47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

Requirements management: tracking and evolve

Ilídio Oliveira

V2020/10/30 | TP-08

universidade de aveiro
departamento de eletrónica,
telecomunicações e informática
deti

Learning objectives for this lecture

- Distinguish user-centric and product-centric requirements specification
- Describe requirements documentation techniques
- Understand the relation between requirements and use cases
- Identify the requirements related disciplines and activities in the OpenUP
- Explain the meaning of evolutionary requirements concept
- Describe the types of information the Analyst collects in the requirements engineering activities
- Explain how to implement requirements tracking activities

Requirements engineering activities

Discover/develop

Requirements gathering and elicitation

Critical aspect in the overall lifecycle

Manage

Document, track and evolve

Apply a change process (assess the risk,…)

The approach to requirements development should be adapted to the kind of project in hands.

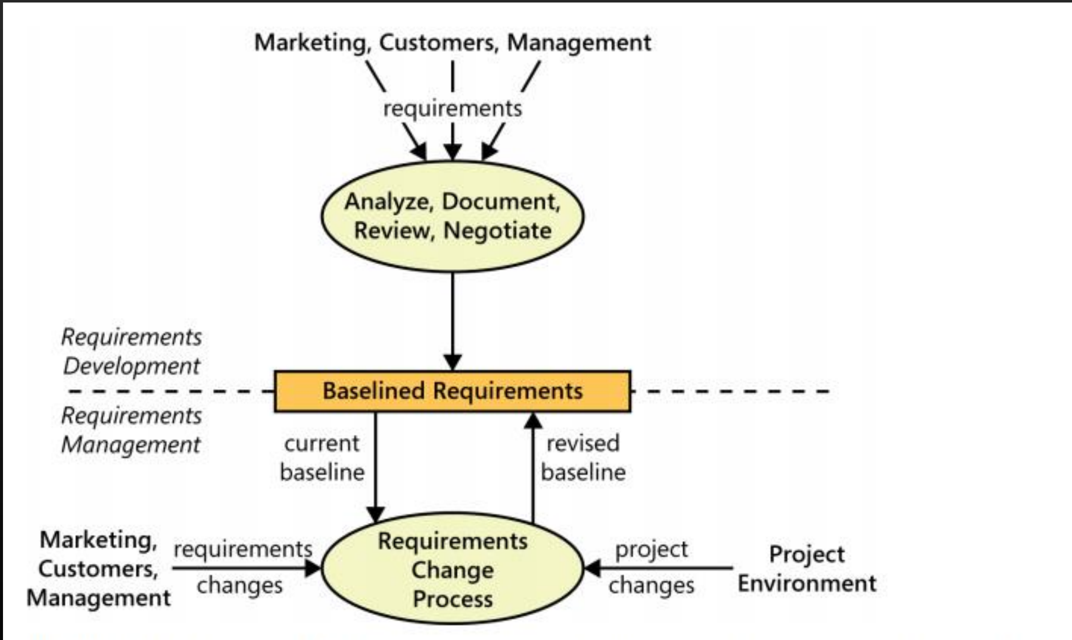


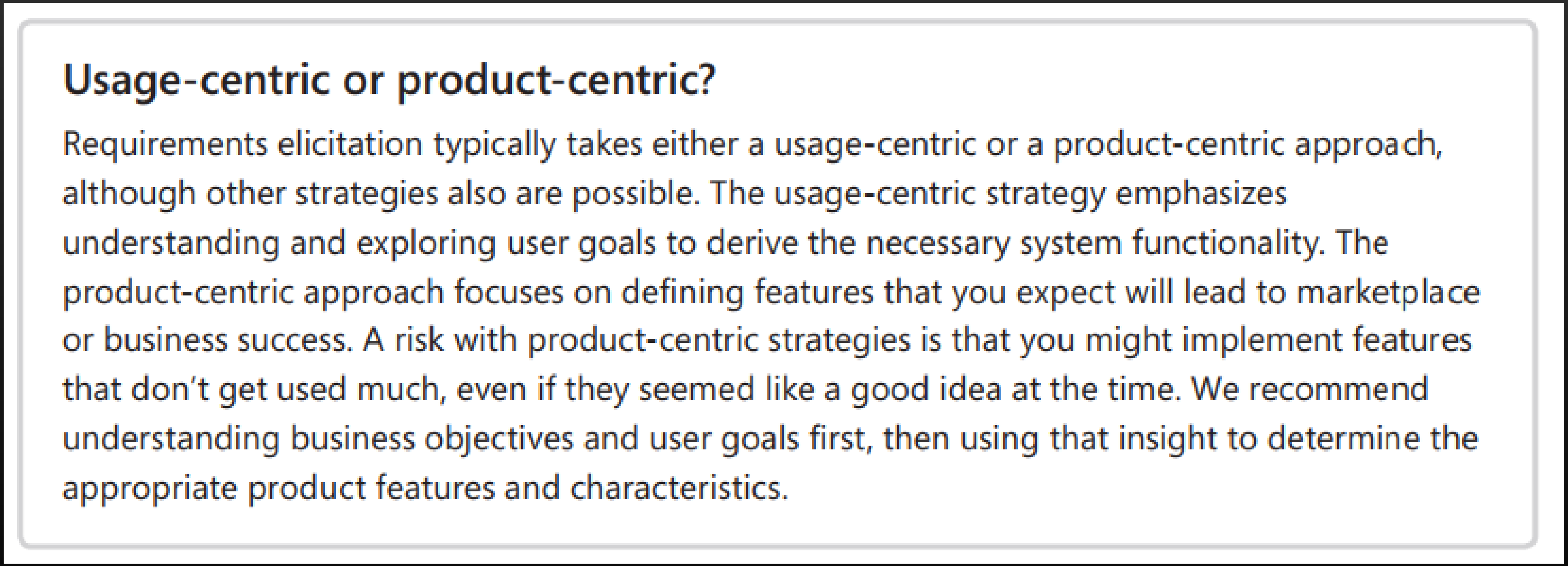
FIGURE 1-5 The boundary between requirements development and requirements management.

Requirements and/with use cases

Which main elicitation approaches exist?

What is the goal the user wants to achieve?.

What capability should the system process?



Usage-centric or product-centric?

Requirements elicitation typically takes either a usage-centric or a product-centric approach, although other strategies also are possible. The usage-centric strategy emphasizes understanding and exploring user goals to derive the necessary system functionality. The product-centric approach focuses on defining features that you expect will lead to marketplace or business success. A risk with product-centric strategies is that you might implement features that don't get used much, even if they seemed like a good idea at the time. We recommend understanding business objectives and user goals first, then using that insight to determine the appropriate product features and characteristics.

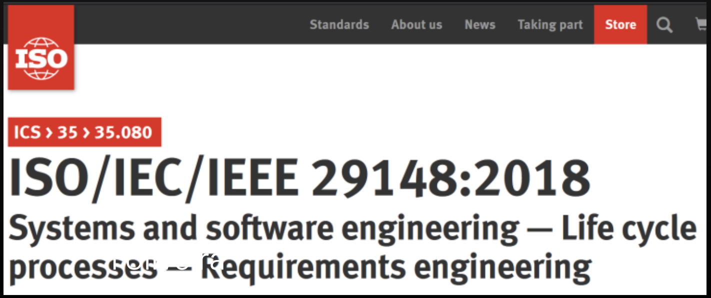
Product centric: “The system shall....”

#	Requisito
RF.1	O sistema deve permitir a um profissional criar um novo pedido de adesão , em auto-serviço, na web.
RF.2	O sistema deve enviar credenciais temporárias para os pedidos de adesão e enviá-las, por email, aos solicitantes.
RF.3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.
RNF.1	As pesquisas de cheques-dentista têm de retornar resultados em <5 segundos ou um evento de tempo expirado.
...	

What are well-formed requirements? (ISO-IEEE 29148)

A statement that:

- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- can be verified,
- is qualified by measurable conditions and bounded by constraints, and
- defines the performance of the system when used by a specific stakeholder or the corresponding capability of the system, but not a capability of the user, operator, or other stakeholder.



ICS › 35 › 35.080

ISO/IEC/IEEE 29148:2018

Systems and software engineering — Life cycle processes — Requirements engineering

Elements of style

A requirement is a statement which expresses a need and its associated constraints and conditions.

If expressed in the form of a natural language, the statement should comprise a subject, a verb and a complement. A requirement shall state the subject of the requirement (e.g., the system, the software, etc.) and what shall be done (e.g., operate at a power level, provide a field for).

E.g: The Invoice System *[Subject]*, shall display pending customer invoices *[Action]* in ascending order *[Value]* in which invoices are to be paid.

User/usage centric: capture requirements by telling stories

A) Use cases

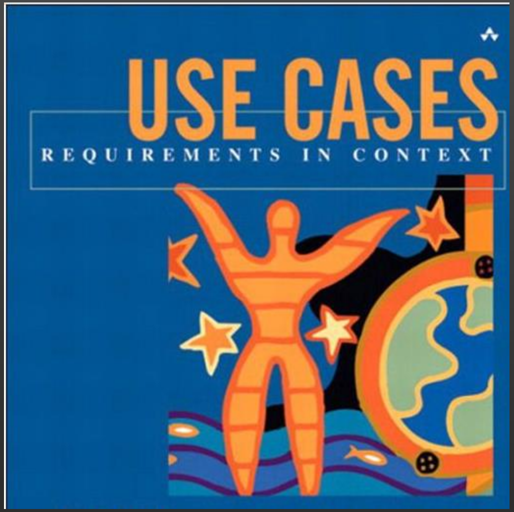
- Focus on the actor/system interaction scenario
- UML supported

