投票 最多可选1项

设置

Are you ready?

A Yes

B No

提交

Software Engineering

Part 2 Modeling

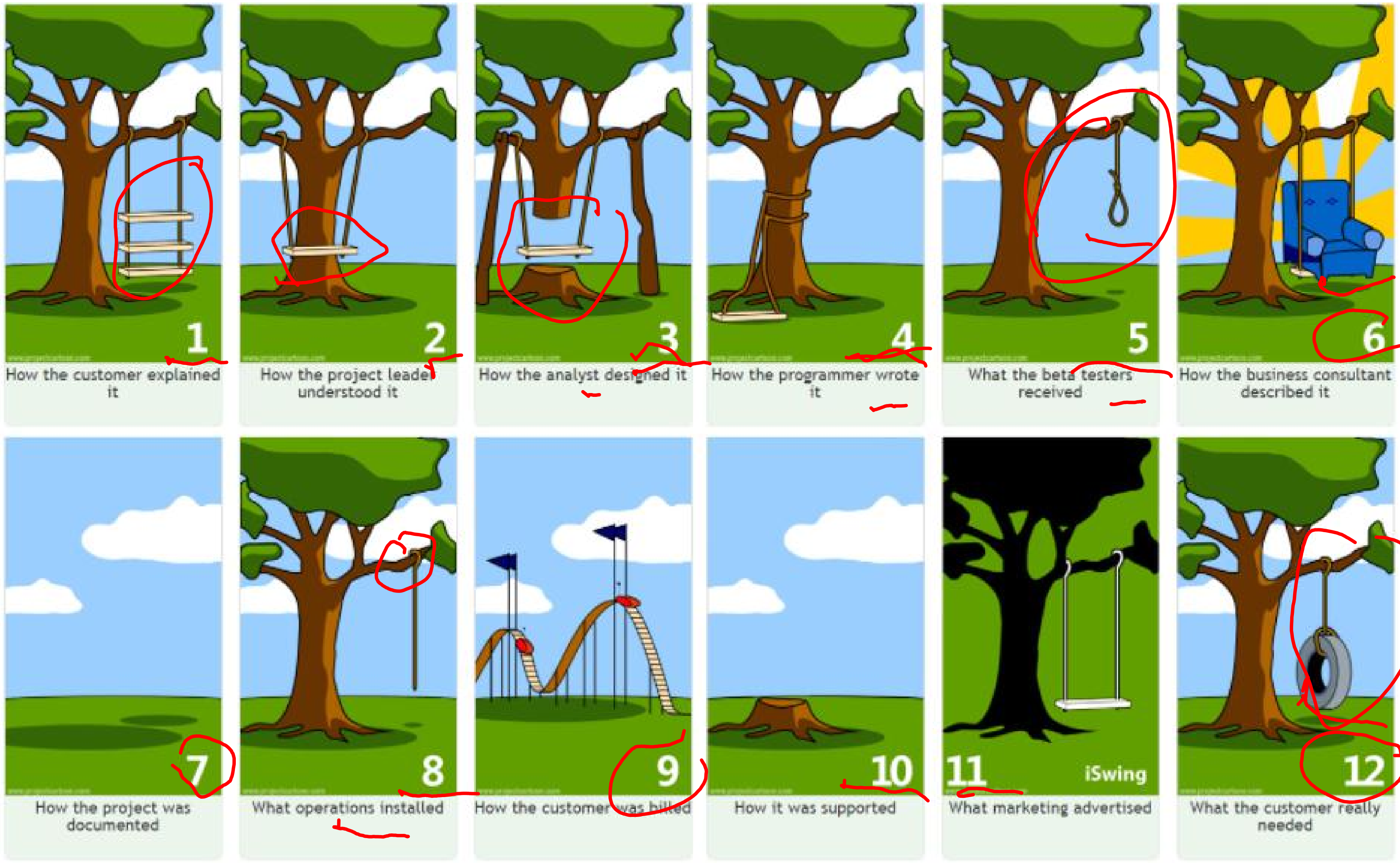
Chapter 8 Understanding Requirement

Reproduced by Ning Li , 2022

Contents

Contents

A Tree Swing Story


1
How the customer explained it
2
How the project leader understood it
3
How the analyst designed it
4
How the programmer wrote it
5
What the beta testers received
6
How the business consultant described it
7
How the project was documented
8
What operations installed
9
How the customer was billed
10
How it was supported
11
iSwing
What marketing advertised
12
What the customer really needed

Quiz

In 1994, the Standish Group surveyed over 350 companies about their over 8000 software projects to find out how well they were faring.

1/3 projects: canceled ;

only 9%: delivered on time/within budget(large companies)

only 16%: met those criteria (small companies)

Why ? The three top factors (by survey):

