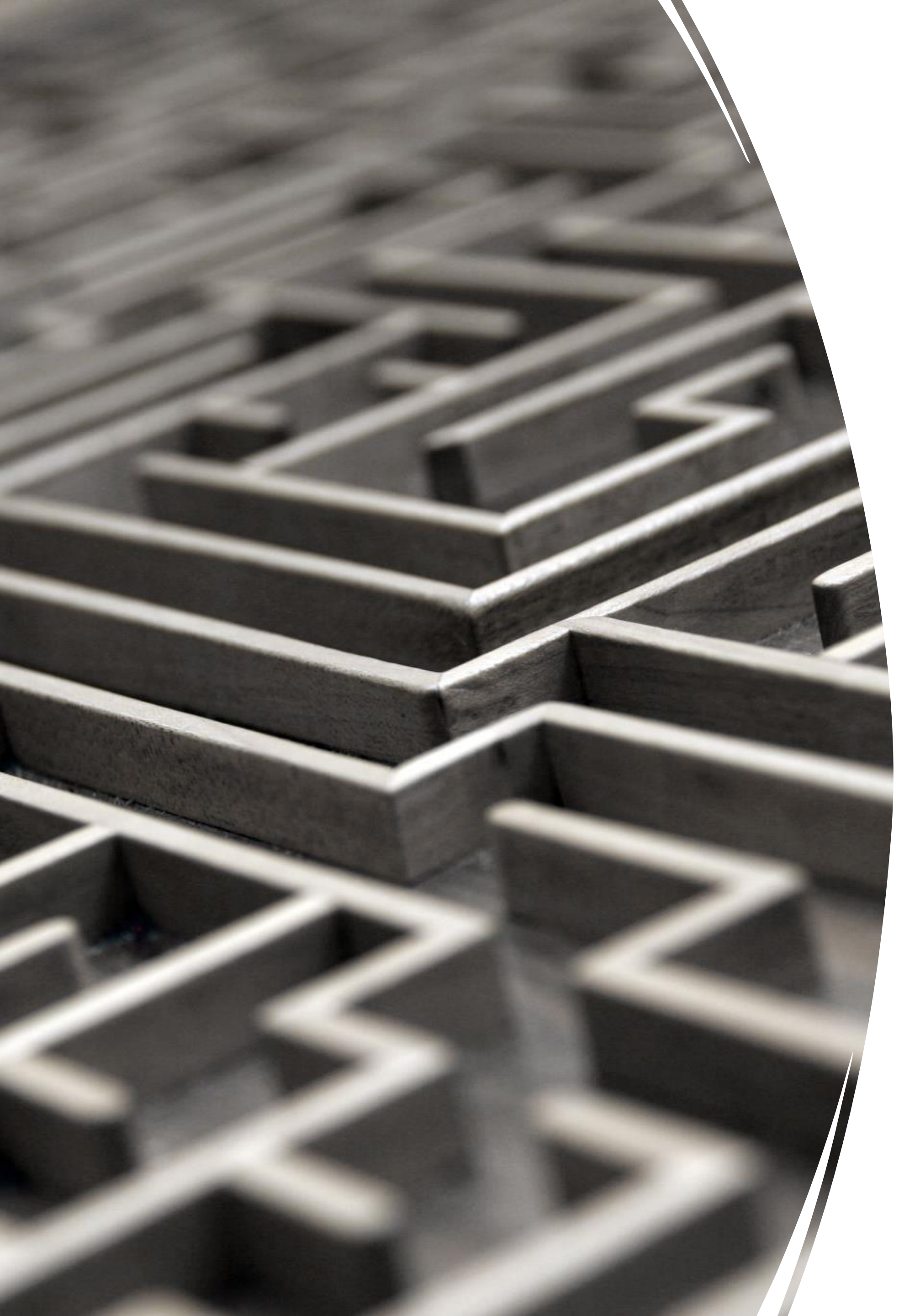
Software Engineering COMP1035

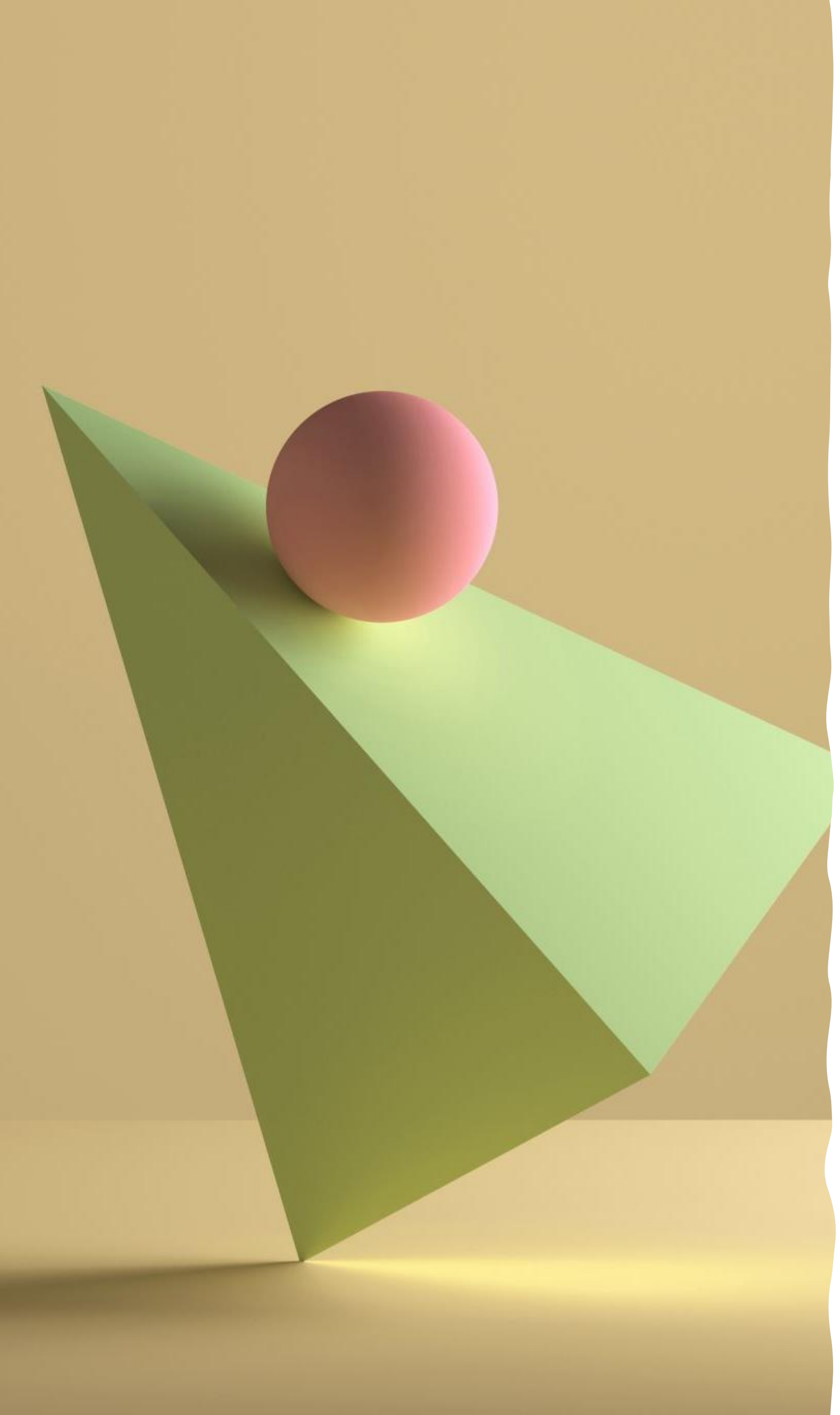
Lecture 07

Specifications



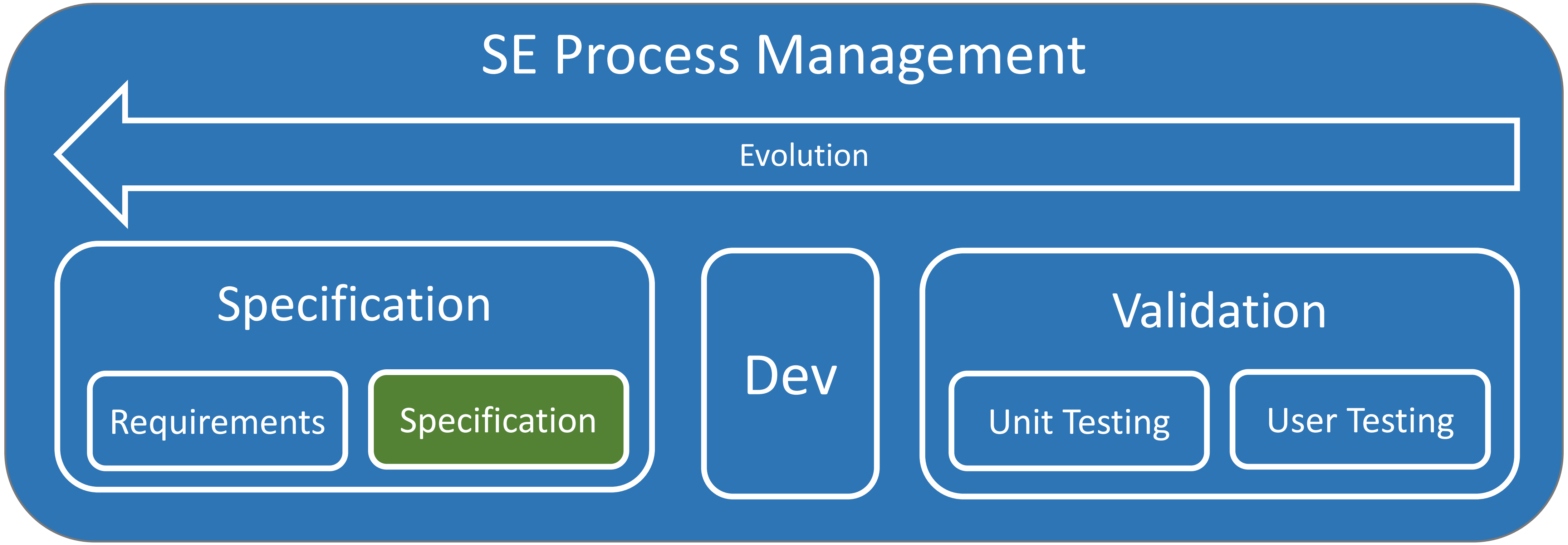
Today’s Objectives:

