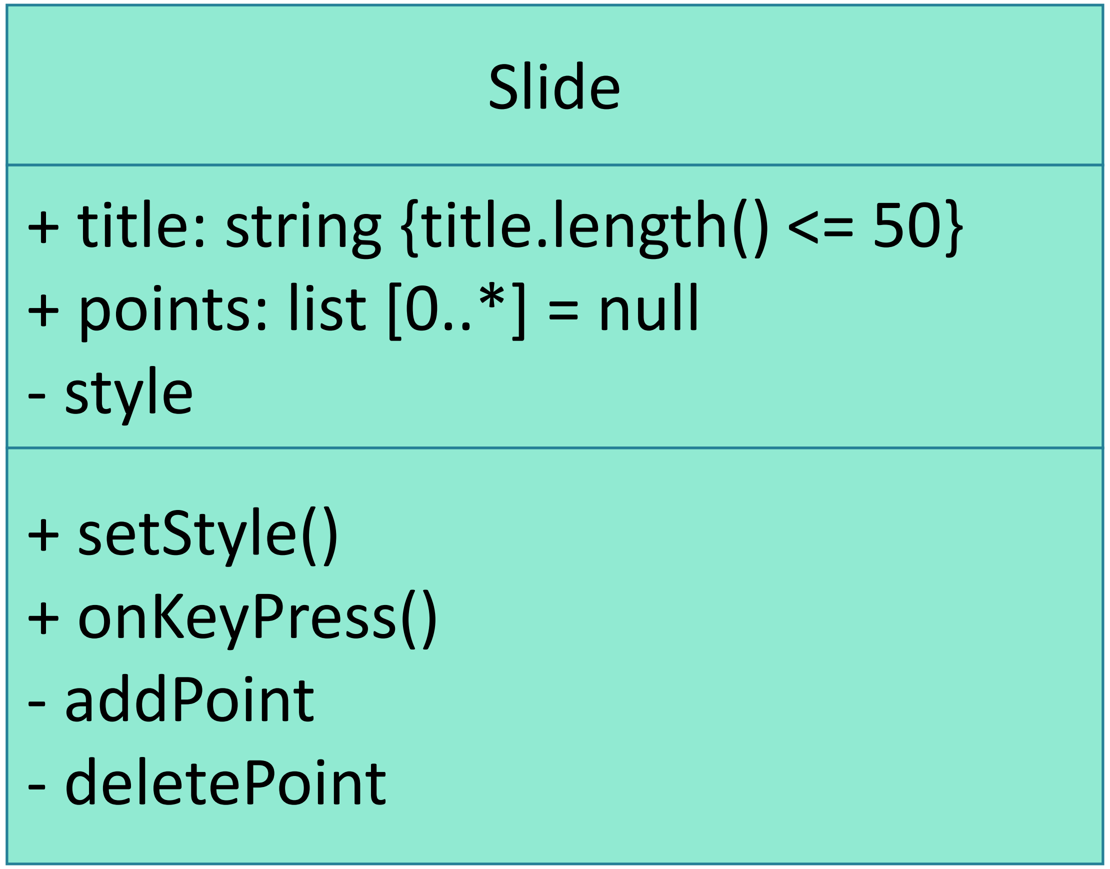
- Classes, their relationships, their methods and fields
- Can be used to automatically generate (startup) code
- Strict rules on presentation
  - Classes, visibility, constraints, attributes,
  - Six types of relationships
  - Navigability
  - Multiplicity

UML Class Diagram: Classes

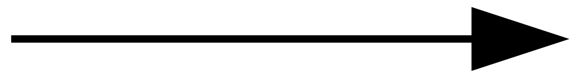
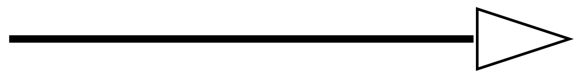
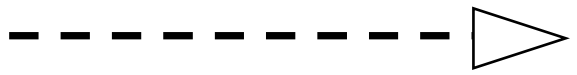
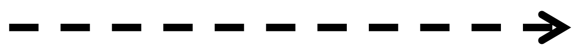
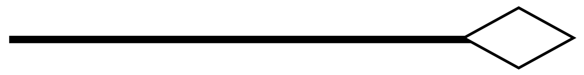
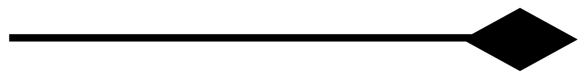
- Each box is one class (entity)
- Top part of box is fields
- Bottom part is methods
- Starts with:

+	Public
-	Private
#	Protected

- Can have type, constraints {}, attribute [], default value =, etc.

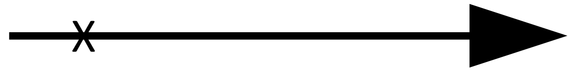

Slide
+ title: string {title.length() <= 50}
+ points: list [0..*] = null
- style
+ setStyle()
+ onKeyPress()
- addPoint
- deletePoint

UML Class Diagram: Relationships

Association		General relationship, e.g. A has a object reference to B
Generalization		Equivalent to “extends” parent
Realization		Equivalent to “implements” interface
Dependency		A is implemented/specified by B
Aggregation		Several of A make up B
Composition		Stronger aggregation

UML Class Diagram: Relationships

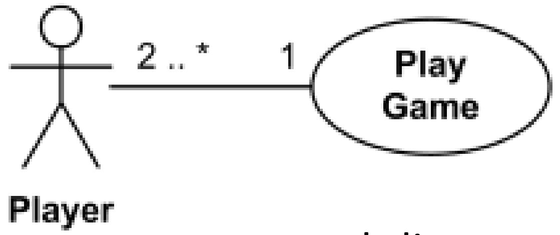
- Both sides of relationship can be specified
- Navigability: Whether or not A can easily reference B
  - Draw a cross at one end if it's not navigable



  - A link without arrow is an association with unclear navigability
- What's the difference between aggregation and composition?
  - If B composites A, then B must be a part of exactly one A
    - e.g. composition: wheels of a car, aggregation: students of a class
  - If A is deleted, then B is also deleted
    - e.g. composition: pixels of a circle, aggregation: treasure chests in a room

UML Class Diagram: Multiplicity

- Specify how many of each
- One end can be unspecified
- a..b means “at least a and at most b instances”
  - * means any number
  - a can be 0
  - a by itself means “exactly a instances”



uml-diagrams.org