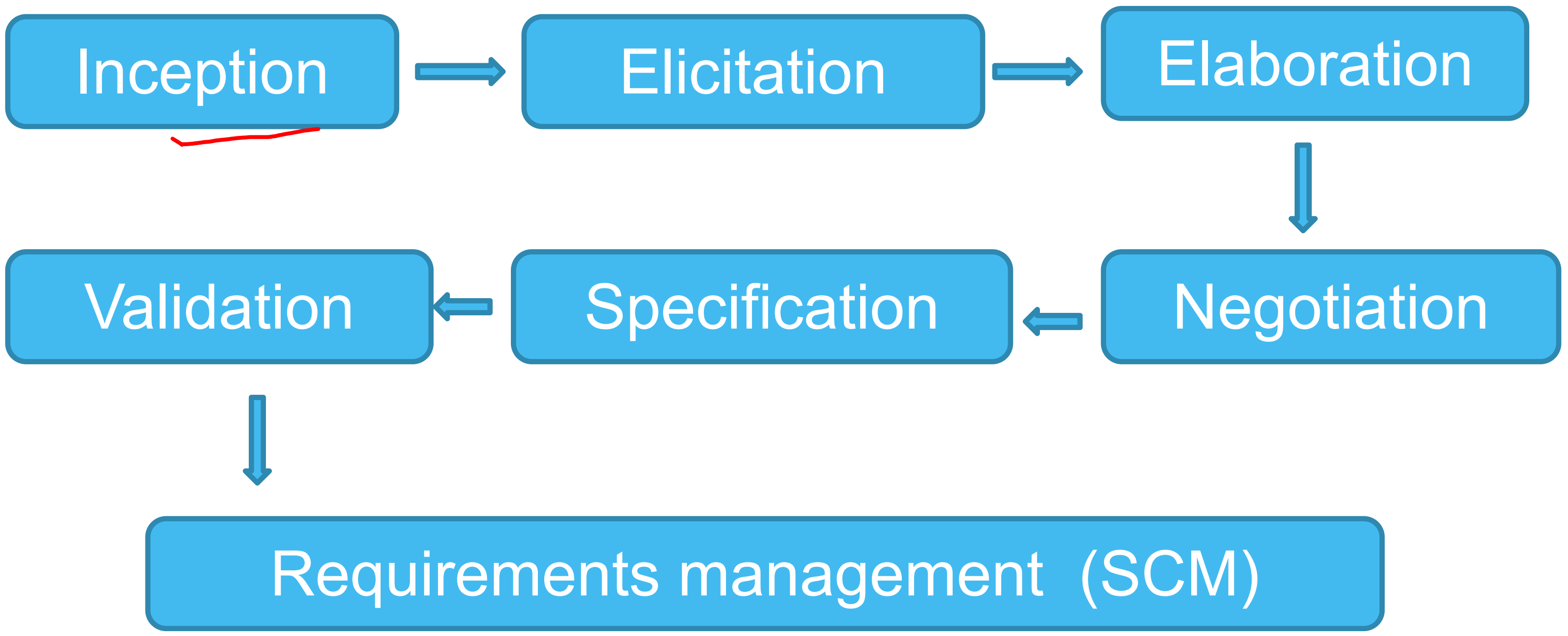
1) Incomplete requirements (C)

2) Lack of user involvement (12.4%)

3) Lack of resources (10.6%)

A) 26.3%   B) 16.5%   [✓ C) 13.1%]

8.1 Requirements Engineering



8.1 Requirements Engineering

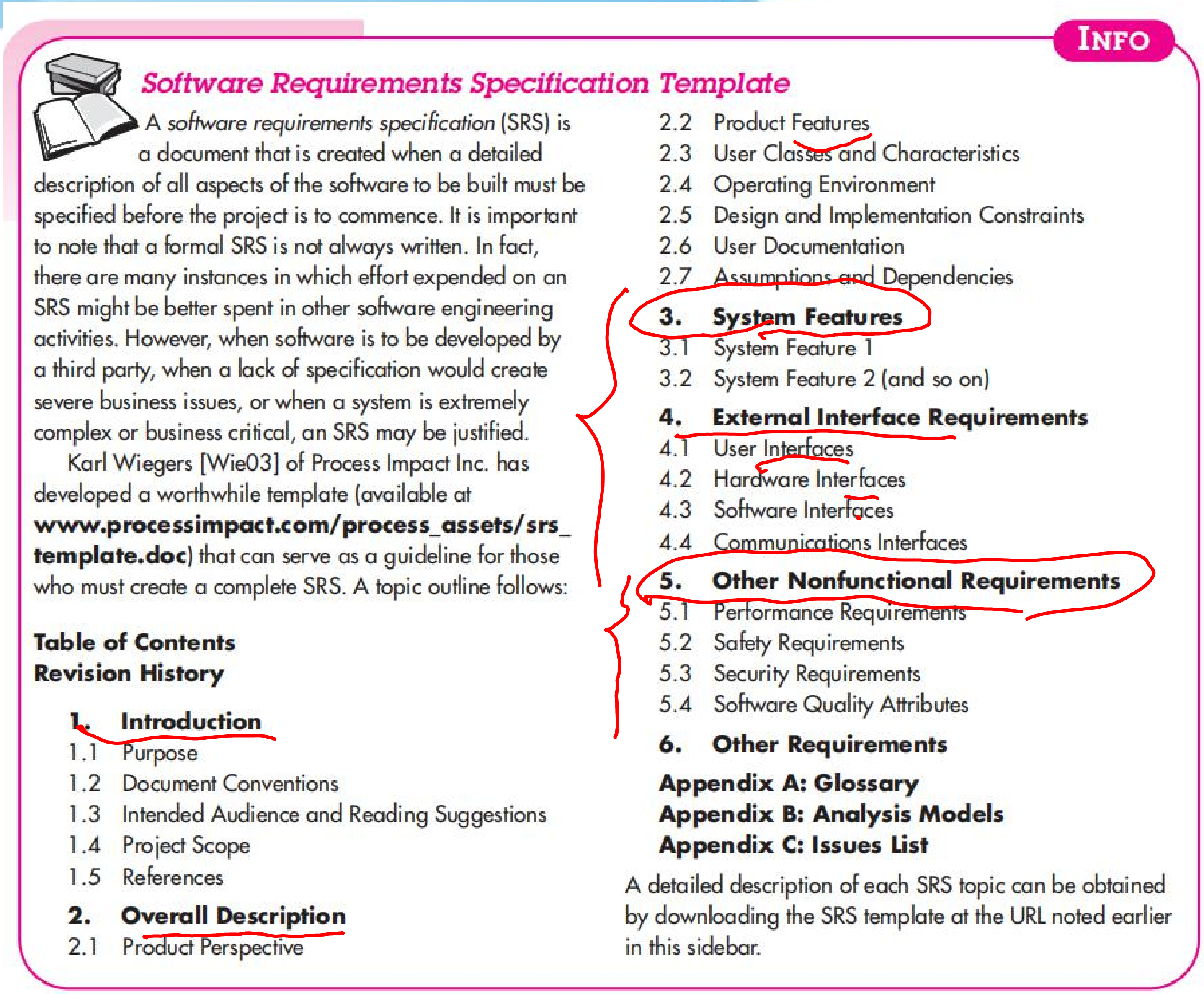
1. Inception — ask a set of questions that establish …
   - basic understanding of the problem
   - the people who want a solution
   - the nature of the solution that is desired, and
   - the effectiveness of preliminary communication and collaboration between the customer and the developer
2. Elicitation — elicit requirements from all stakeholders
3. Elaboration — create a refined model that identifies data, function and behavioral requirements
4. Negotiation — agree on a deliverable system that is realistic for developers and customers