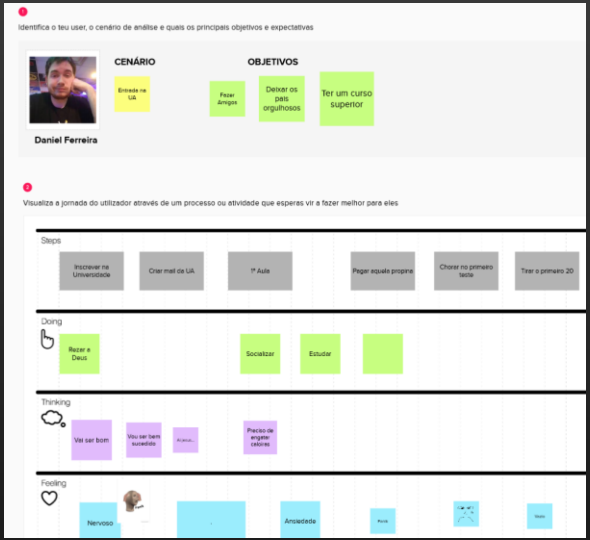
B) Customer Journey maps

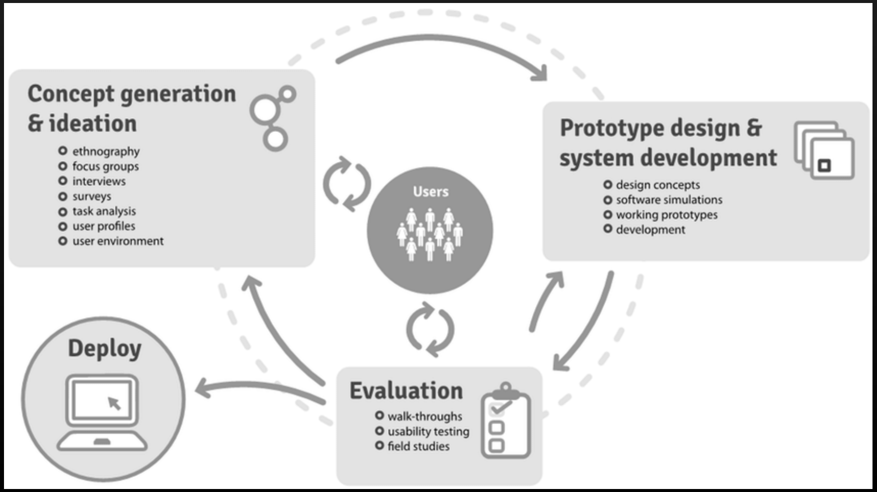
- Focus on the user comprehensive experience
- Look for pain-points & opportunities

C) User-centered design

- Depart from personas & scenarios elaboration
- Evolve through prototyping & validation cycles




Identifica o teu user, o cenário de análise e quais os principais objetivos e expectativas
Daniel Ferreira
CENÁRIO
Entrada na UA
OBJETIVOS
Fazer Amigos
Deixar os pais orgulhosos
Ter um curso superior
Visualiza a jornada do utilizador através de um processo ou atividade que esperas vir a fazer melhor para eles
Steps
Inscrever na Universidade
Criar mail da UA
1ª Aula
Pagar aquela propina
Chorar no primeiro teste
Tirar o primeiro 20
Doing
Rezar a Deus
Socializar
Estudar
Thinking
Vai ser bom
Vou ser bem sucedido
Preciso de engatar calorias
Feeling
Nervoso
Ansiedade



CaU contam histórias que mostram os requisitos funcionais em contexto

Caso Utiliz.:	**Consultar informação clínica (referenciação)**
Motivação:	O Médico Dentista (MD) acede ao sistema para consulta informação clínica inserida pelo médico assistente.
Sequência típica:	1. Inicia-se quando o Médico Dentista recebe um Utente portador de Cheque-Dentista para consulta. 2. O MD acede à opção de pesquisa na sua página de entrada. 3. O sistema apresenta o formulário de pesquisa, com as hipóteses de pesquisa por nr utente, nome, género e data de Nascimento. 4. O MD insere o número de utente do SNS e confirma a pesquisa. 5. O sistema pesquisa os cheques-dentista existentes para aquele utente e apresenta uma listagem ordenada do mais recente para o mais antigo. 6. O MD seleciona uma entrada na lista para abrir a informação de detalhe. 7. O sistema apresenta para esse CD o cabeçalho com a identificação do cheque e utente, e uma seção com a informação clínica disponível.
Sequências	...

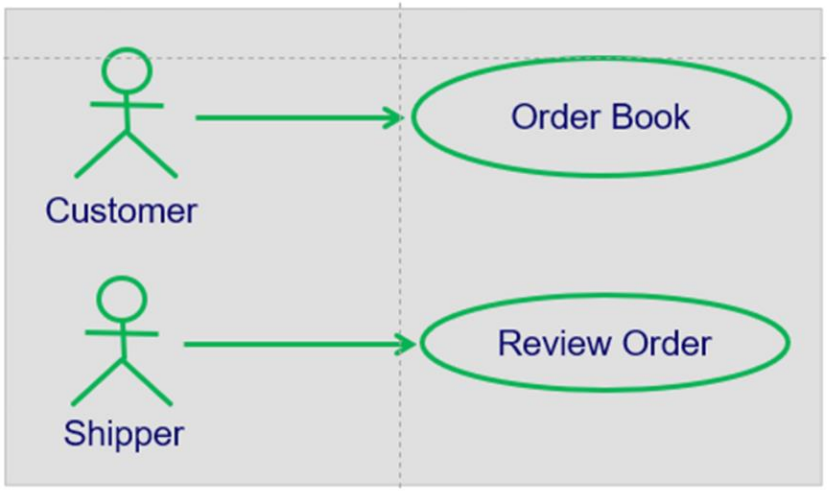
RF.3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.

OpenUP recommended practices

Use Case Driven Development

ECLIPSE PROCESS FRAMEWORK

- This practice describes how to capture requirements with a combination of use cases and system-wide requirements, and then drive development and testing from those use cases.



Guidelines to apply use cases

Conceito apresentado originalmente em:
Jacobson, I., 1993. *Object-oriented software engineering: a use case driven approach*. Pearson Education.

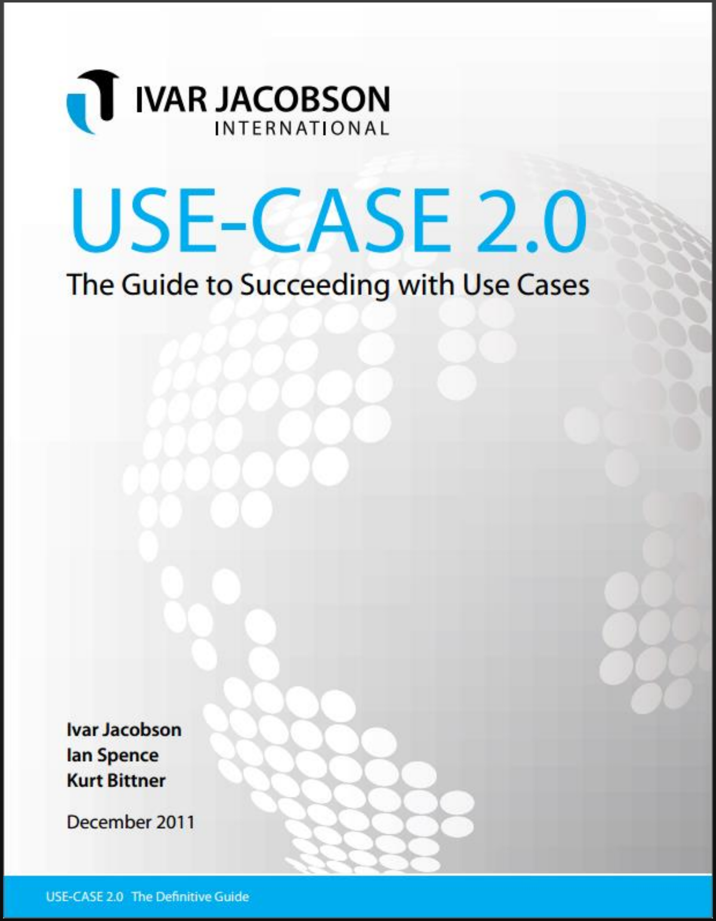
You don't need to apply OpenUP to develop use cases

The technique works for other types of methods

Jabson offers updated views

Adapt an "old" technique to modern agile approaches

The white paper is well worth reading!



IVAR JACOBSON INTERNATIONAL

USE-CASE 2.0

The Guide to Succeeding with Use Cases

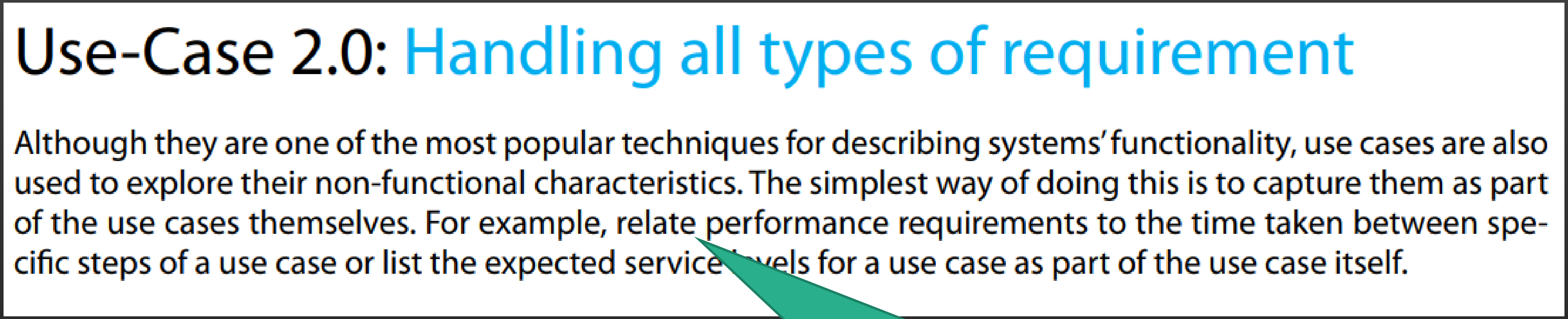
Ivar Jacobson
Ian Spence
Kurt Bittner

December 2011

USE-CASE 2.0 The Definitive Guide

https://www.ivarjacobson.com/publications/white-papers/use-case-ebook

Can use cases address non-functional requirements?



Use-Case 2.0: Handling all types of requirement

Although they are one of the most popular techniques for describing systems' functionality, use cases are also used to explore their non-functional characteristics. The simplest way of doing this is to capture them as part of the use cases themselves. For example, relate performance requirements to the time taken between specific steps of a use case or list the expected service levels for a use case as part of the use case itself.

Relacionar o requisito não funcional com o contetxo em que ele é preciso.

Take note of NFR in the use case description

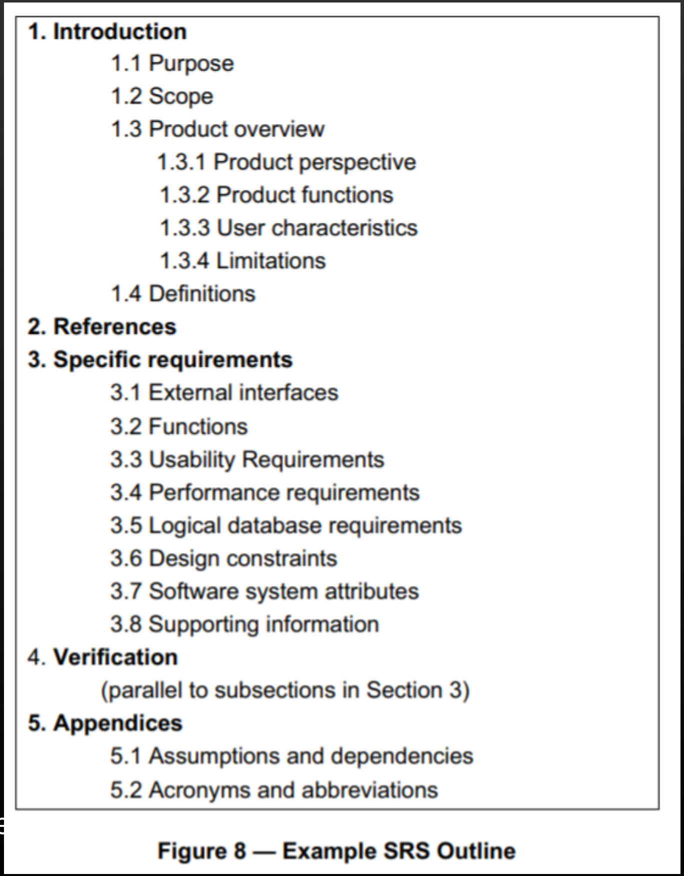
→ Special requirements section in this model

Documenting requirements

Documenting requirements: SRS report

A special report to document the requirements specification

- "System Proposal" [Dennis'15]
- "Software requirements specification" is an industry-standard term (ISO/IEC/IEEE 2011).