1. What specifications are (and what they look like)?
2. How are they different with requirements?
3. What makes a good/bad specification?
4. What are they used for?

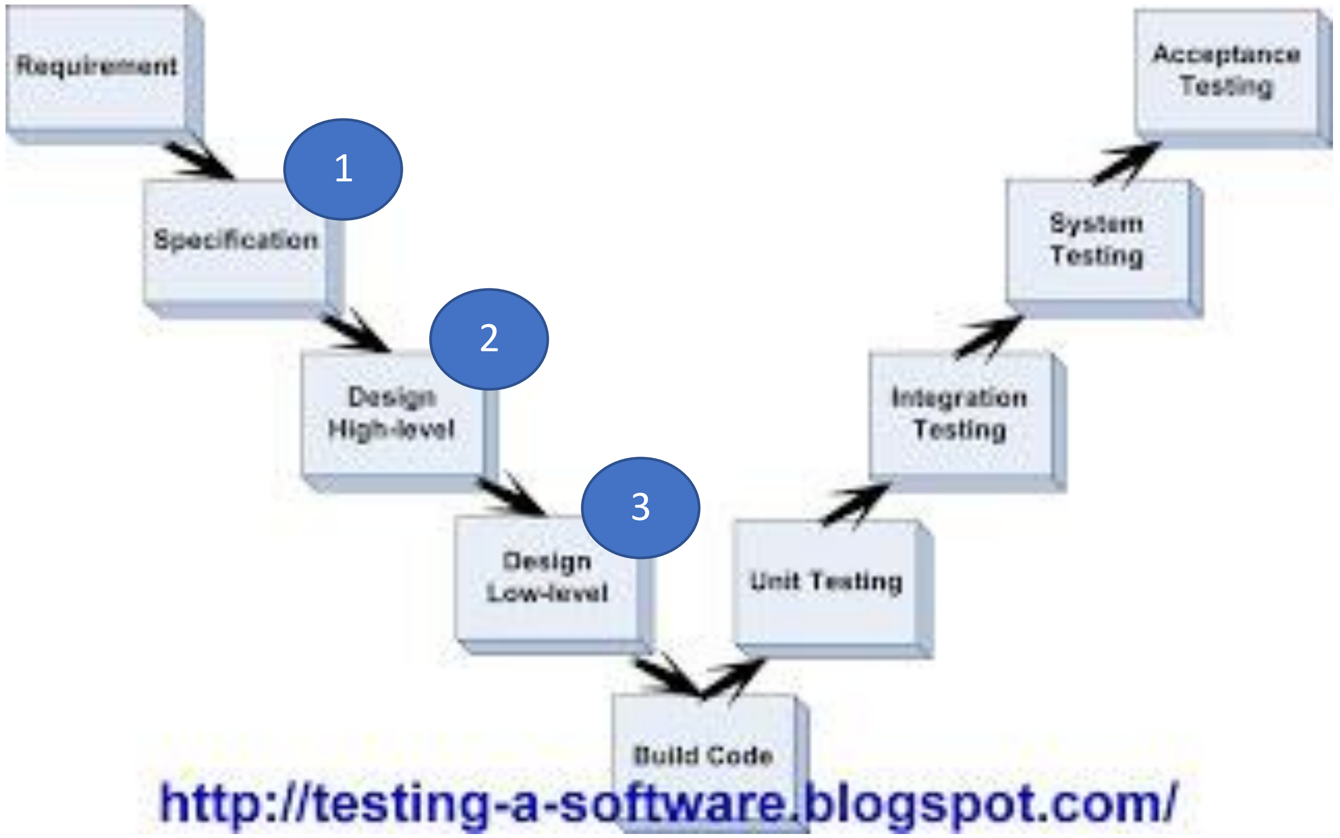


Where We Are In the Process

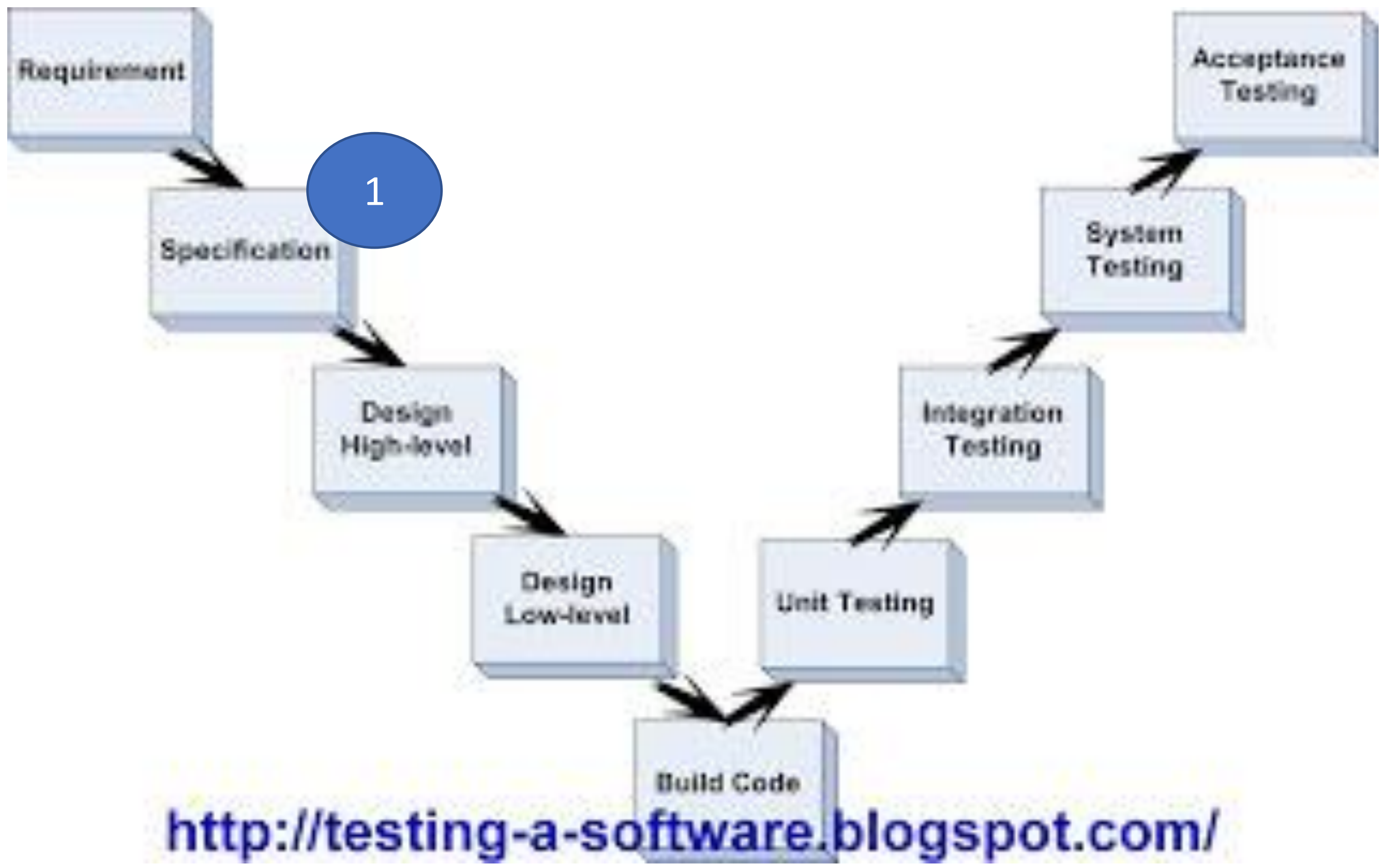
Keeping Track of SE Module

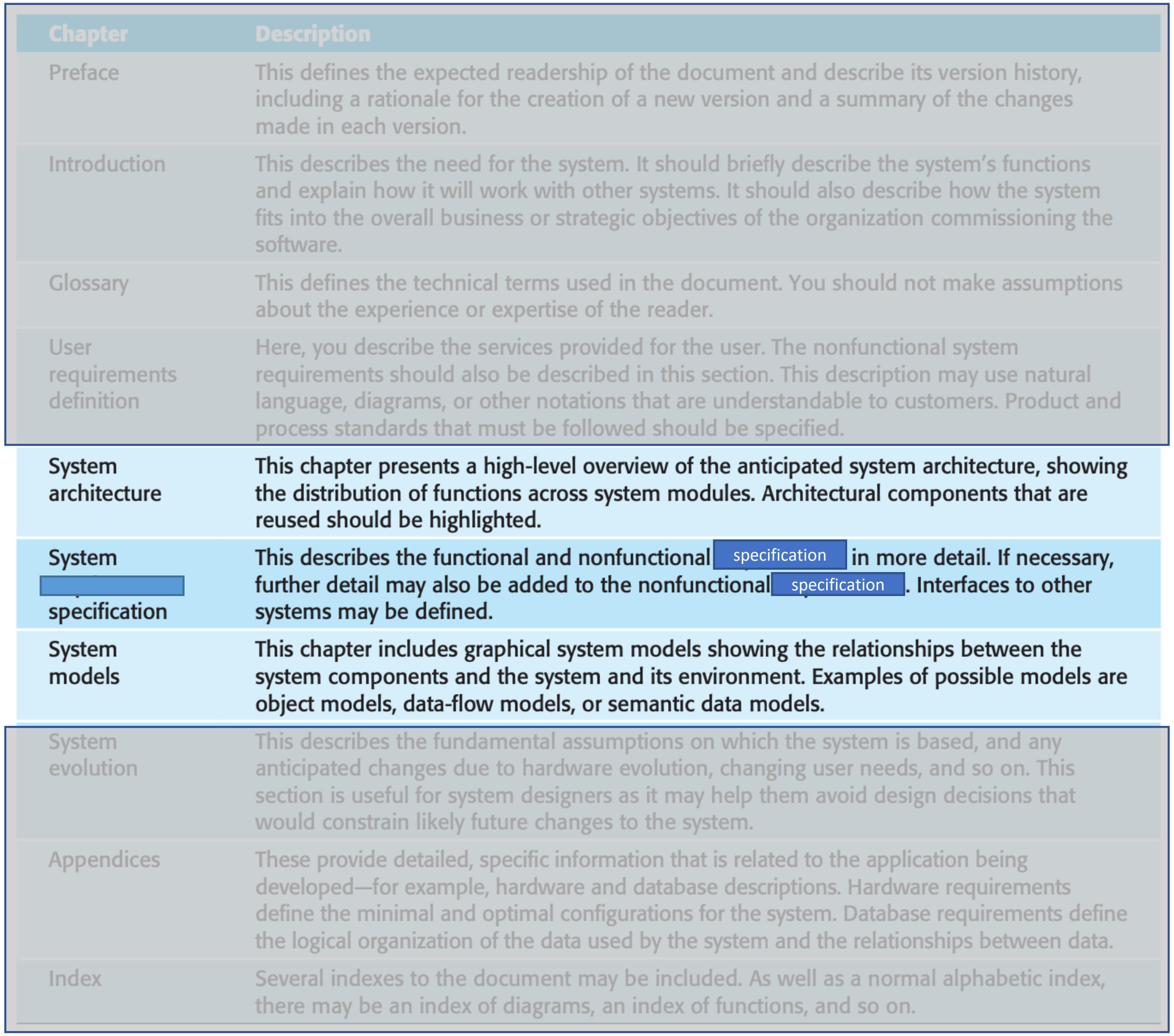


Keeping Track of SE Module



Keeping Track of SE Module





Chapter	Description
Preface	This defines the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This describes the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This defines the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System [redacted] specification	This describes the functional and nonfunctional specification in more detail. If necessary, further detail may also be added to the nonfunctional specification. Interfaces to other systems may be defined.