8.1 Requirements Engineering

5. Specification — can be anyone (or more) of the following:
   - A written document
   - A set of models
   - A formal mathematical
   - A collection of user scenarios (use-cases)
   - A prototype
6. Validation — a review mechanism that looks for
   - errors in content or interpretation
   - areas where clarification may be required
   - missing information
   - inconsistencies (a major problem when large products or systems are engineered)
   - conflicting or unrealistic (unachievable) requirements.
7. Requirements management

git

8.1 Requirements Engineering



INFO

Software Requirements Specification Template

A *software requirements specification* (SRS) is a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at **www.processimpact.com/process_assets/srs_template.doc**) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents
Revision History

1. Introduction
1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Project Scope
1.5 References

2. Overall Description
2.1 Product Perspective
2.2 Product Features
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 User Documentation
2.7 Assumptions and Dependencies

3. System Features
3.1 System Feature 1
3.2 System Feature 2 (and so on)

4. External Interface Requirements
4.1 User Interfaces
4.2 Hardware Interfaces
4.3 Software Interfaces
4.4 Communications Interfaces

5. Other Nonfunctional Requirements
5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted earlier in this sidebar.

8.2 Inception: Establishing the groundwork

- ✓ Identify stakeholders
  - Who else do you think I should talk to?
- ✓ Recognize multiple points of view
  - Who else do you think I should talk to?
- ✓ Work toward collaboration
  - Commonality & conflict; strong "project champion"

8.2 Inception: Establishing the groundwork

- ✓ The first questions
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution?
  - Is there another source for the solution that you need?
- ✓ The next questions
  - How would you characterize “good” output that would be generated by a successful solution?
  - What problem(s) will this solution address?
  - Can you show me (or describe) the business environment in which the solution will be used?
  - Will special performance issues or constraints affect the way the solution is approached?

8.3 Eliciting Requirements

- Meetings are conducted and attended by both software engineers and customers
- Rules for preparation and participation are established an agenda is suggested
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers, chat room or virtual forum) is used
- The goal is
  - to identify the problem
  - propose elements of the solution
  - negotiate different approaches, and
  - specify a preliminary set of solution requirements

8.3 Eliciting Requirements

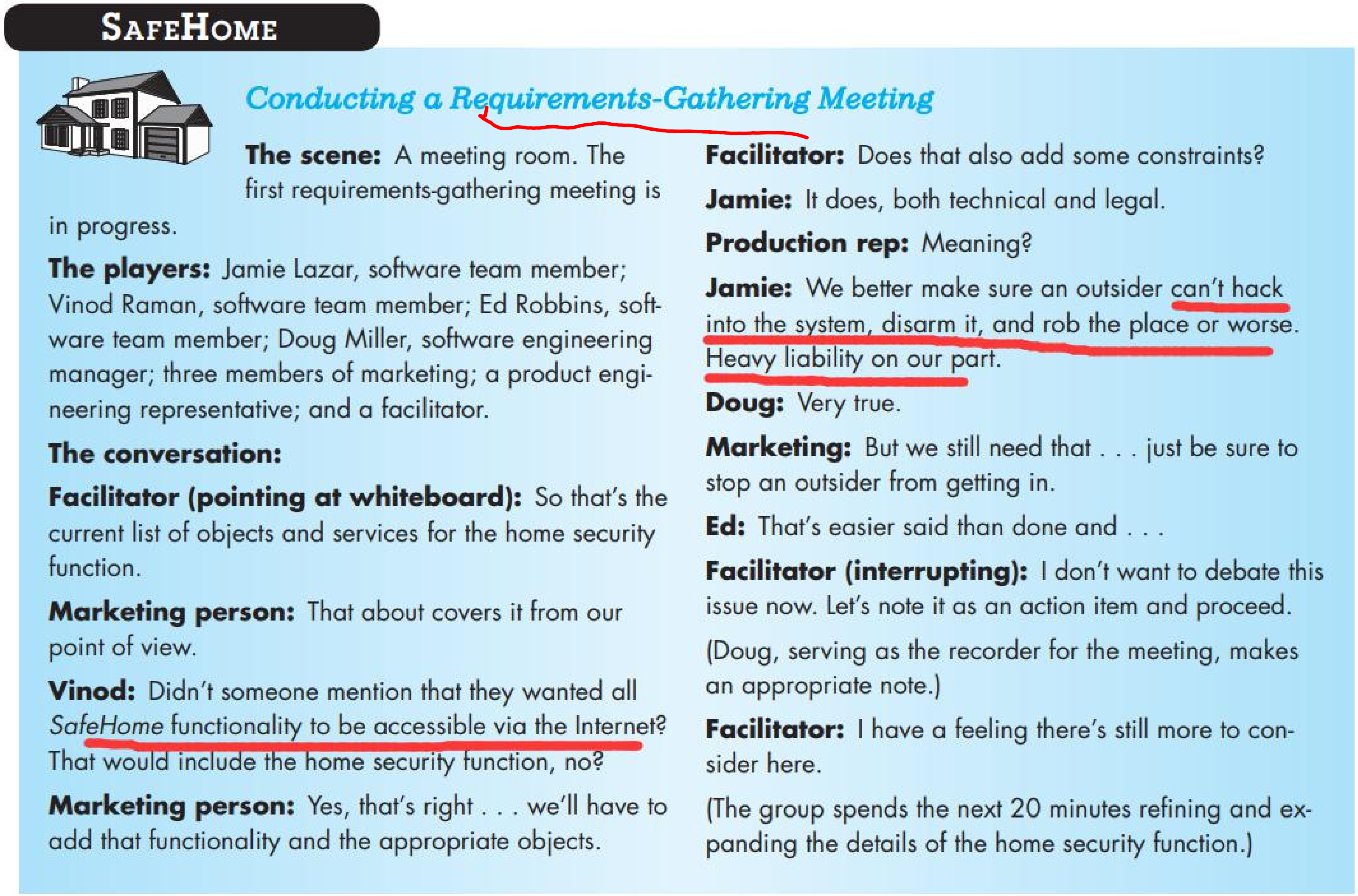
- Non-Functional Requirment (NFR) – quality attribute, performance attribute, security attribute, or general system constraint.
- A two phases process is used to determine which NFR's are compatible:
  1. Create a matrix using each NFR as a column heading and the system SE guidelines a row labels
  2. For the team to prioritize each NFR using a set of decision rules to decide which to implement by classifying each NFR and guideline pair as complementary, overlapping, conflicting, or independent