1. **Introduction**
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Product overview
        - 1.3.1 Product perspective
        - 1.3.2 Product functions
        - 1.3.3 User characteristics
        - 1.3.4 Limitations
    - 1.4 Definitions
2. **References**
3. **Specific requirements**
    - 3.1 External interfaces
    - 3.2 Functions
    - 3.3 Usability Requirements
    - 3.4 Performance requirements
    - 3.5 Logical database requirements
    - 3.6 Design constraints
    - 3.7 Software system attributes
    - 3.8 Supporting information
4. **Verification**
    (parallel to subsections in Section 3)
5. **Appendices**
    - 5.1 Assumptions and dependencies
    - 5.2 Acronyms and abbreviations

Figure 8 — Example SRS Outline

Who relies on the SRS?

Customers, the marketing department, and sales staff need to know what product they can expect to be delivered.

Project managers base their estimates of schedule, effort, and resources on the requirements.

Software development teams need to know what to build.

Testers use it to develop requirements-based tests, test plans, and test procedures.

Documentation writers base user manuals and help screens on the SRS and the user interface design.

Training personnel use the SRS and user documentation to develop educational materials.

Legal staff ensures that the requirements comply with applicable laws and regulations.

Subcontractors base their work on—and can be legally held to—the specified requirements.

Software requirements reports

ISO/IEC/IEEE 29148:2011: Systems and software engineering -- Life cycle processes --Requirements engineering

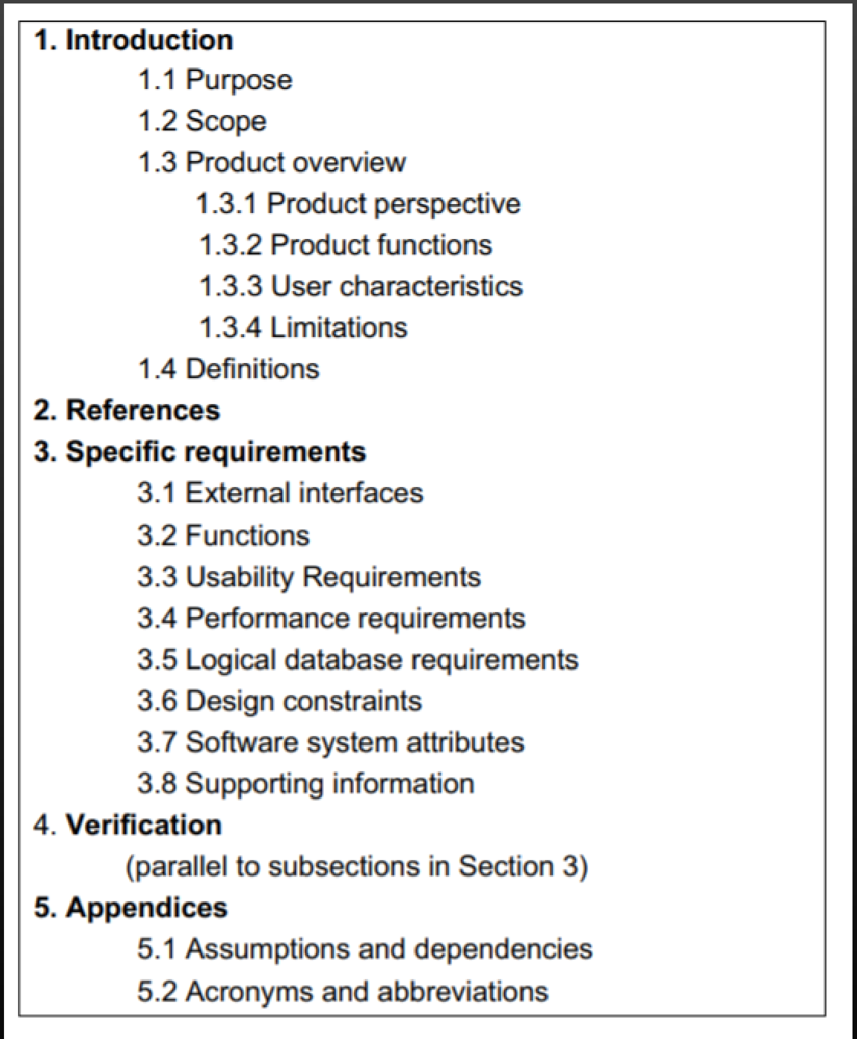
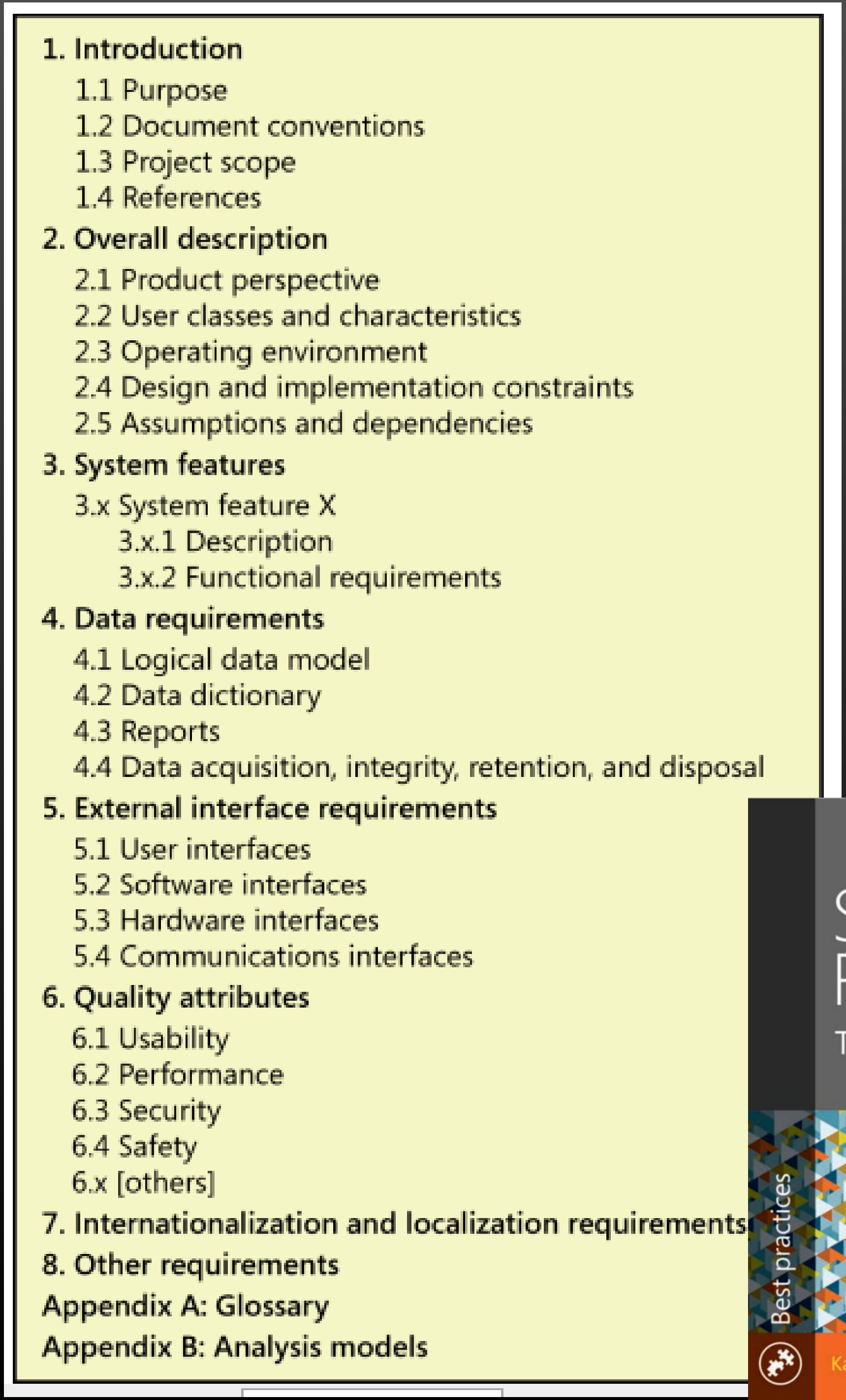

1. **Introduction**
    1.1 Purpose
    1.2 Scope
    1.3 Product overview
        1.3.1 Product perspective
        1.3.2 Product functions
        1.3.3 User characteristics
        1.3.4 Limitations
    1.4 Definitions
2. **References**
3. **Specific requirements**
    3.1 External interfaces
    3.2 Functions
    3.3 Usability Requirements
    3.4 Performance requirements
    3.5 Logical database requirements
    3.6 Design constraints
    3.7 Software system attributes
    3.8 Supporting information
4. **Verification**
    (parallel to subsections in Section 3)
5. **Appendices**
    5.1 Assumptions and dependencies
    5.2 Acronyms and abbreviations

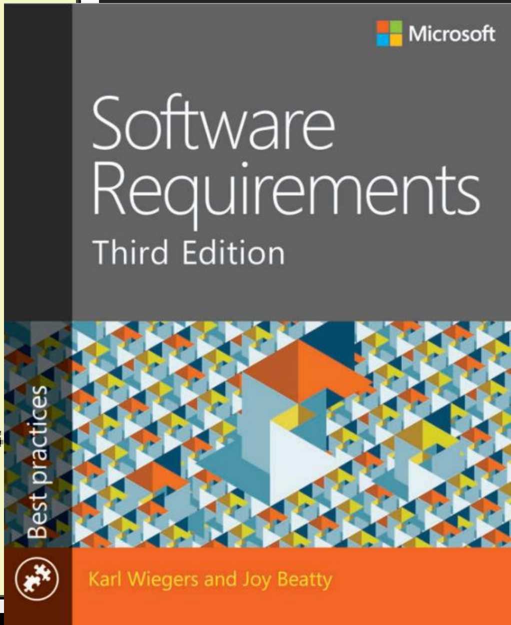
Figure 8 — Example SRS Outline

Wieger's proposal


1. Introduction
   - 1.1 Purpose
   - 1.2 Document conventions
   - 1.3 Project scope
   - 1.4 References
2. Overall description
   - 2.1 Product perspective
   - 2.2 User classes and characteristics
   - 2.3 Operating environment
   - 2.4 Design and implementation constraints
   - 2.5 Assumptions and dependencies
3. System features
   - 3.x System feature X
     - 3.x.1 Description
     - 3.x.2 Functional requirements
4. Data requirements
   - 4.1 Logical data model
   - 4.2 Data dictionary
   - 4.3 Reports
   - 4.4 Data acquisition, integrity, retention, and disposal
5. External interface requirements
   - 5.1 User interfaces
   - 5.2 Software interfaces
   - 5.3 Hardware interfaces
   - 5.4 Communications interfaces
6. Quality attributes
   - 6.1 Usability
   - 6.2 Performance
   - 6.3 Security
   - 6.4 Safety
   - 6.x [others]
7. Internationalization and localization requirements
8. Other requirements

Appendix A: Glossary

Appendix B: Analysis models



Requirements-related artifacts in (Open)UP?