System models	This chapter includes graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These provide detailed, specific information that is related to the application being developed—for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Figure 4.17 The structure of a requirements document

Specifications vs Requirements

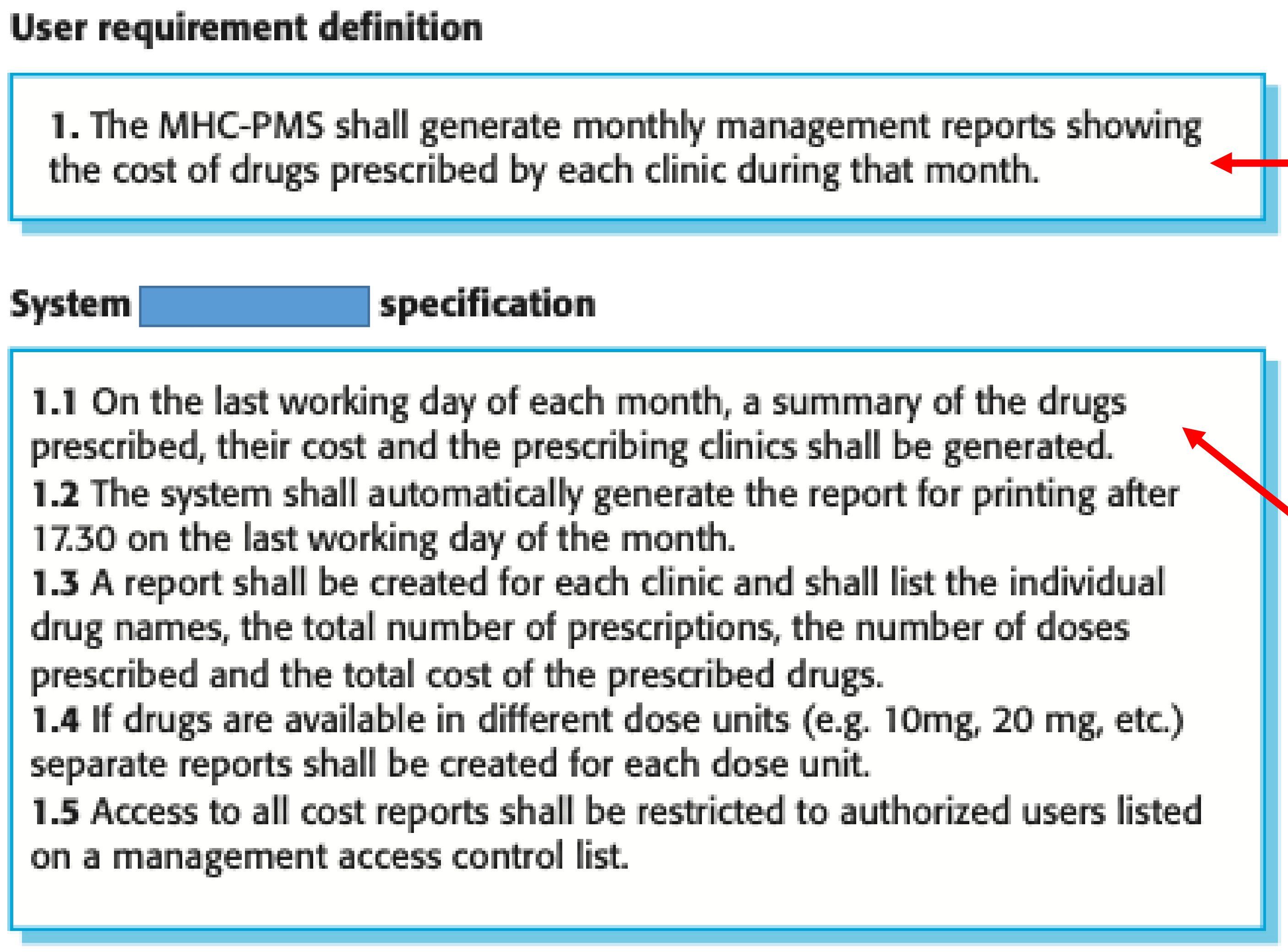
Requirements vs Specifications

	Functional	Non-Functional
(User) Req.	1. A module conve has the pre-requisite knowledge to ta	
(System) Spec.	1. A module convenor shall be able to see a list of students who want to take their module. 2. A module conve es of a student. 3. A module conve nts put by students. 4. A module conve nt. 5. A module convenor shall be able to contact/ask the student to clarify.	

What a stakeholder needs to be able to do.

What the software must do to meet the requirement above.