8.3 Eliciting Requirements

Let’s try a role play for requirement!

8.3 Eliciting Requirements

P145



SAFEHOME

Conducting a Requirements-Gathering Meeting

The scene: A meeting room. The first requirements-gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

1 **Facilitator (pointing at whiteboard):** So that's the current list of objects and services for the home security function.

2 **Marketing person:** That about covers it from our point of view.

3 **Vinod:** Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

2 **Marketing person:** Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints? 1

Jamie: It does, both technical and legal. 4

Production rep: Meaning? 5

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part. 4

Doug: Very true. 6

Marketing: But we still need that . . . just be sure to stop an outsider from getting in. 2

Ed: That's easier said than done and . . . 5

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed. 1

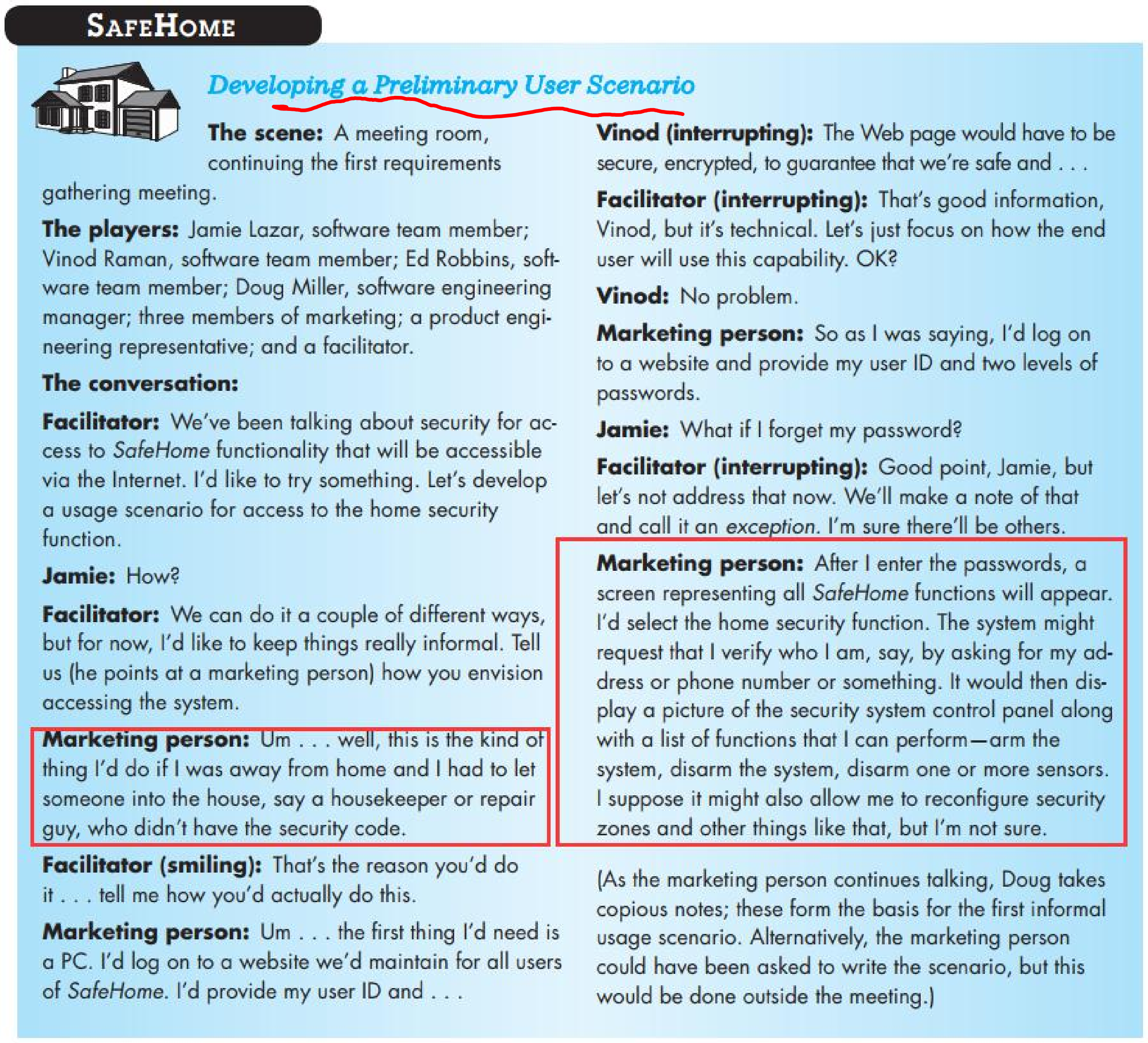
(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here. 1