Use-Case Model

A set of typical scenarios of using a system. There are primarily for **functional** (behavioral) requirements.

Supplementary Specification (System-wide requirements)

Basically, everything not in the use cases. This artifact is primarily for all **non-functional** requirements, such as performance or licensing. It is also the place to record functional features not expressed (or expressible) as use cases (for example, a report generation).

Glossary

In its simplest form, the Glossary defines noteworthy terms. It also encompasses the concept of the data dictionary, which records **requirements related to data**, such as validation rules, acceptable values, and so forth. The Glossary can detail any element: an attribute of an object, a parameter of an operation call, a report layout, and so forth.

Business Rules

Business rules (also called Domain Rules) typically describe **requirements or policies that transcend one software project** they are required in the domain or business, and many applications may need to conform to them. An good example is government tax laws.

Examples of SRS reports

→ SRS example fom Wiegers

→ System Proposal from Dennis (chap. 3)

→ OpenUP: use cases + system-wide requirements

AMS (and OpenUP) approach to build the SRS?...

Evolutionary requirements

Evolutionary requirements

Plan-driven, “waterfall” approach

Attempting to fully define and stabilize the requirements in the first phase of a project before programming

Requirements defined up-front for the entire systems

Agile, evolutionary approach

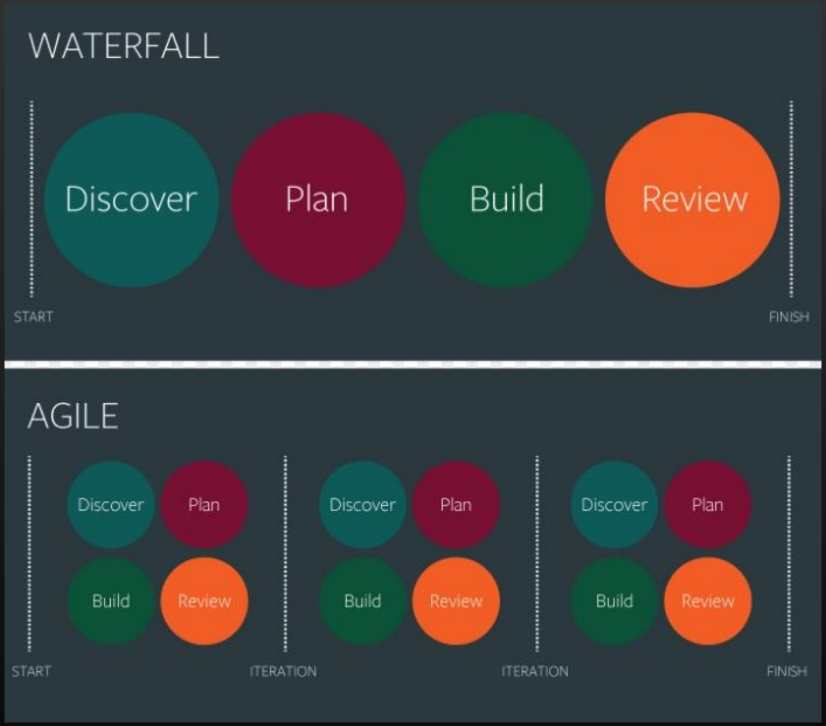
Accept and prepare for the inevitably changing and unclear stakeholder's wishes/needs

Some sort of systematic approach to finding, documenting, organizing, and tracking the *changing* requirements of a system

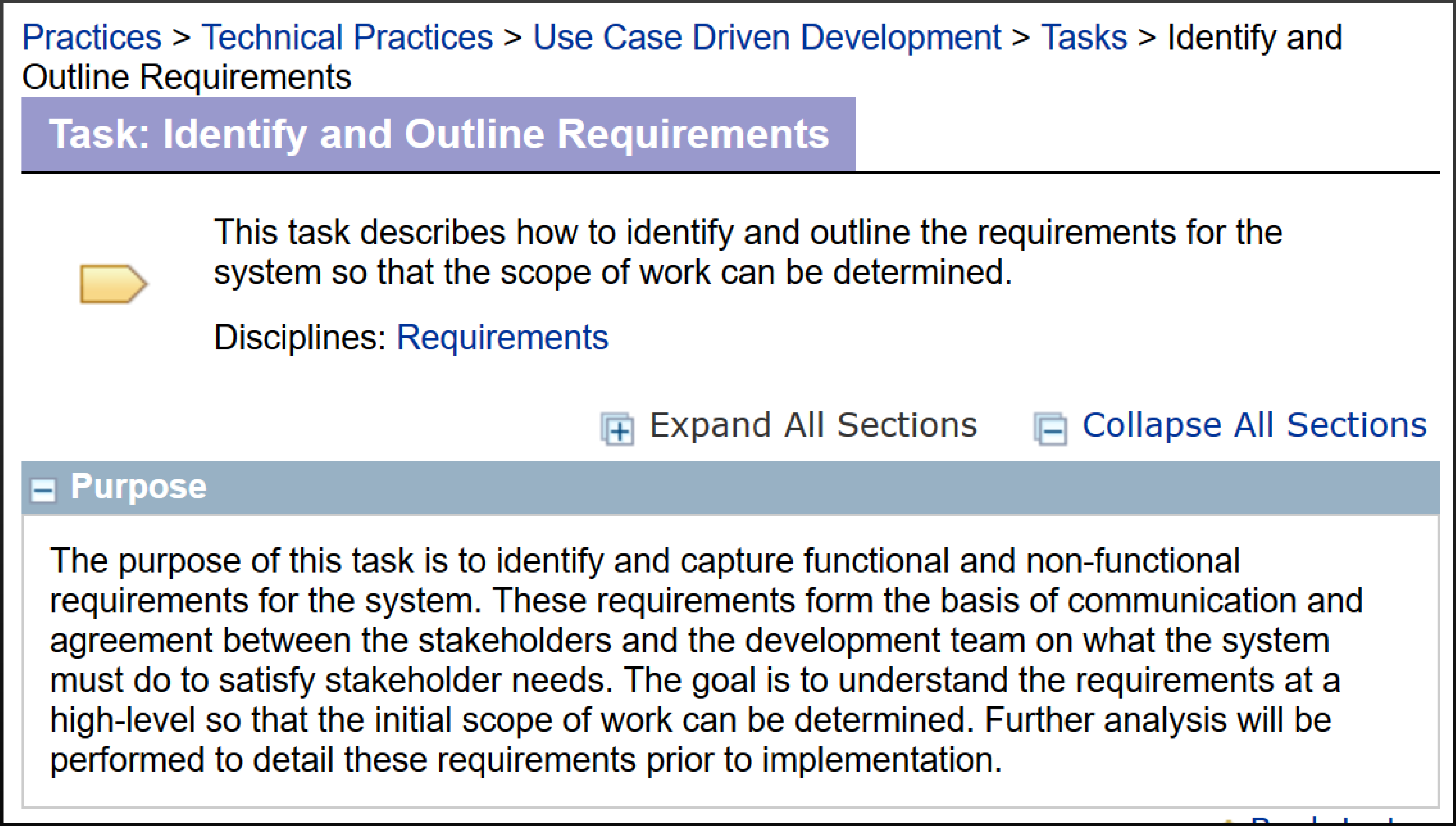
High-level, soon; detailed, closer to the build process

Wiegers:

You don't have to write the SRS for the entire product before beginning development, but you should capture the requirements for each increment before building that increment.



Does OpenUP propose evolutionary requirements?



Practices > Technical Practices > Use Case Driven Development > Tasks > Identify and Outline Requirements

Task: Identify and Outline Requirements

This task describes how to identify and outline the requirements for the system so that the scope of work can be determined.

Disciplines: Requirements

Purpose

The purpose of this task is to identify and capture functional and non-functional requirements for the system. These requirements form the basis of communication and agreement between the stakeholders and the development team on what the system must do to satisfy stakeholder needs. The goal is to understand the requirements at a high-level so that the initial scope of work can be determined. Further analysis will be performed to detail these requirements prior to implementation.

Requirements engineering practices (Wiegers)