Requirements vs Specifications



User requirement definition

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

System [redacted] **specification**

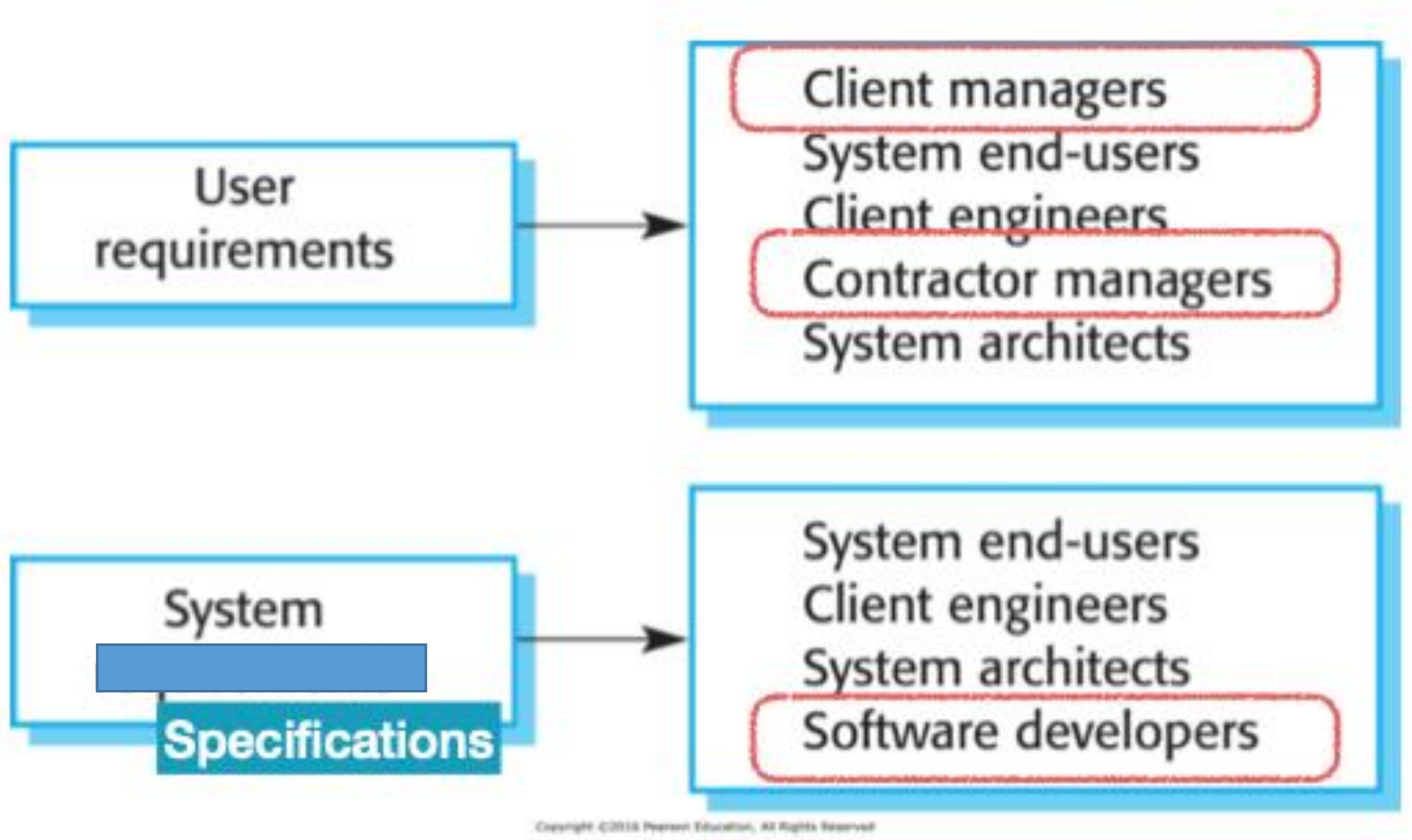
1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic 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, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Something the stakeholder will need – or a service the software must provide.

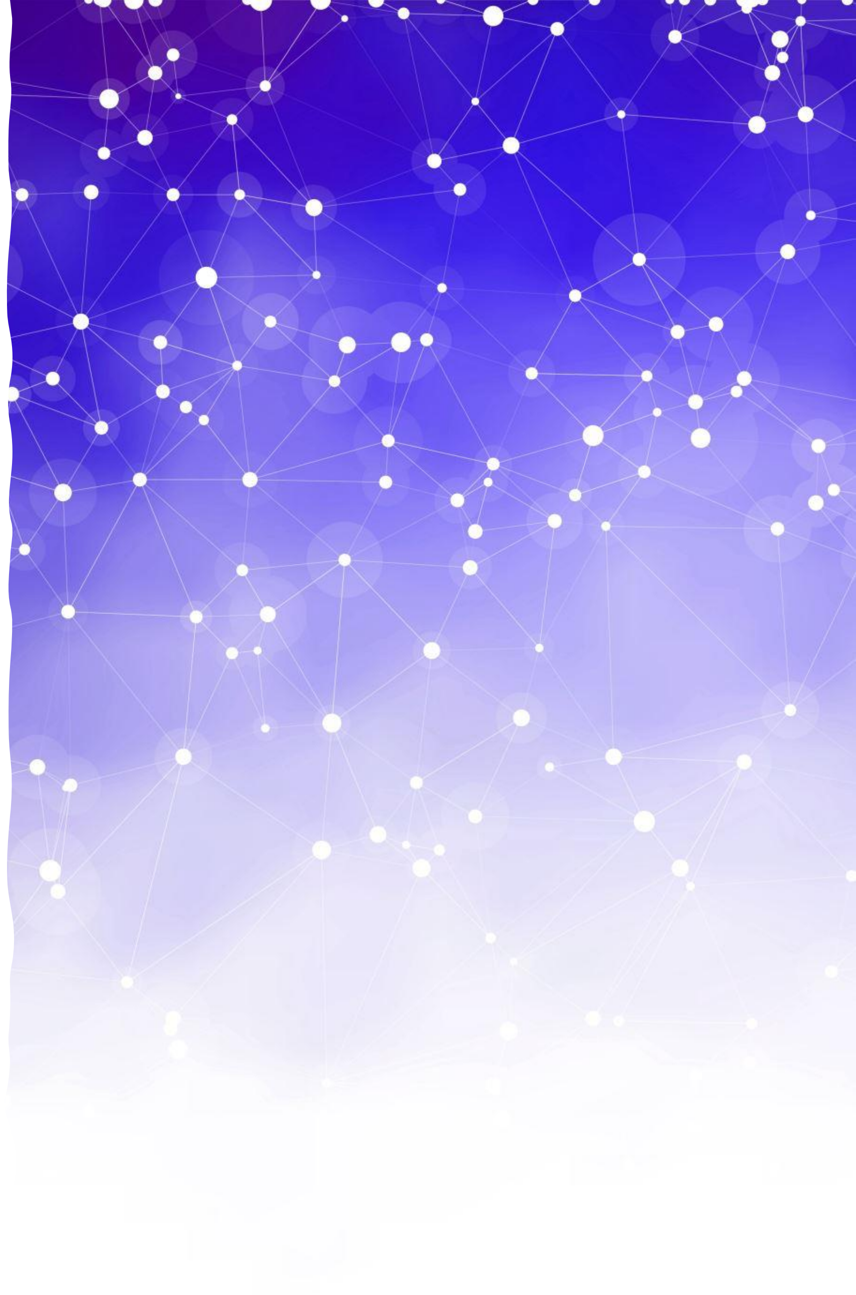
What the system must do to meet this user requirements.

Who Looks at Specifications?

Figure 4.2 Readers of different types of requirements specification



What Specifications Are Like

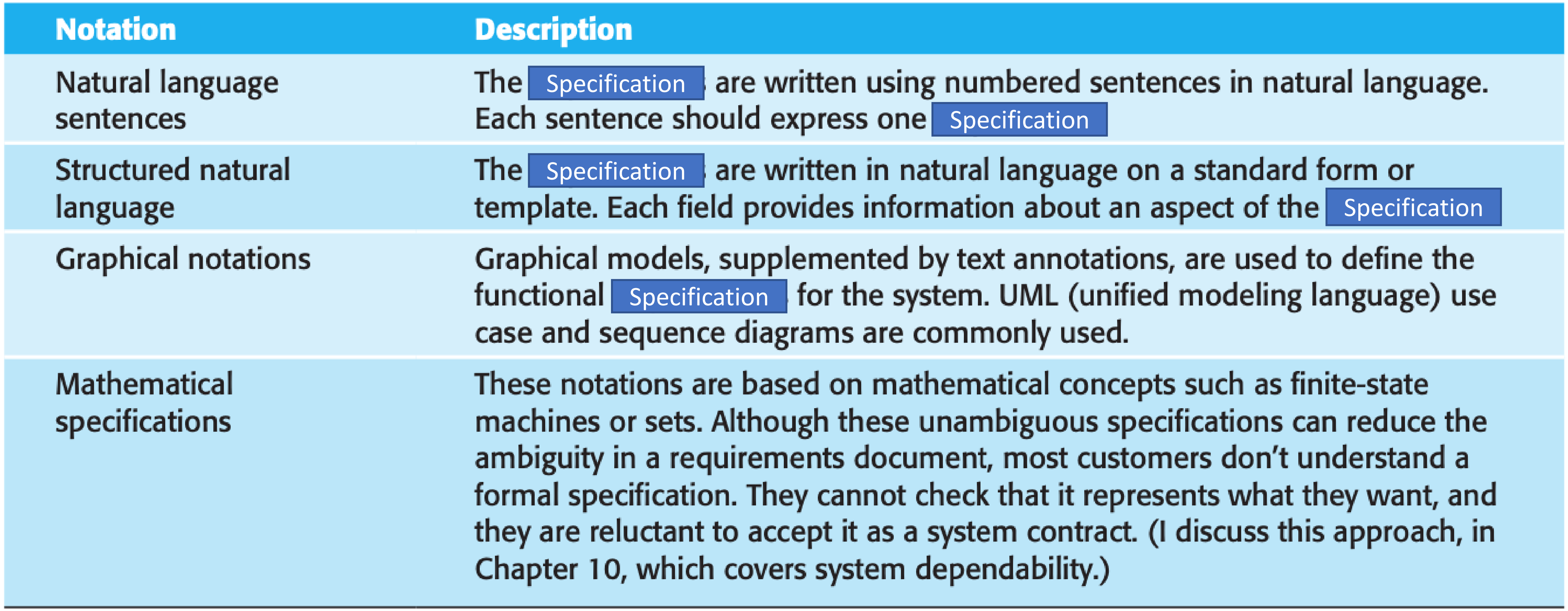


Specifications

"… detailed descriptions of the software system's functions, services and operational constraints … they … should define **exactly what is to be implemented**."