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

8.3 Eliciting Requirements

P147



SafeHome

Developing a Preliminary User Scenario

The scene: A meeting room, continuing the first requirements gathering meeting.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

1 **Facilitator:** We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

2 **Jamie:** How?

1 **Facilitator:** We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

3 **Marketing person:** Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

1 **Facilitator (smiling):** That's the reason you'd do it . . . tell me how you'd actually do this.

3 **Marketing person:** Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user ID and . . .

Vinod (interrupting): The Web page would have to be secure, encrypted, to guarantee that we're safe and . . . 4

Facilitator (interrupting): That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK? 1

Vinod: No problem. 4

Marketing person: So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords. 3

Jamie: What if I forget my password? 2

Facilitator (interrupting): Good point, Jamie, but let's not address that now. We'll make a note of that and call it an *exception*. I'm sure there'll be others. 1

Marketing person: After I enter the passwords, a screen representing all *SafeHome* functions will appear. I'd select the home security function. The system might request that I verify who I am, say, by asking for my address or phone number or something. It would then display a picture of the security system control panel along with a list of functions that I can perform—arm the system, disarm the system, disarm one or more sensors. I suppose it might also allow me to reconfigure security zones and other things like that, but I'm not sure. 3

(As the marketing person continues talking, Doug takes copious notes; these form the basis for the first informal usage scenario. Alternatively, the marketing person could have been asked to write the scenario, but this would be done outside the meeting.)

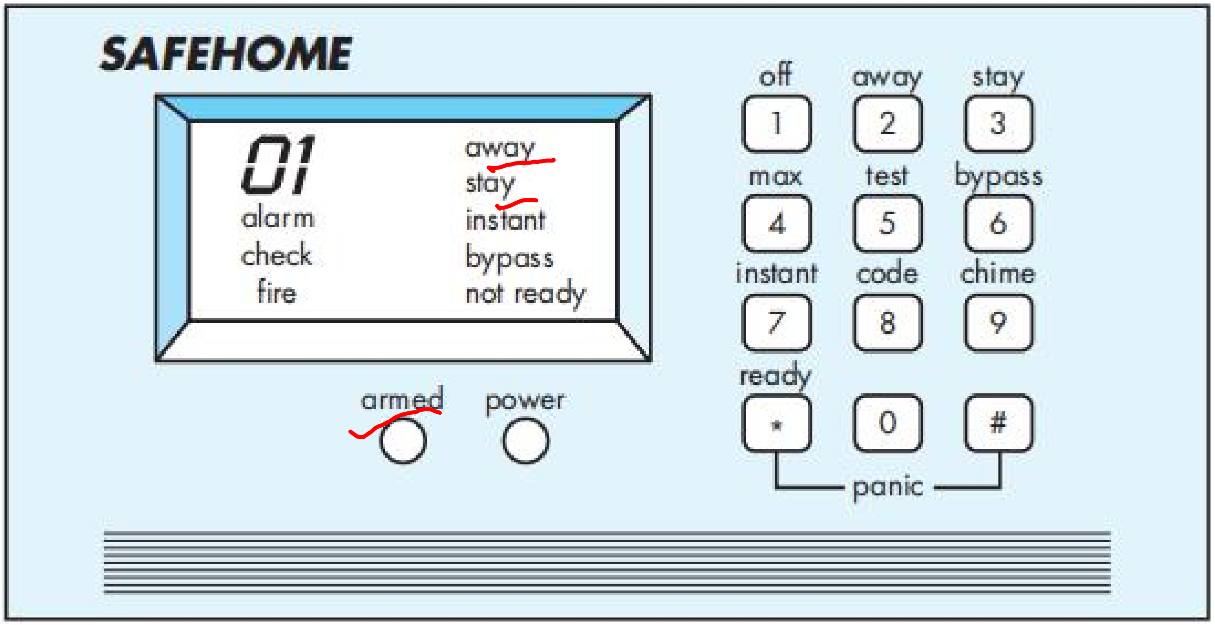
8.4 Use-Cases

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor’s goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor’s interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

8.4 Use-Cases

FIGURE 8.1

SafeHome control panel



8.4 Use-Case

Use case:	*InitiateMonitoring*
Primary actor:	Homeowner.
Goal in context:	To set the system to monitor sensors when the homeowner leaves the house or remains inside.
Preconditions:	System has been programmed for a password and to recognize various sensors.
Trigger:	The homeowner decides to “set” the system, that is, to turn on the alarm functions.