Requirements development activities

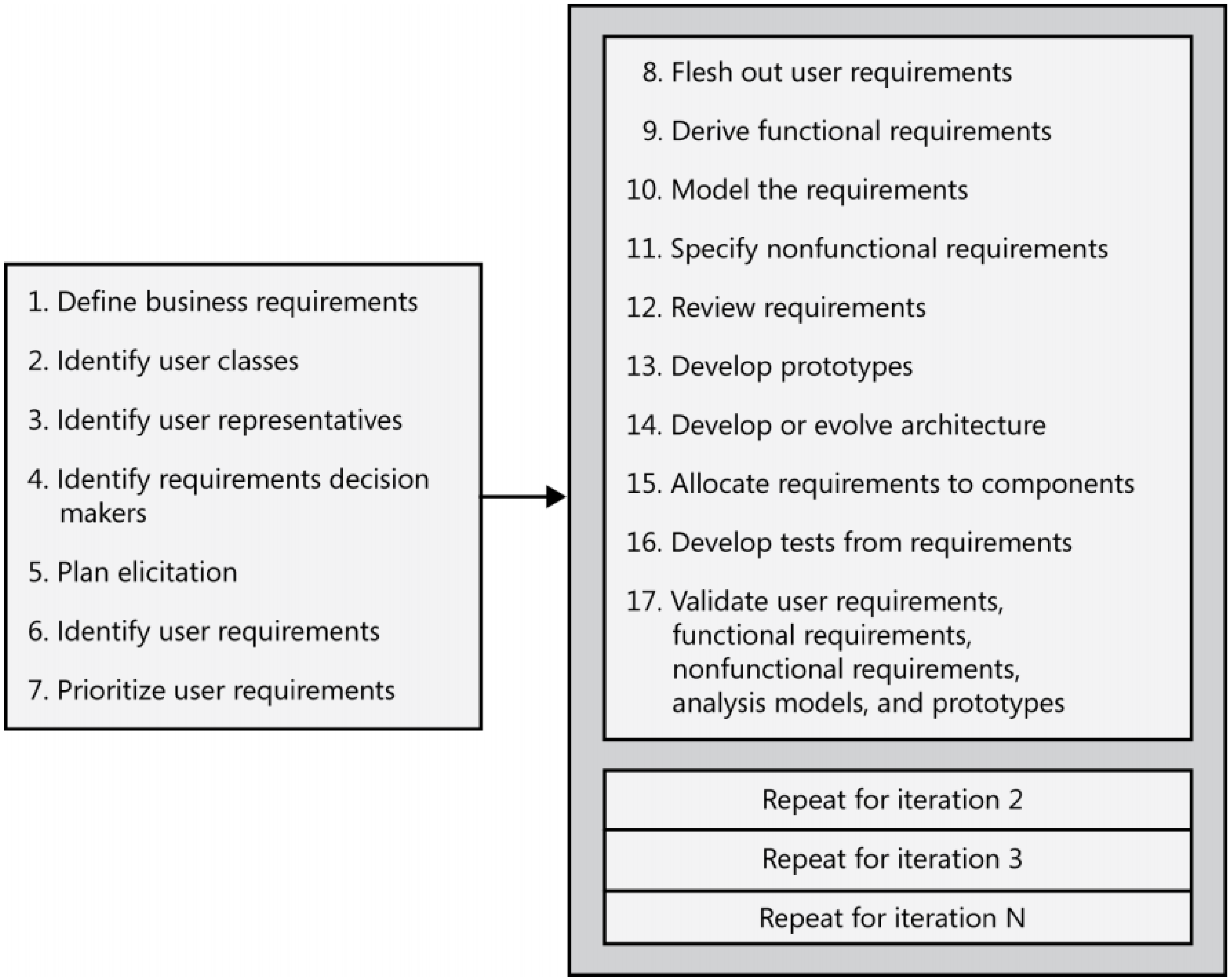

1. Define business requirements
2. Identify user classes
3. Identify user representatives
4. Identify requirements decision makers
5. Plan elicitation
6. Identify user requirements
7. Prioritize user requirements
8. Flesh out user requirements
9. Derive functional requirements
10. Model the requirements
11. Specify nonfunctional requirements
12. Review requirements
13. Develop prototypes
14. Develop or evolve architecture
15. Allocate requirements to components
16. Develop tests from requirements
17. Validate user requirements, functional requirements, nonfunctional requirements, analysis models, and prototypes
Repeat for iteration 2
Repeat for iteration 3
Repeat for iteration N

FIGURE 3-2 A representative requirements development process.

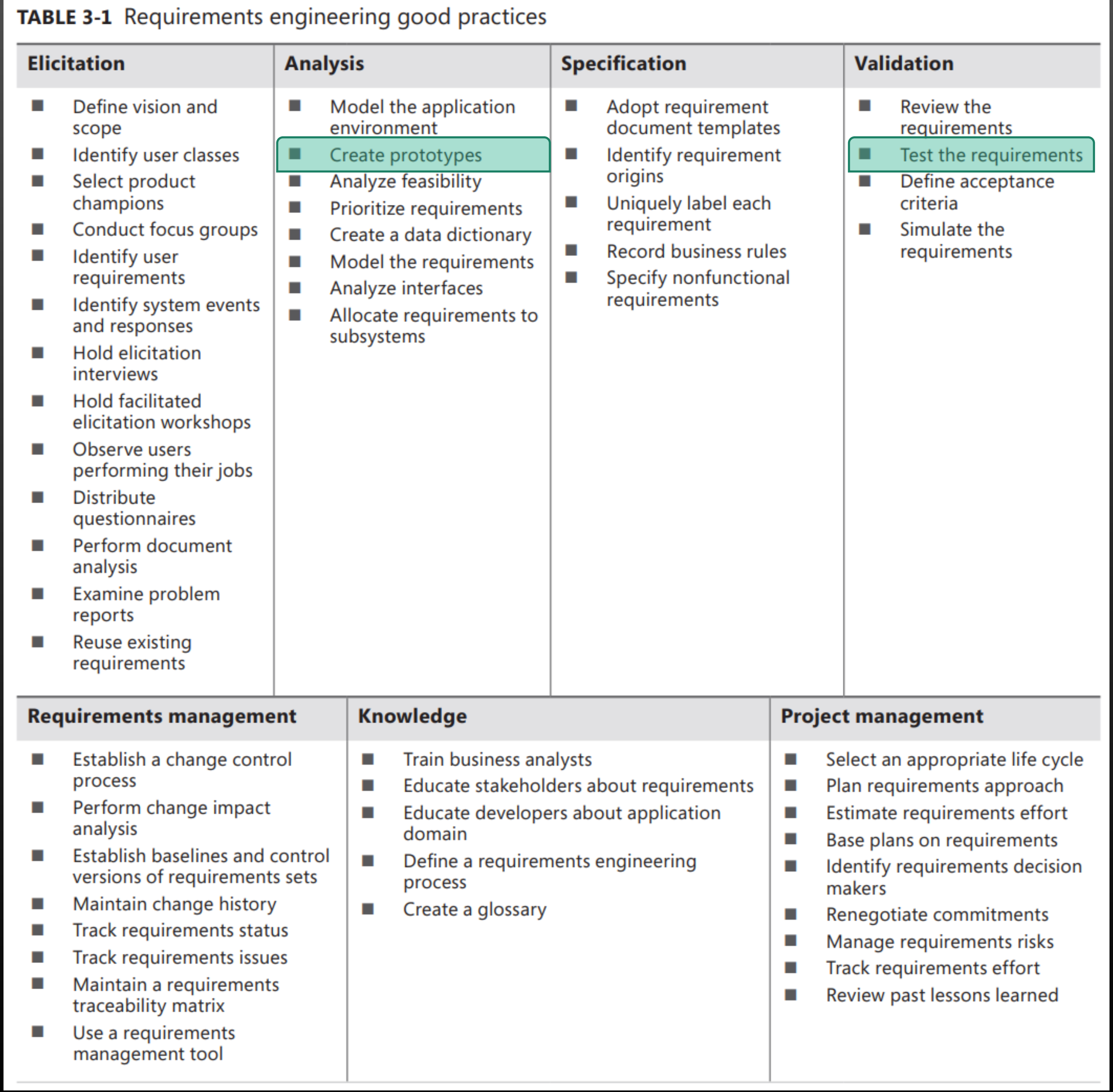


TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
■ Define vision and scope ■ Identify user classes ■ Select product champions ■ Conduct focus groups ■ Identify user requirements ■ Identify system events and responses ■ Hold elicitation interviews ■ Hold facilitated elicitation workshops ■ Observe users performing their jobs ■ Distribute questionnaires ■ Perform document analysis ■ Examine problem reports ■ Reuse existing requirements	■ Model the application environment ■ Create prototypes ■ Analyze feasibility ■ Prioritize requirements ■ Create a data dictionary ■ Model the requirements ■ Analyze interfaces ■ Allocate requirements to subsystems	■ Adopt requirement document templates ■ Identify requirement origins ■ Uniquely label each requirement ■ Record business rules ■ Specify nonfunctional requirements	■ Review the requirements ■ Test the requirements ■ Define acceptance criteria ■ Simulate the requirements

Requirements management	Knowledge	Project management
■ Establish a change control process ■ Perform change impact analysis ■ Establish baselines and control versions of requirements sets ■ Maintain change history ■ Track requirements status ■ Track requirements issues ■ Maintain a requirements traceability matrix ■ Use a requirements management tool	■ Train business analysts ■ Educate stakeholders about requirements ■ Educate developers about application domain ■ Define a requirements engineering process ■ Create a glossary	■ Select an appropriate life cycle ■ Plan requirements approach ■ Estimate requirements effort ■ Base plans on requirements ■ Identify requirements decision makers ■ Renegotiate commitments ■ Manage requirements risks ■ Track requirements effort ■ Review past lessons learned

Prototyping as a requirements validation practice

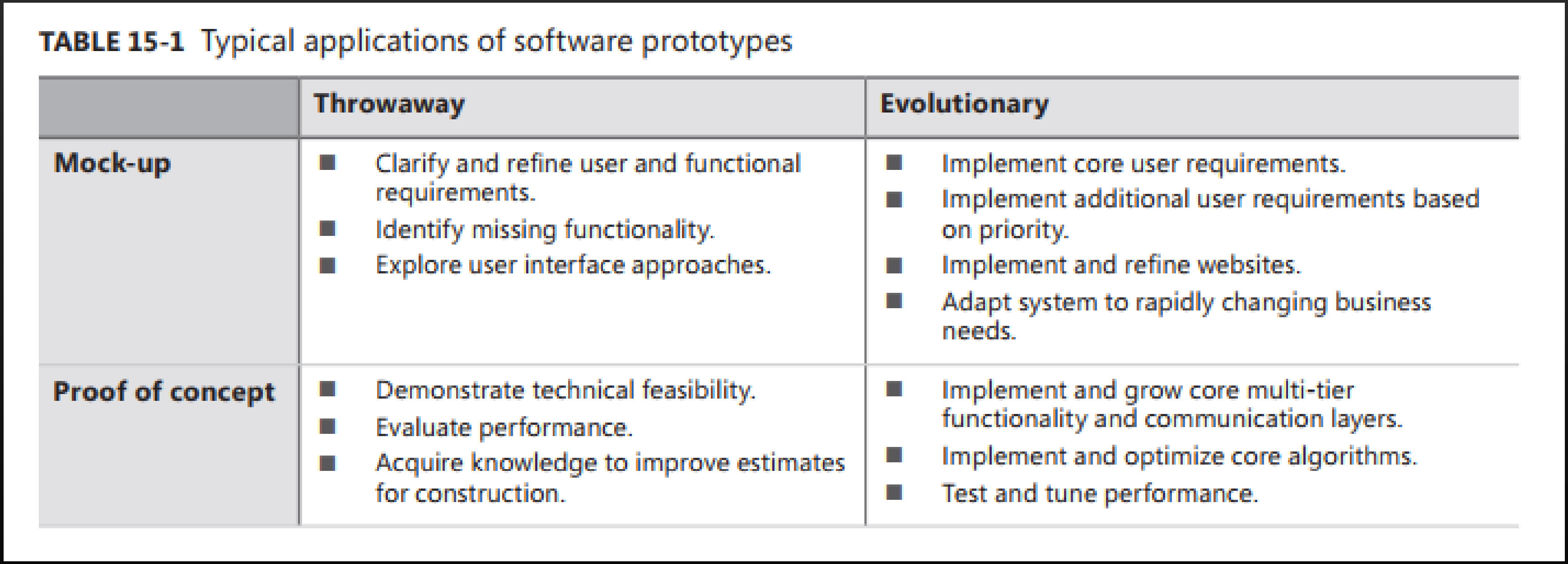


TABLE 15-1 Typical applications of software prototypes

	Throwaway	Evolutionary
Mock-up	■ Clarify and refine user and functional requirements. ■ Identify missing functionality. ■ Explore user interface approaches.	■ Implement core user requirements. ■ Implement additional user requirements based on priority. ■ Implement and refine websites. ■ Adapt system to rapidly changing business needs.
Proof of concept	■ Demonstrate technical feasibility. ■ Evaluate performance. ■ Acquire knowledge to improve estimates for construction.	■ Implement and grow core multi-tier functionality and communication layers. ■ Implement and optimize core algorithms. ■ Test and tune performance.