- A focus on WHAT should be built – but not necessarily HOW.
- Specifying: What a system will do to meet the user requirements.
- This means specifications should be tied to a requirement.
  - Specifications are often tabulated.
  - With a column saying which requirement ID it is for, etc.
  - Priorities, risks, etc.

Specifications



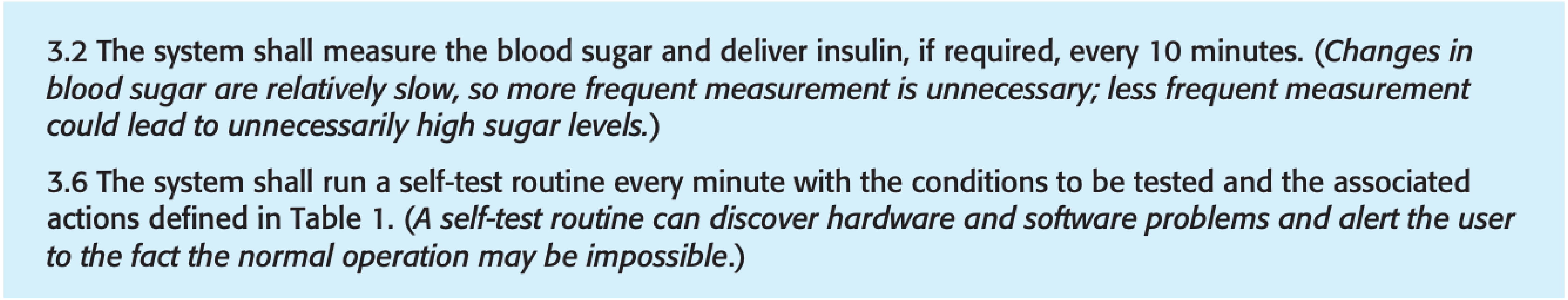
Notation	Description
Natural language sentences	The Specification are written using numbered sentences in natural language. Each sentence should express one Specification
Structured natural language	The Specification are written in natural language on a standard form or template. Each field provides information about an aspect of the Specification
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional Specification for the system. UML (unified modeling language) use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want, and they are reluctant to accept it as a system contract. (I discuss this approach, in Chapter 10, which covers system dependability.)

Figure 4.11 Notations for writing system Specification

Specifications – Natural Language

- Can be expressive, intuitive and universal.
- Can also be ambiguous, vague, and interpreted differently.
- Guidelines
  1. Use a standard format: 1 sentence, linked to a user requirements.
  2. Distinguish between mandatory (‘shall’) and desirable (‘should’).
  3. Emphasis important elements with bold, italic etc.
  4. Avoid jargon, unless clearly specified in a key words section.
  5. **Make sure the specification is measurable** in some form
     - Can’t be partly achieved or has a metric for successfully achieved.

Specifications – Natural Language



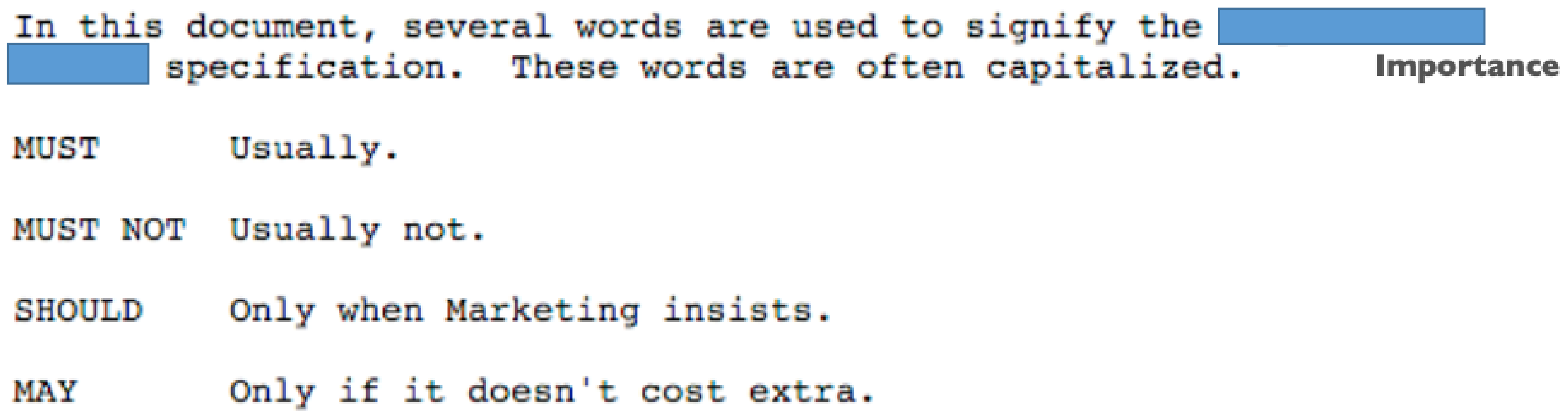
3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (*Changes in blood sugar are relatively slow, so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.*)

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

Figure 4.12 Example Specification for the insulin pump software system

Specifications

- From April 1999 – one project defining their terms.



In this document, several words are used to signify the [redacted] [redacted] specification. These words are often capitalized. **Importance**

MUST	Usually.
MUST NOT	Usually not.
SHOULD	Only when Marketing insists.
MAY	Only if it doesn't cost extra.

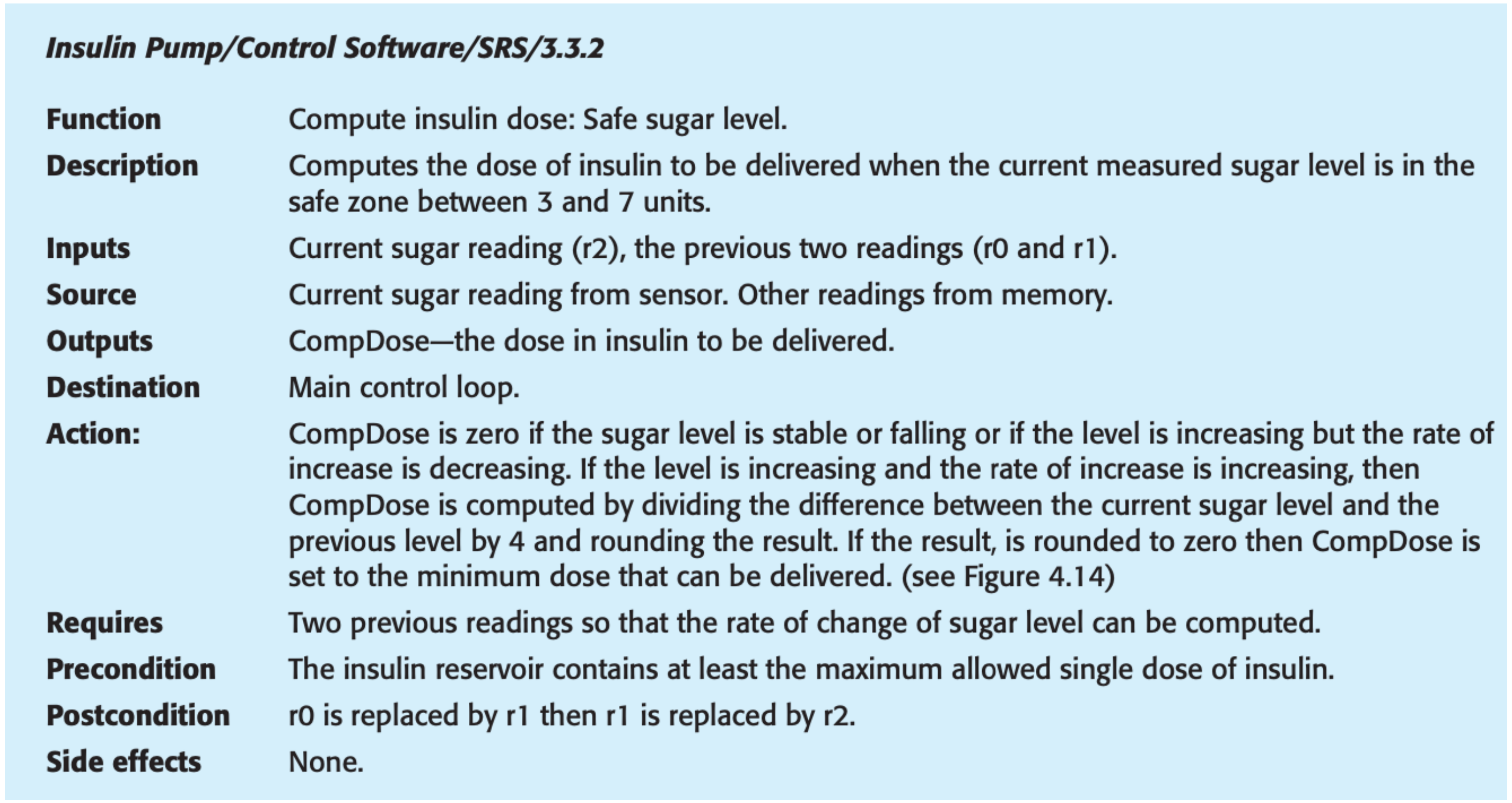
- Other formal definitions
  - https://tools.ietf.org/html/rfc2119

Specifications - Structured

When you need to be **exact**, about e.g., conditions or calculations.