8.4 Use-Case

Scenario:

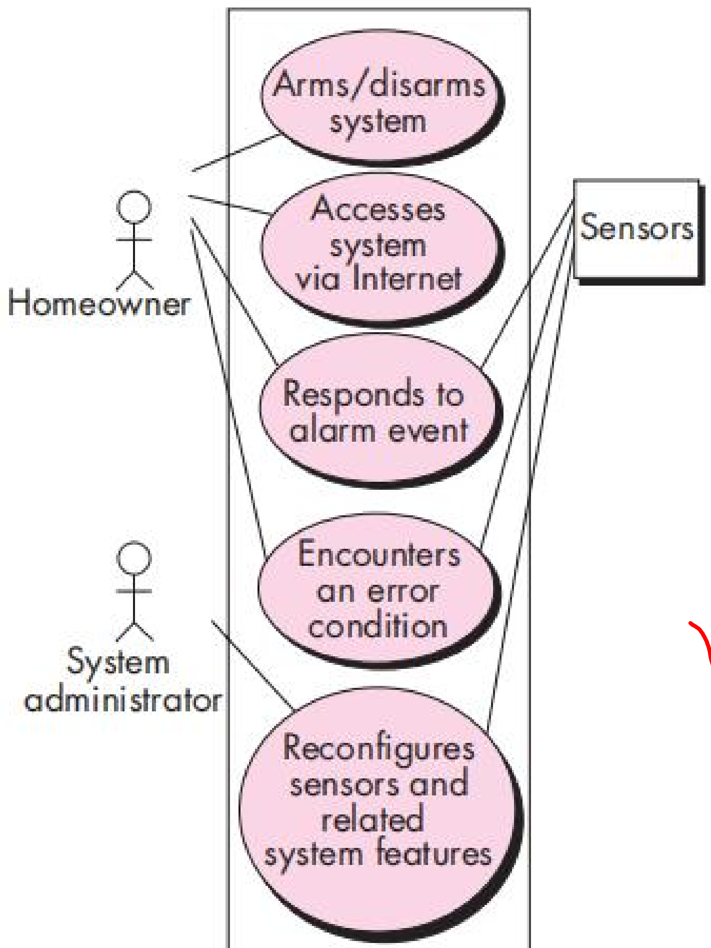
1. Homeowner: observes control panel
2. Homeowner: enters password
3. Homeowner: selects "stay" or "away"
4. Homeowner: observes read alarm light to indicate that *SafeHome* has been armed

Exceptions:

1. Control panel is *not ready*: homeowner checks all sensors to determine which are open; closes them.
2. Password is incorrect (control panel beeps once): homeowner reenters correct password.
3. Password not recognized: monitoring and response subsystem must be contacted to reprogram password.
4. *Stay* is selected: control panel beeps twice and a *stay* light is lit; perimeter sensors are activated.
5. *Away* is selected: control panel beeps three times and an *away* light is lit; all sensors are activated.

8.4 Use-Case Diagram

UML


Homeowner
System administrator
Arms/disarms system
Accesses system via Internet
Responds to alarm event
Encounters an error condition
Reconfigures sensors and related system features
Sensors

✓

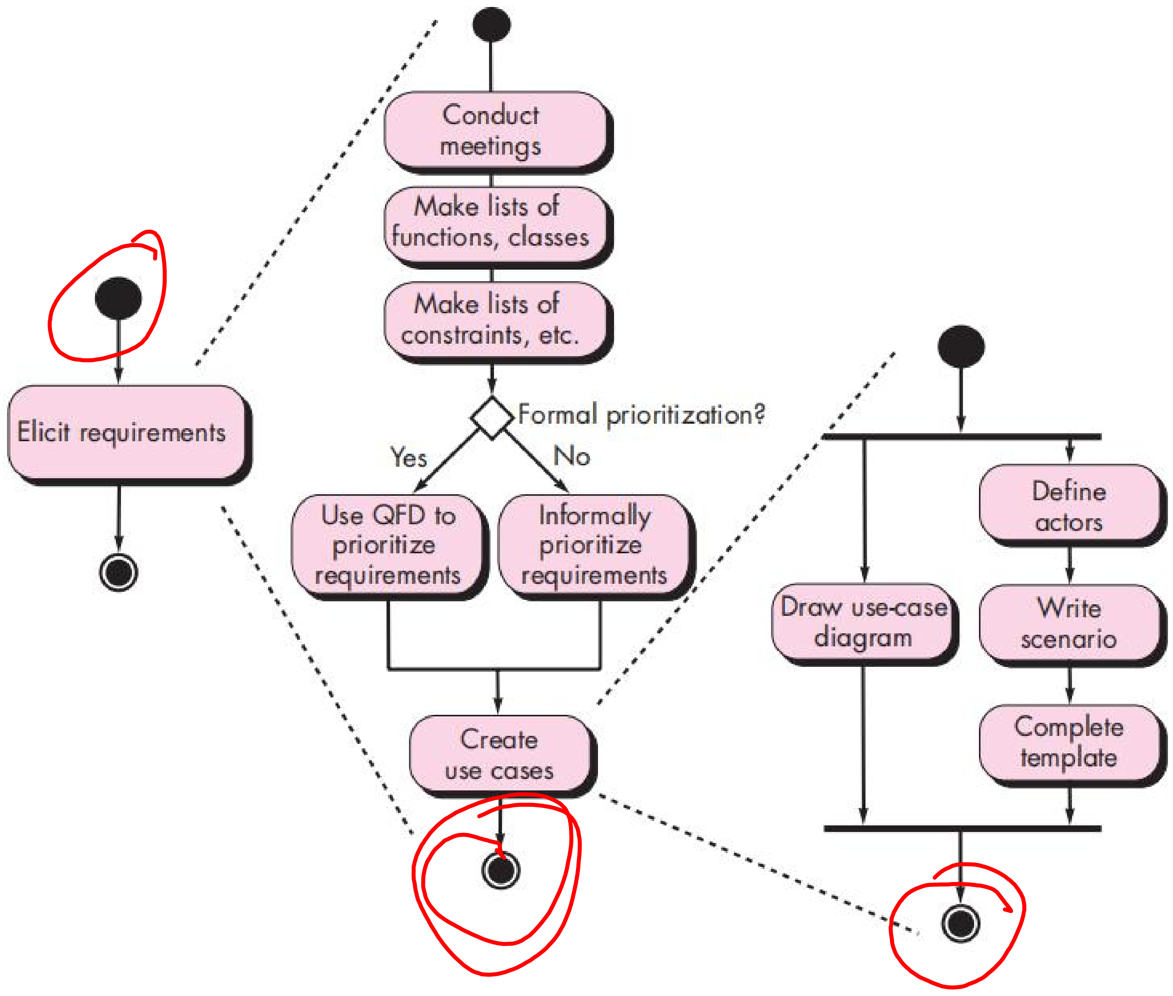
8.5 Building the Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