Evolving requirements: life after the baseline

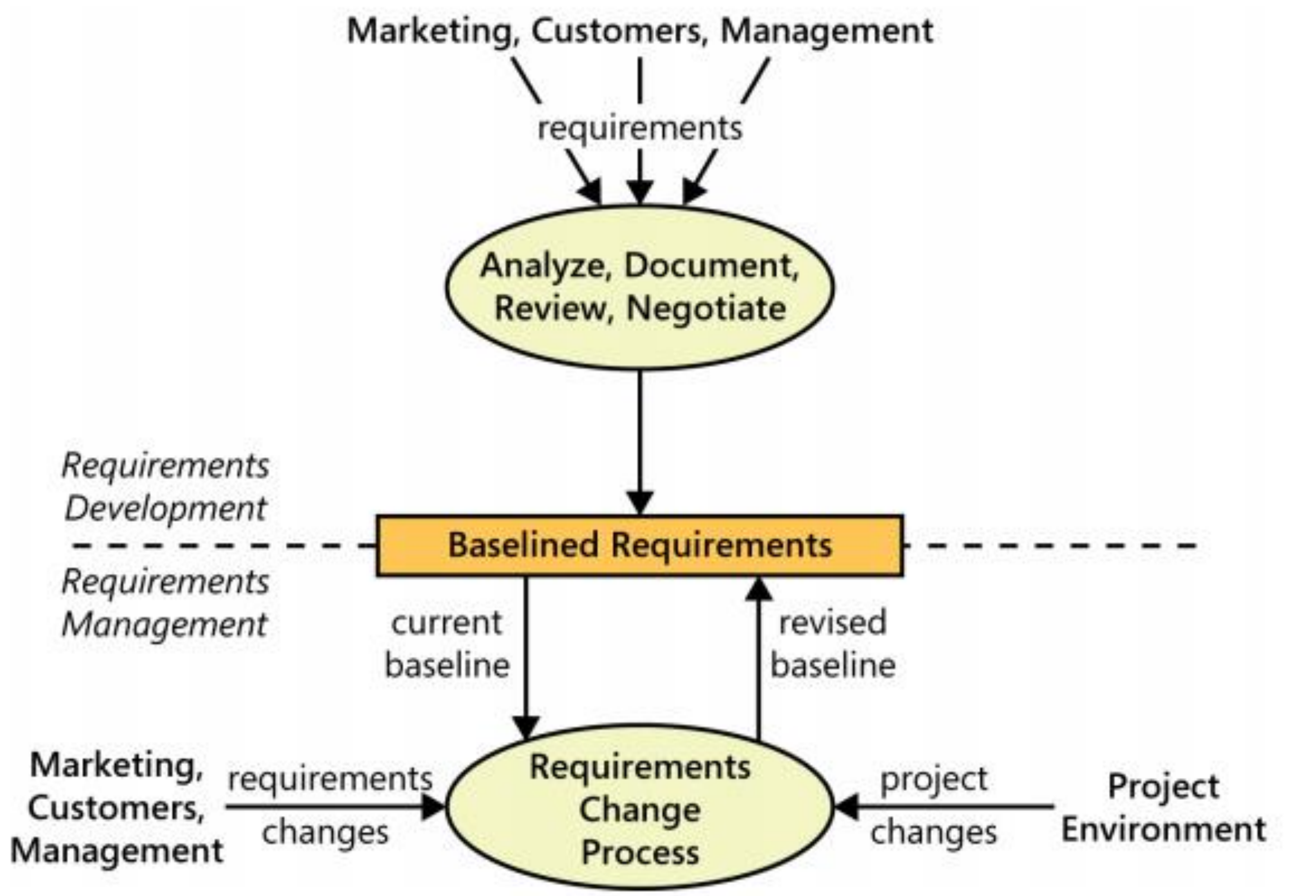


FIGURE 1-5 The boundary between requirements development and requirements management.

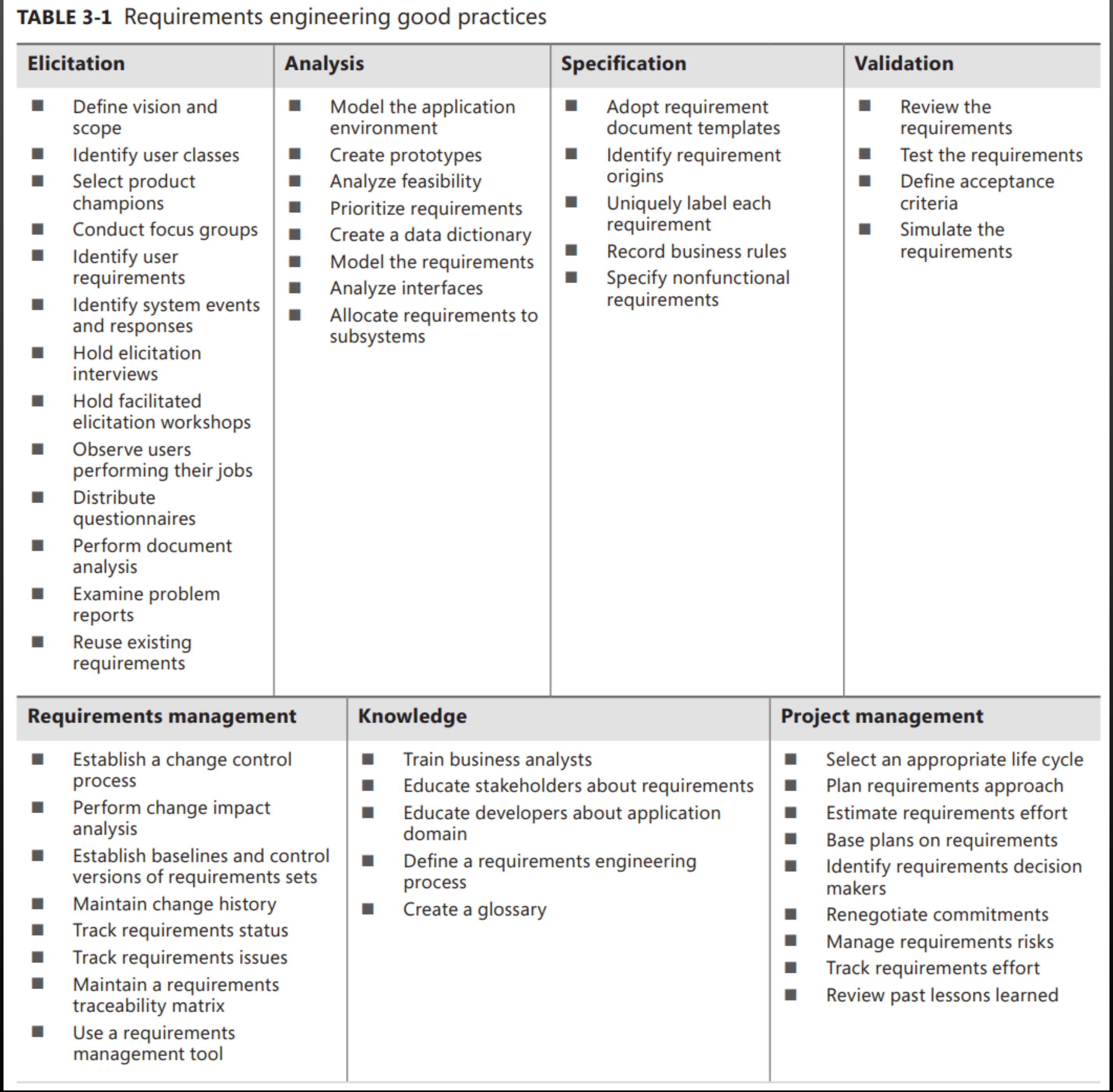


TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
■ Define vision and scope	■ Model the application environment	■ Adopt requirement document templates	■ Review the requirements
■ Identify user classes	■ Create prototypes	■ Identify requirement origins	■ Test the requirements
■ Select product champions	■ Analyze feasibility	■ Uniquely label each requirement	■ Define acceptance criteria
■ Conduct focus groups	■ Prioritize requirements	■ Record business rules	■ Simulate the requirements
■ Identify user requirements	■ Create a data dictionary	■ Specify nonfunctional requirements	
■ Identify system events and responses	■ Model the requirements		
■ Hold elicitation interviews	■ Analyze interfaces		
■ Hold facilitated elicitation workshops	■ Allocate requirements to subsystems		
■ Observe users performing their jobs			
■ Distribute questionnaires			
■ Perform document analysis			
■ Examine problem reports			
■ Reuse existing requirements			

Requirements management	Knowledge	Project management
■ Establish a change control process	■ Train business analysts	■ Select an appropriate life cycle
■ Perform change impact analysis	■ Educate stakeholders about requirements	■ Plan requirements approach
■ Establish baselines and control versions of requirements sets	■ Educate developers about application domain	■ Estimate requirements effort
■ Maintain change history	■ Define a requirements engineering process	■ Base plans on requirements
■ Track requirements status	■ Create a glossary	■ Identify requirements decision makers
■ Track requirements issues		■ Renegotiate commitments
■ Maintain a requirements traceability matrix		■ Manage requirements risks
■ Use a requirements management tool		■ Track requirements effort
		■ Review past lessons learned