- Go further than natural language specifications, to tabulate specifications, or put them in templates.
- Can be used to specify additional information
  - Associated logic in the function.
  - Inputs and outputs.
  - An explanations.
  - Conditions.
  - Side effects or relations to other functions.

Specifications - Structured



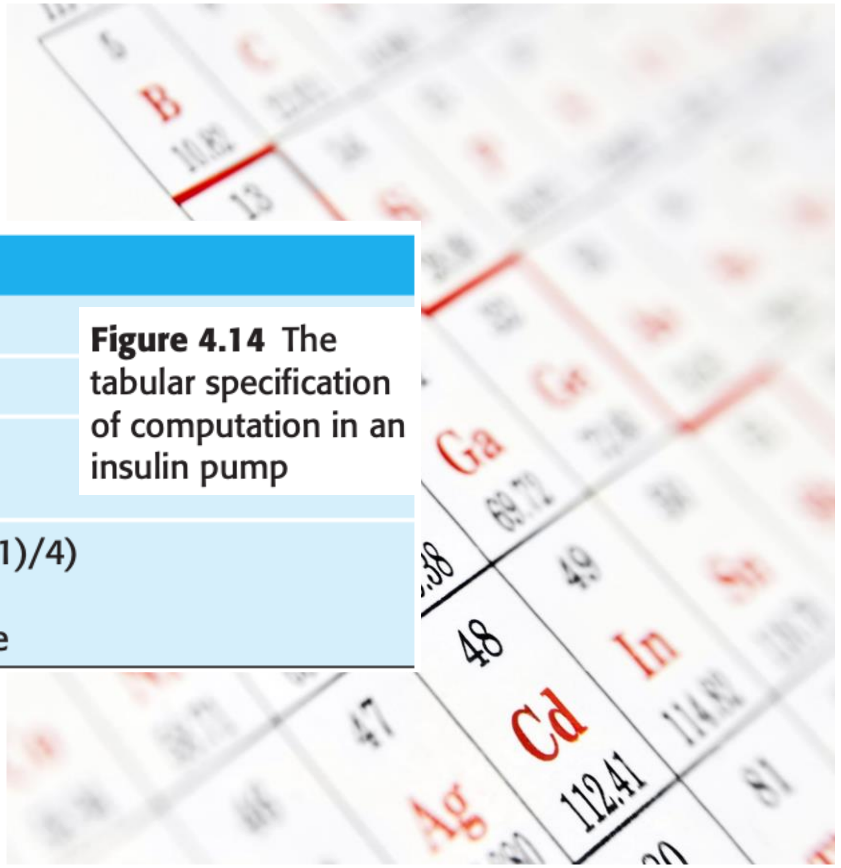
Insulin Pump/Control Software/SRS/3.3.2

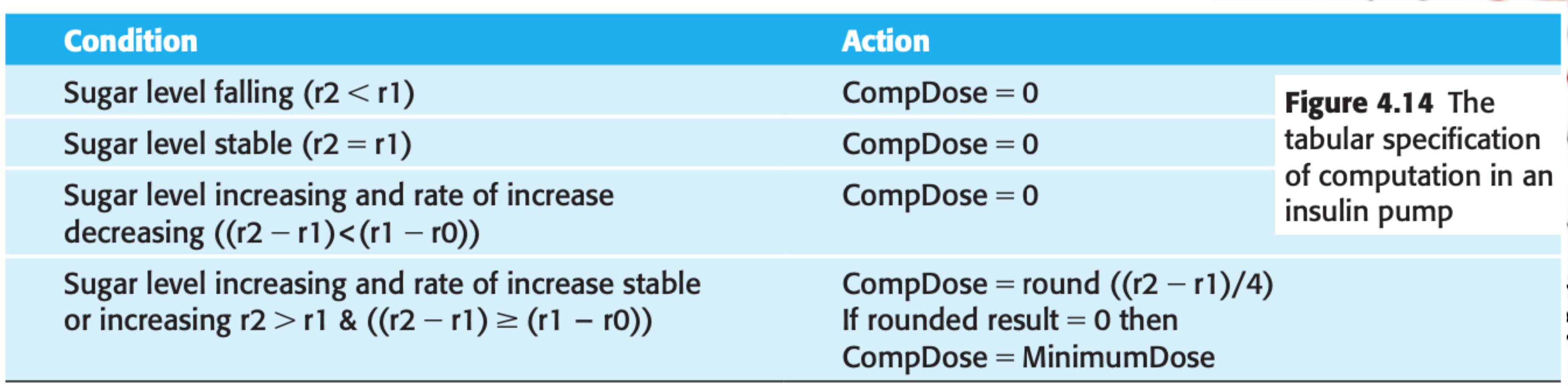
Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. 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.

Figure 4.13 The structured specification of a requirement for an insulin pump

Specifications – Structured – Tabular

- For e.g., when a specification has options that need explicitly listing.





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) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Figure 4.14 The tabular specification of computation in an insulin pump

- Often helps with mathematical ones too.

Specfications – Graphical

- UML models, diagrams, prototypes.
  - At this stage, though, the emphasis is on saying.
    - ”this is how it will work”.
    - “this is what you “mr. developer” should build”.
- It’s often easier to see a UML sequence diagram.
  - Than to read 30 decision-dependent specifications.
- So, when you find specifications are complex – visualize them.
  - Just as when you found a requirement to be complex.
- UML can be used for either.
  - And indeed, for post-development documentation (as said a few lectures ago).