use-case

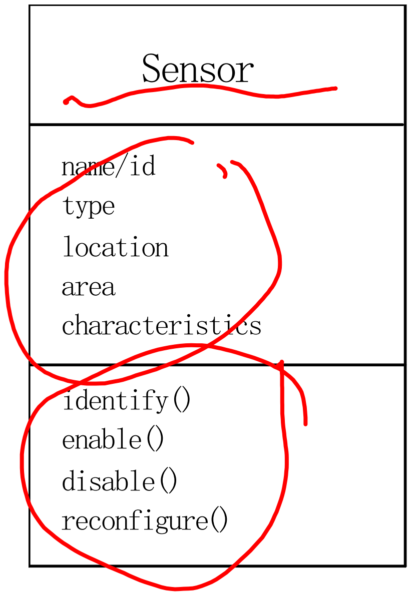
Sequence

8.5.1 Activity Diagrams

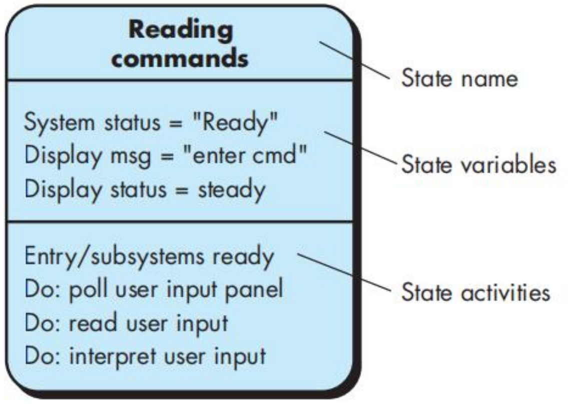


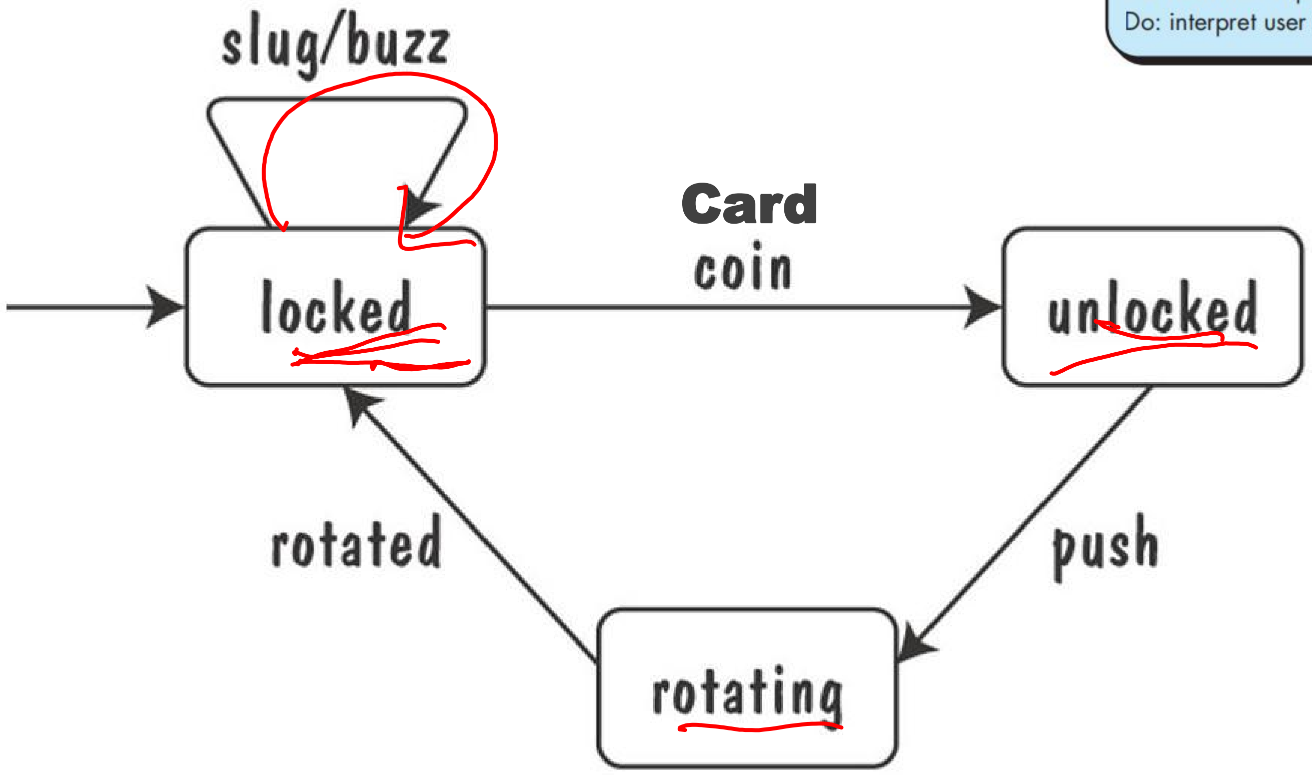
8.5.1 Class Diagram

From the *SafeHome* system …



8.5.1 State Diagram





8.5.2 Analysis Patterns

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

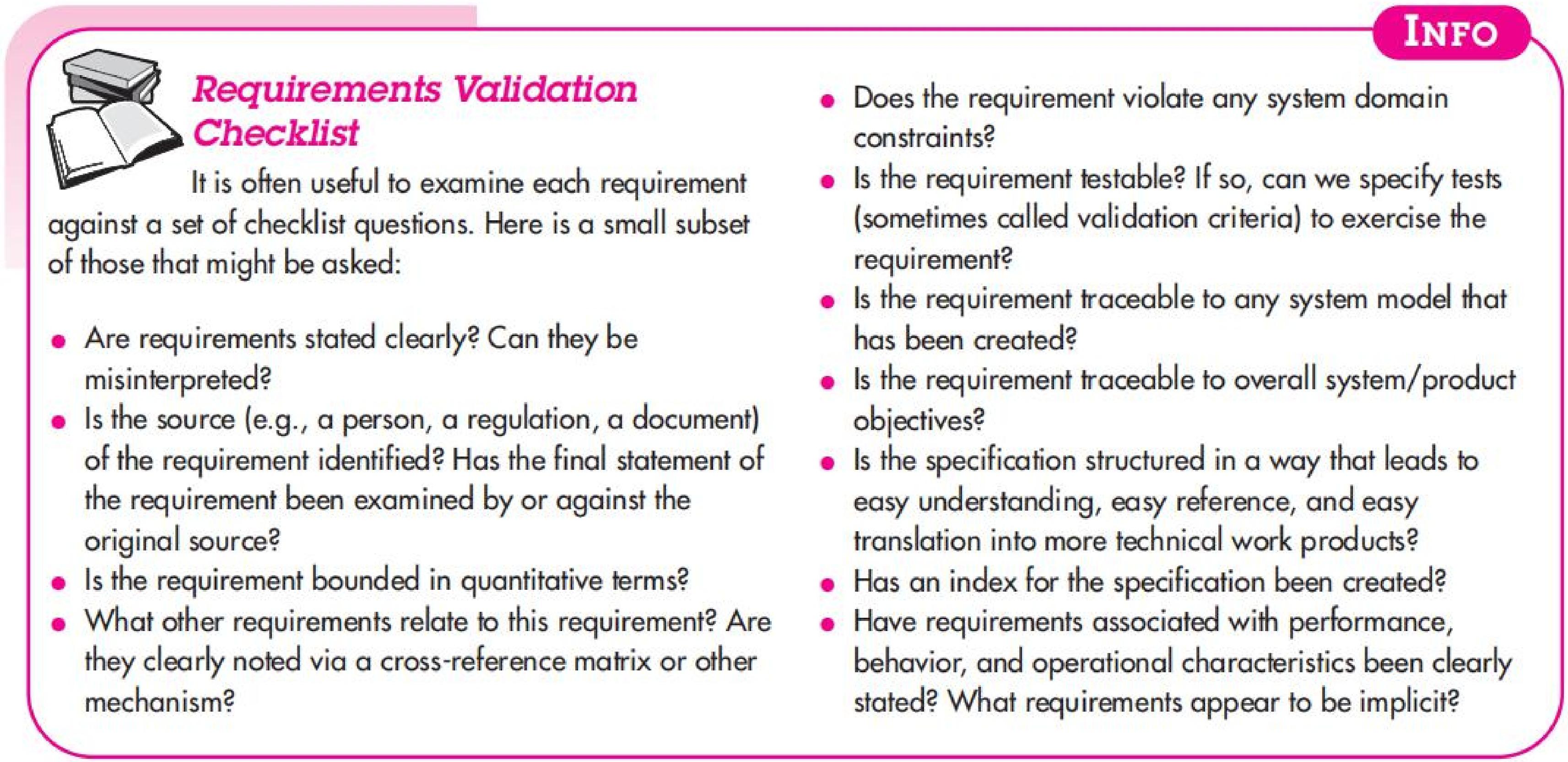
Known uses: Examples of uses within actual systems.

Related patterns: One or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern; (2) it is structurally similar to the named pattern; (3) it is a variation of the named pattern.

8.5.3 Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders “win conditions”
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to “win-win”

8.5.3 Requirements Engineering



INFO

Requirements Validation Checklist

It is often useful to examine each requirement against a set of checklist questions. Here is a small subset of those that might be asked:

- Are requirements stated clearly? Can they be misinterpreted?
- Is the source (e.g., a person, a regulation, a document) of the requirement identified? Has the final statement of the requirement been examined by or against the original source?
- Is the requirement bounded in quantitative terms?
- What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?
- Does the requirement violate any system domain constraints?
- Is the requirement testable? If so, can we specify tests (sometimes called validation criteria) to exercise the requirement?
- Is the requirement traceable to any system model that has been created?
- Is the requirement traceable to overall system/product objectives?
- Is the specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?
- Has an index for the specification been created?
- Have requirements associated with performance, behavior, and operational characteristics been clearly stated? What requirements appear to be implicit?

8.5.3 Validating Requirements

1. Is each requirement consistent with the overall objective for the system/product?
2. Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
4. Is each requirement bounded and unambiguous?
5. Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
6. Do any requirements conflict with other requirements?

8.5.3 Validating Requirements

7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?
9. Does the requirements model properly reflect the information, function and behavior of the system to be built.
10. Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
11. Have requirements patterns been used to simplify the requirements model. Have all patterns been properly validated? Are all patterns consistent with customer requirements?

THE END