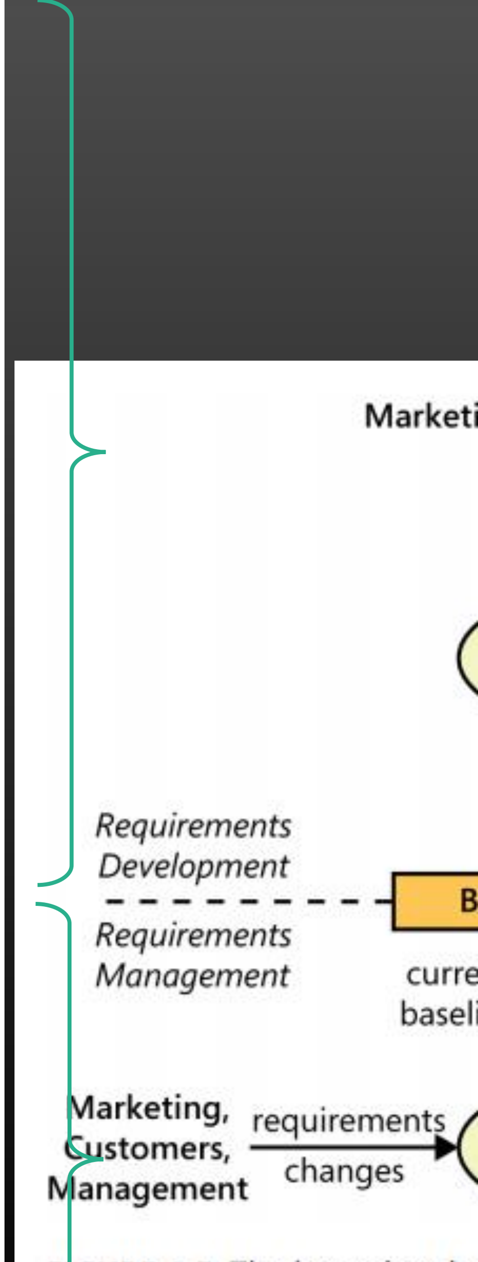


FIGURE 1-5 The boundary be

Types of information collected in requirements development

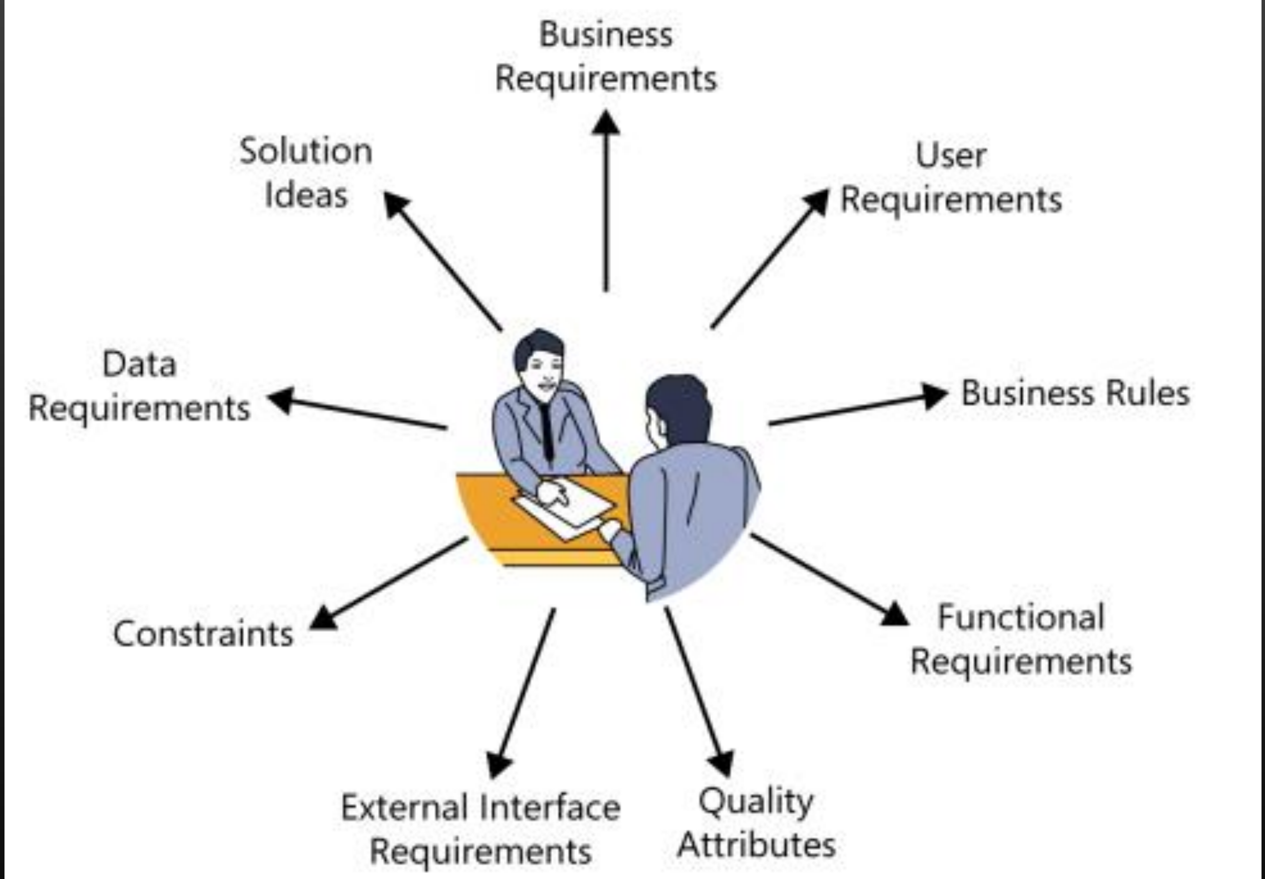


FIGURE 7-7 Classifying customer input.

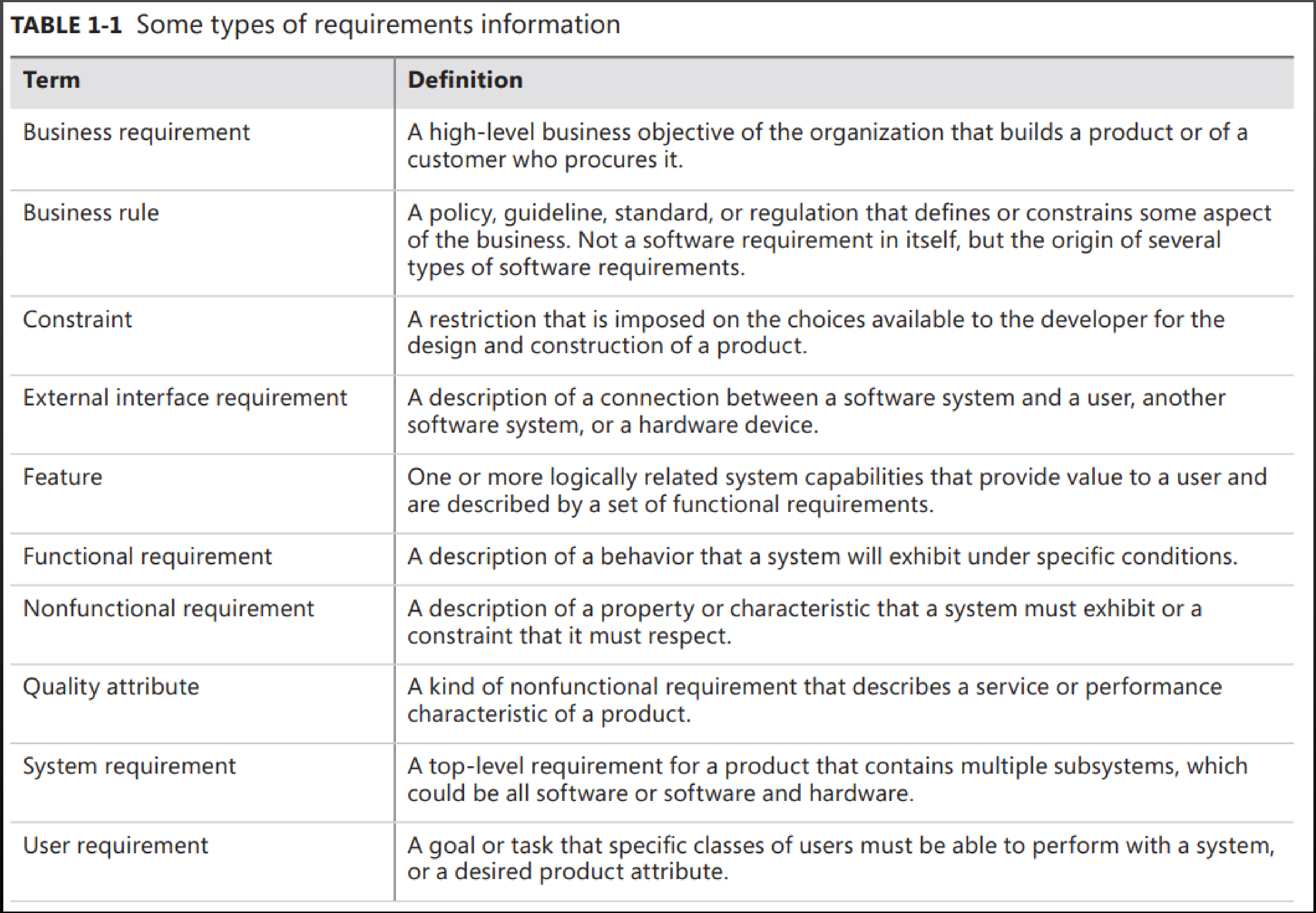


TABLE 1-1 Some types of requirements information

Term	Definition
Business requirement	A high-level business objective of the organization that builds a product or of a customer who procures it.
Business rule	A policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself, but the origin of several types of software requirements.
Constraint	A restriction that is imposed on the choices available to the developer for the design and construction of a product.
External interface requirement	A description of a connection between a software system and a user, another software system, or a hardware device.
Feature	One or more logically related system capabilities that provide value to a user and are described by a set of functional requirements.
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Nonfunctional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect.
Quality attribute	A kind of nonfunctional requirement that describes a service or performance characteristic of a product.
System requirement	A top-level requirement for a product that contains multiple subsystems, which could be all software or software and hardware.
User requirement	A goal or task that specific classes of users must be able to perform with a system, or a desired product attribute.

Types of information connected to requirements 1/2

Business rule

Business rule → a policy, guideline, standard, or regulation that defines or constrains some aspect of the business. Not a software requirement in itself (because they have an existence beyond the boundaries of any specific software application), but the origin of several types of software requirements.

Phrases such as "Must comply with . . . ," "If <some condition is true>, then <something happens>," or "Must be calculated according to . . . " suggest that the user is describing a business rule. E.g.:

→ "A new client must pay 30 percent of the estimated consulting fee and travel expenses in advance."

→ "Time-off approvals must comply with the company's HR vacation policy."

External interface requirements

Requirements in this category describe the connections between your system and the rest of the universe.

Phrases such as "Must read signals from . . . ," "Must send messages to . . . ," "Must be able to read files in <format>," and "User interface elements must conform to <a standard>" indicate that the customer is describing an external interface requirement. Following are some examples:

→ "The manufacturing execution system must control the wafer sorter."

→ "The mobile app should send the check image to the bank after I photograph the check I'm depositing."

Types of information connected to requirements 2/2

Functional requirements

Functional requirements describe the observable behaviors the system will exhibit under certain conditions and the actions the system will let users take.

Here are some examples of functional requirements as you might hear them from users:

→ "If the pressure exceeds 40.0 psi, the high-pressure warning light should come on."

→ "The user must be able to sort the project list in forward and reverse alphabetical order."

These statements illustrate how users typically present functional requirements, but they don't represent good ways to write functional requirements. **The BA will need to craft these into more precise specifications**.

Quality attributes

Statements that describe how well the system does something are quality attributes. Listen for words that describe **desirable system characteristics**: fast, easy, user-friendly, reliable, secure.