State Diagram - Requirements

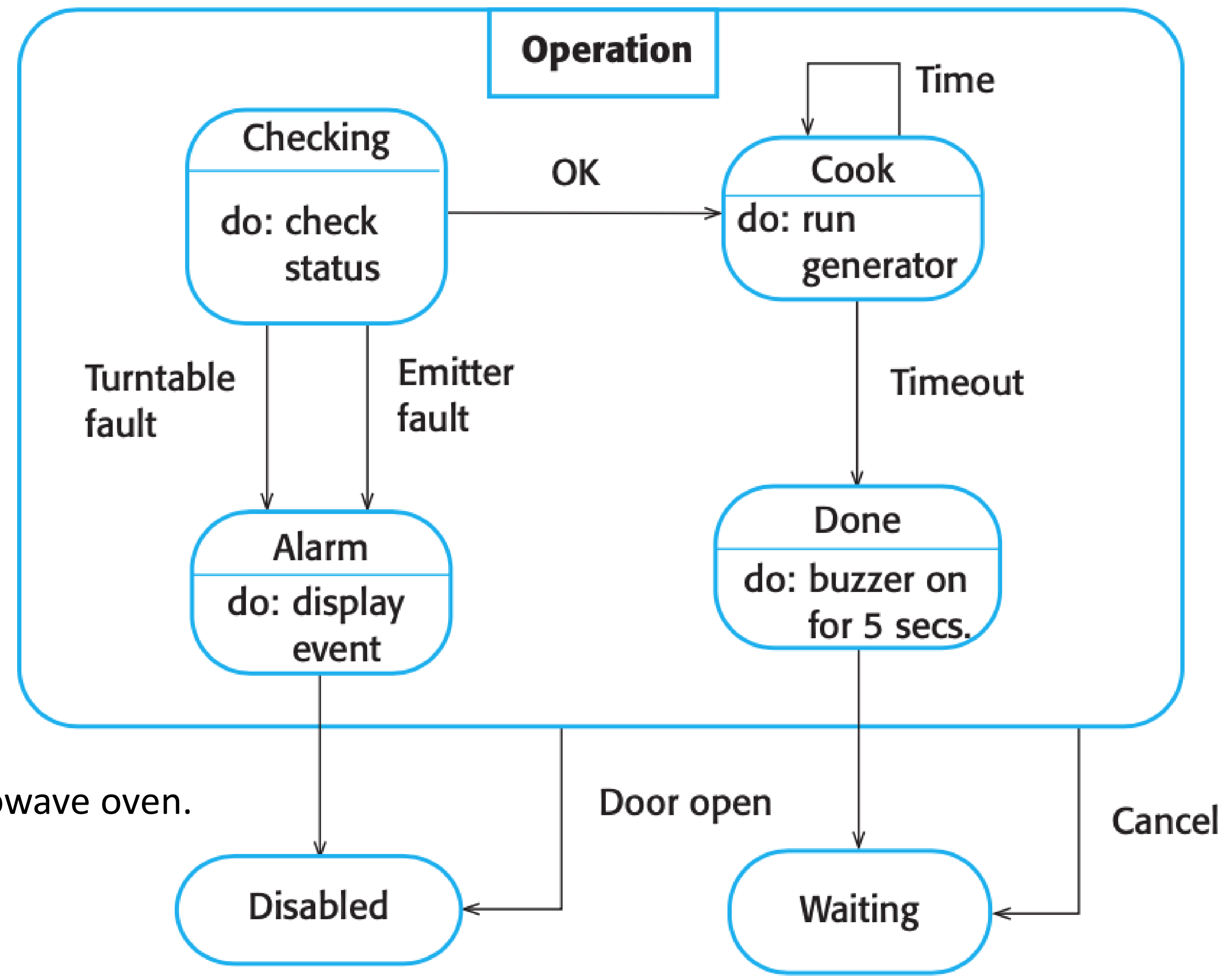
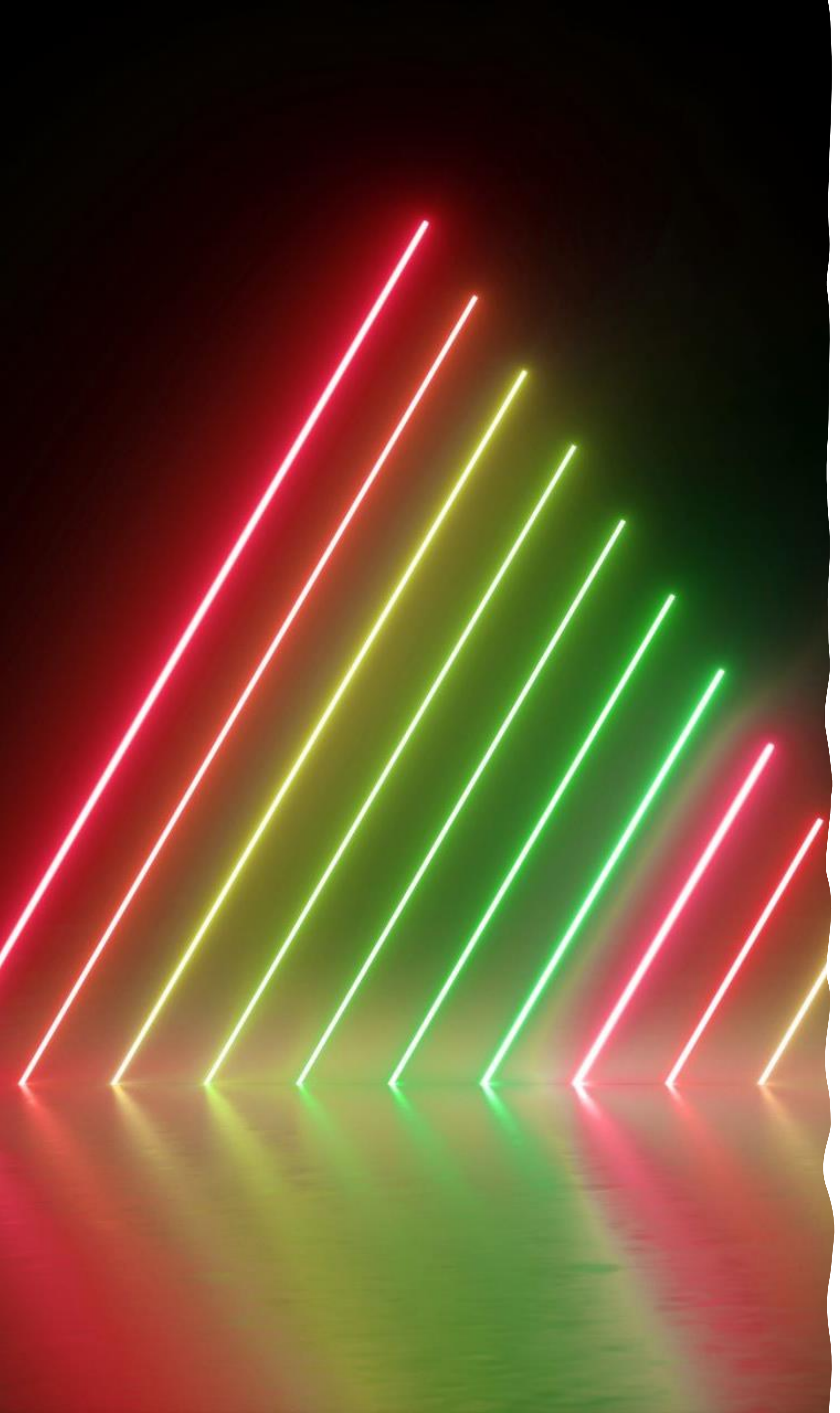


Fig. 1 The state diagram of microwave oven.



Good Specifications

Qualities of Good Specifications

- Analysing the quality of specifications is also important.
- Good quality specifications have a few qualities.
- Specifications: 3C principles

Correct	Complete
Necessary	Clear
Feasible	Consistent
Unique	Traceable
Concise	Verifiable

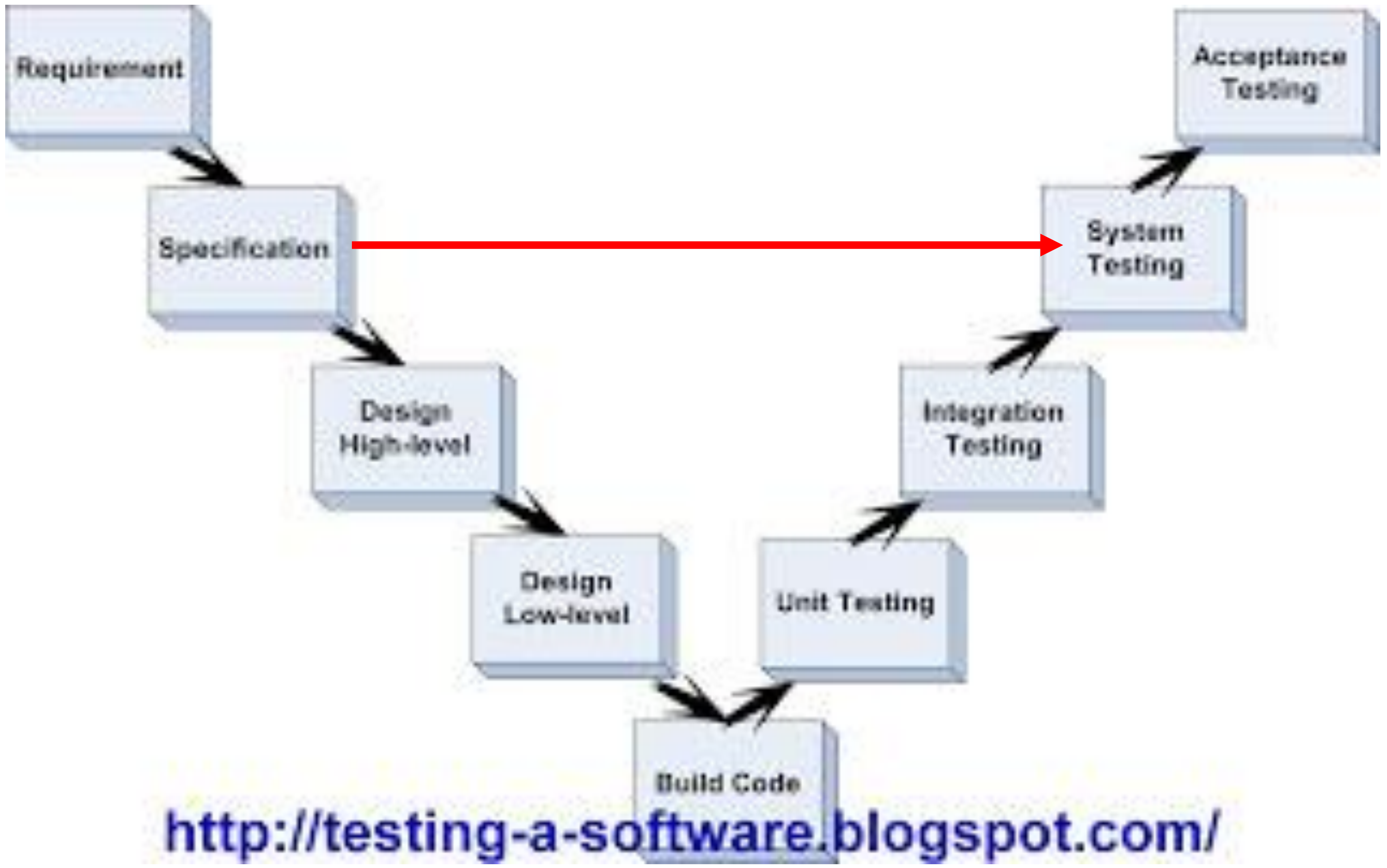
Specifications – Traceability

- It’s important that all specifications can be traced to user requirements.
- In reports, you should, for every specification.
  - State which user requirement(s) it is supporting.
- You may also categorise them by importance, difficulty etc.

Specifications – Testability

- Verification – did we build it right?
- To have implemented a single specification, you'll want to know you achieved what it specified.
  - You can only do this if specifications are testable.
- Can all the things you specify be tested and proved?
  - 'It must load fast' – unprovable.
  - 'it must load within 10s' – testable and provable.

Specifications – Testability

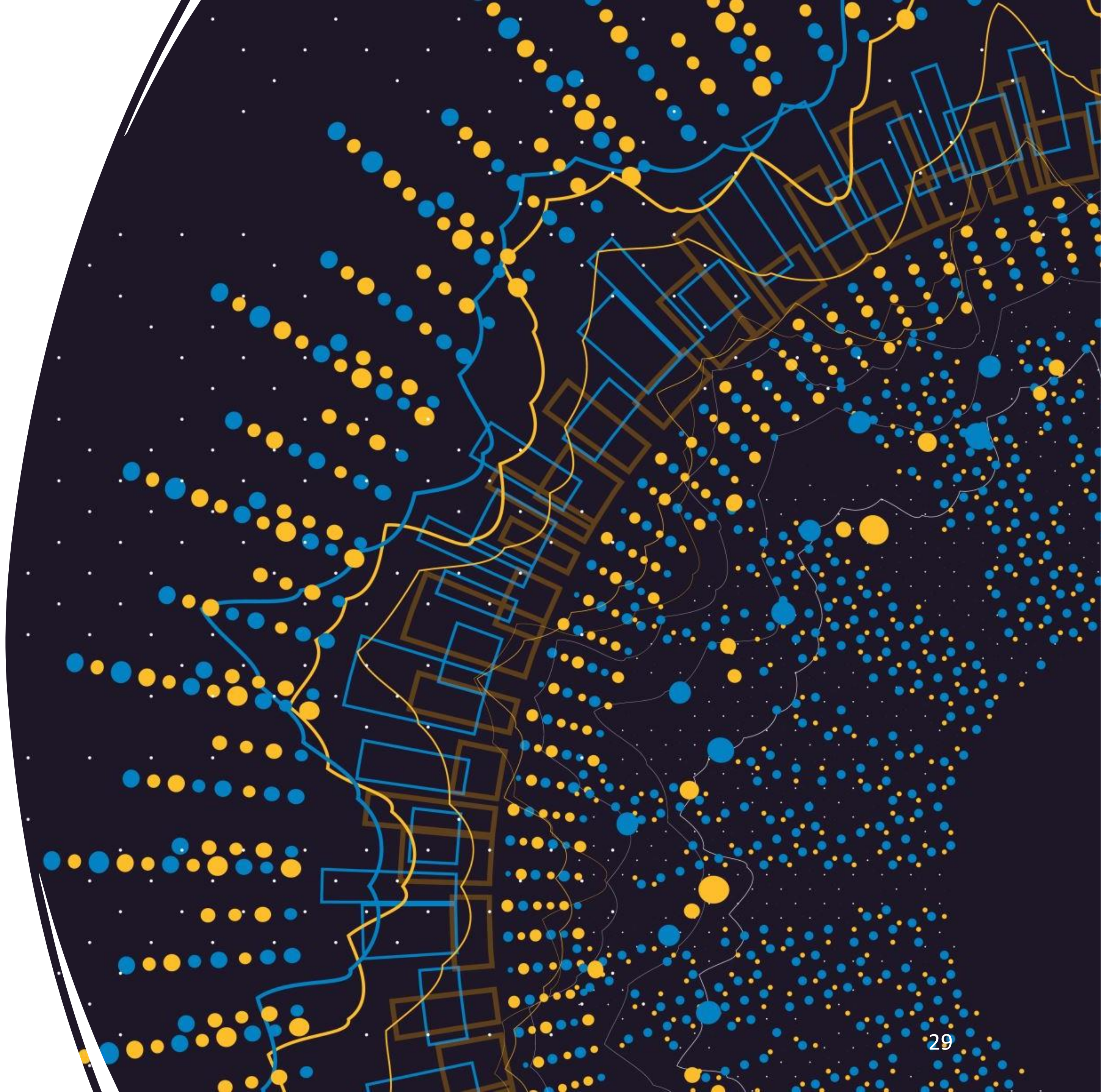


Specifications Reviews

"The specifications are analyzed systematically by a team of reviewers who **check for errors and inconsistencies**."

- There is a formal review process you can go through.
  - Several people in a room, **reading each specification a loud**.
- Each person takes a Role – to systematically review the specifications:
  - Validity checks (are the areas of functionality identified as necessary).
  - Consistency checks (do specifications conflict with one another).
  - Completeness checks (does it specify a coherent system or only parts of it).
  - Realism checks (can specifications actually be implemented).
  - Verifability checks (can specifications be tested).
- Are the system specifications: correct, necessary, important?

So, What [Else] Happens to Specifications?



Summary

"… it's **difficult to show that a set of [specifications] does in fact meet a user's need**. Users need to picture the system in operation and imagine how that system would fit into their work."

- Can a client/user understand a specification?
  - Probably not, and they aren't intended to see them.
- But we do need to show clients the plans for what we are building.
- A prototype combines:
  - The graphical specifications – UML models (also scenarios etc. from User Requirements).
  - The textual specifications –in the System Requirements Specifications).
- To demonstrate how they might work together in a system.
- Like specifications help to validate requirements – prototypes help to validate specifications.

Specifications – A Stage in the Process

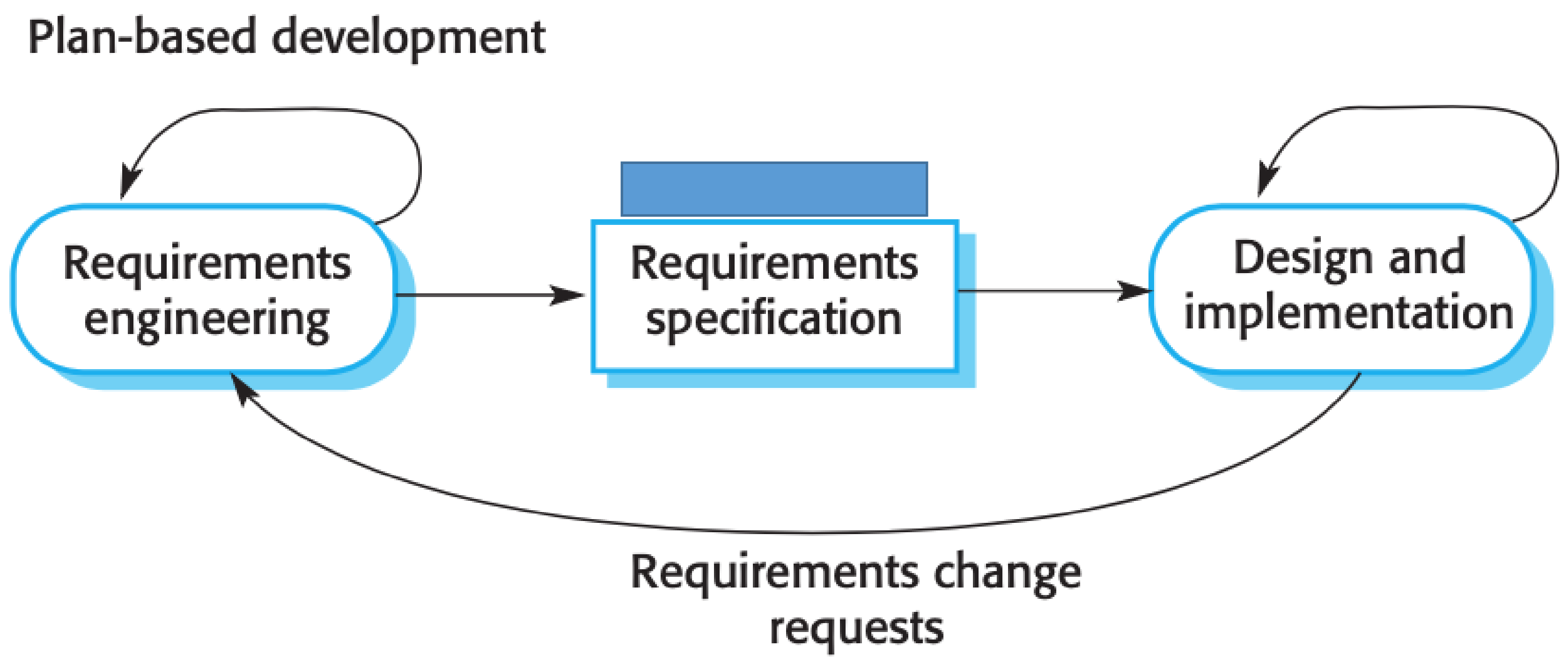

Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests

Figure 3.1 Plan-driven and agile development

Expected Readings

- How to write good specifications?
  - https://www.joelonsoftware.com/2000/10/15/painless-functional-specifications-part-4-tips/

Optional Readings

- NASA – Prototyping Software?
  - https://www.justinmind.com/learn-ux-design/nasa-ux-design-ron-kim

THANK
YOU