You'll need to work with the users to understand just what they mean by these ambiguous and subjective terms so that you can write clear, variable quality goals.

The following examples suggest what quality attributes might sound like when described by users:

→ "The mobile software must respond quickly to touch commands."

→ "The shopping cart mechanism has to be simple to use so my new customers don't abandon the purchase.

Relationship between types of information

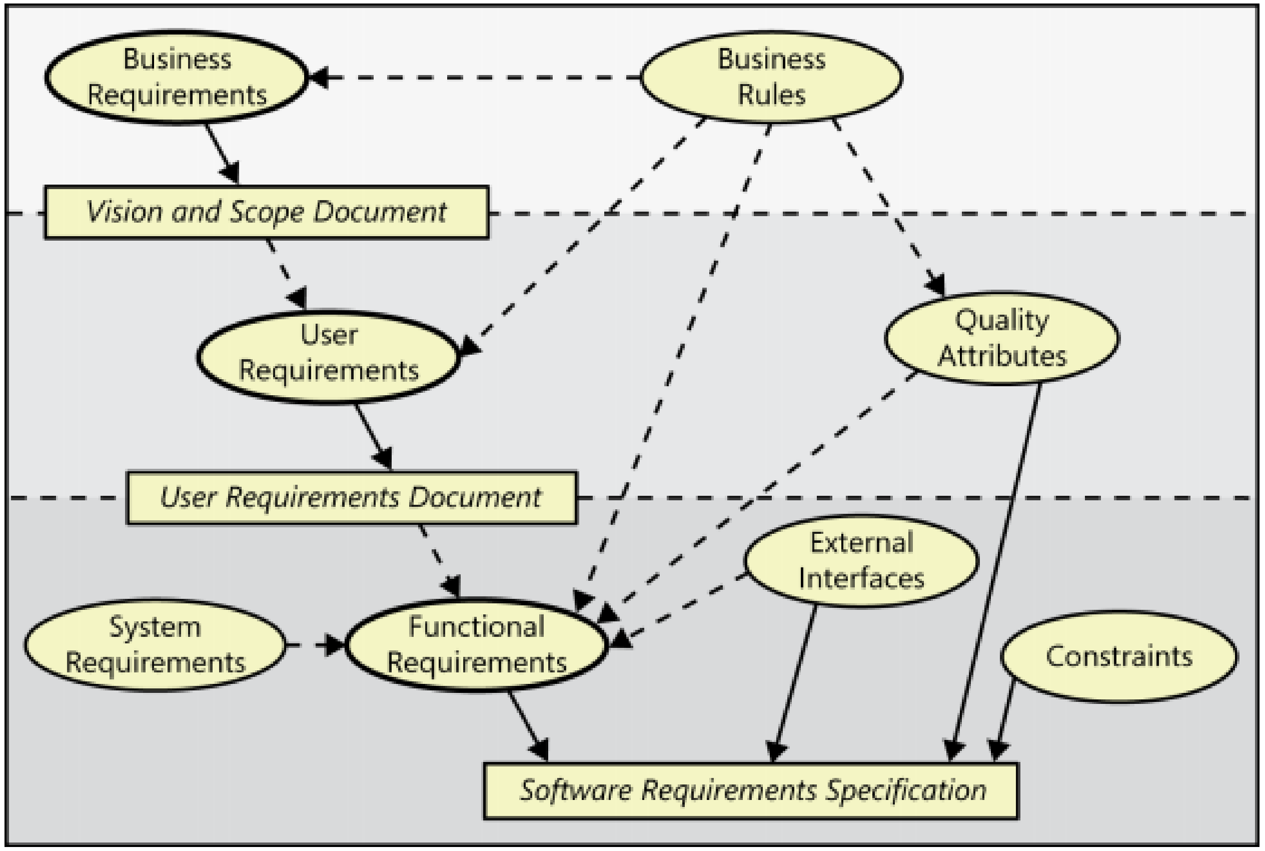


FIGURE 1-1 Relationships among several types of requirements information. Solid arrows mean "are stored in"; dotted arrows mean "are the origin of" or "influence."

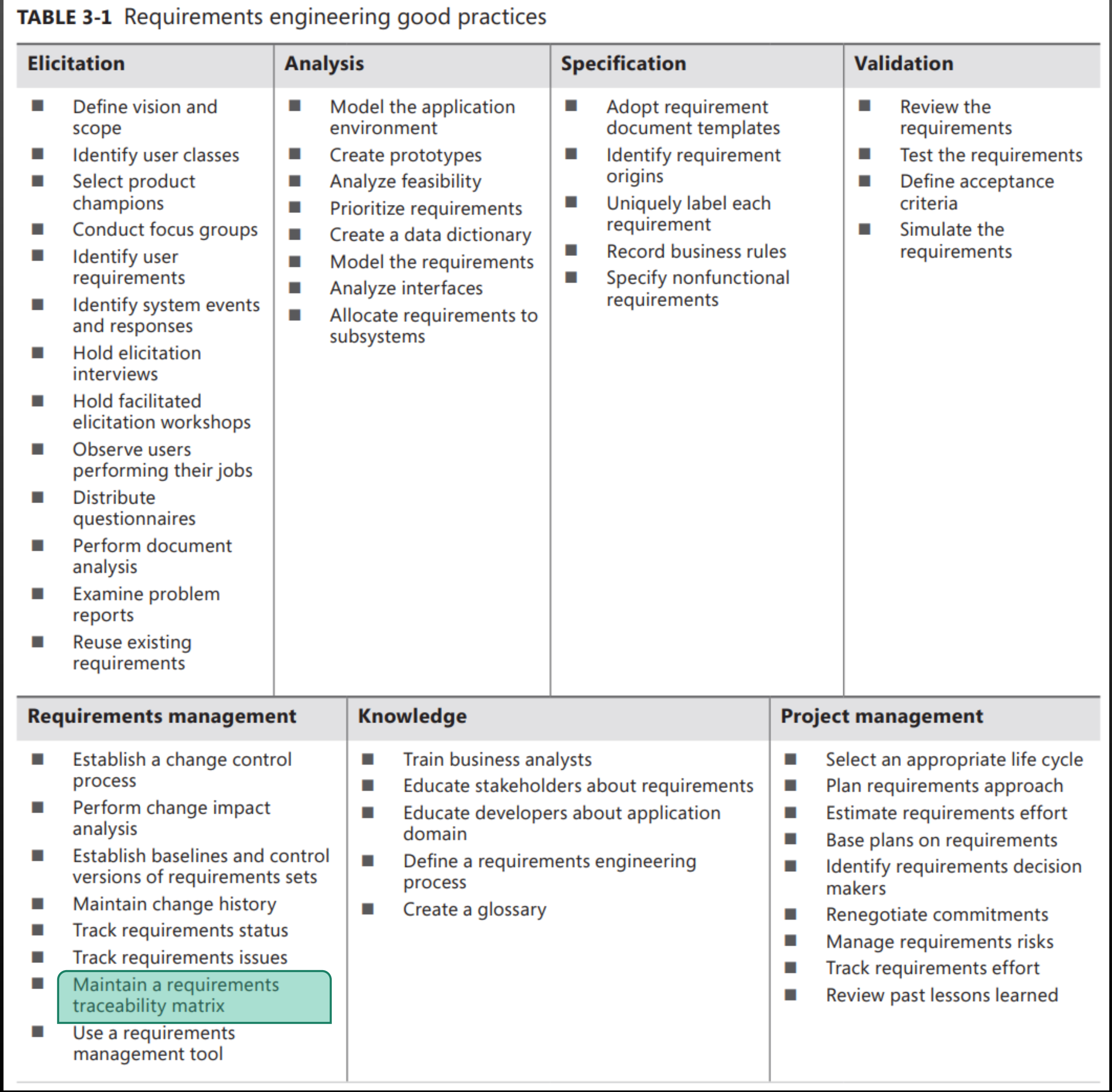


TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
■ Define vision and scope	■ Model the application environment	■ Adopt requirement document templates	■ Review the requirements
■ Identify user classes	■ Create prototypes	■ Identify requirement origins	■ Test the requirements
■ Select product champions	■ Analyze feasibility	■ Uniquely label each requirement	■ Define acceptance criteria
■ Conduct focus groups	■ Prioritize requirements	■ Record business rules	■ Simulate the requirements
■ Identify user requirements	■ Create a data dictionary	■ Specify nonfunctional requirements	
■ Identify system events and responses	■ Model the requirements		
■ Hold elicitation interviews	■ Analyze interfaces		
■ Hold facilitated elicitation workshops	■ Allocate requirements to subsystems		
■ Observe users performing their jobs			
■ Distribute questionnaires			
■ Perform document analysis			
■ Examine problem reports			
■ Reuse existing requirements			

Requirements management	Knowledge	Project management
■ Establish a change control process	■ Train business analysts	■ Select an appropriate life cycle
■ Perform change impact analysis	■ Educate stakeholders about requirements	■ Plan requirements approach
■ Establish baselines and control versions of requirements sets	■ Educate developers about application domain	■ Estimate requirements effort
■ Maintain change history	■ Define a requirements engineering process	■ Base plans on requirements
■ Track requirements status	■ Create a glossary	■ Identify requirements decision makers
■ Track requirements issues		■ Renegotiate commitments
■ Maintain a requirements traceability matrix		■ Manage requirements risks
■ Use a requirements management tool		■ Track requirements effort
		■ Review past lessons learned

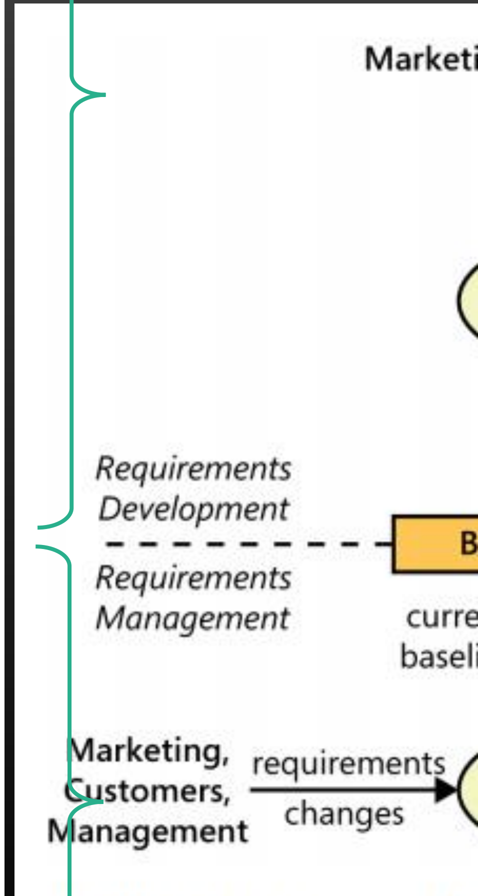


FIGURE 1-5 The boundary be

What is a requirements traceability matrix?

The most common way to represent the links between requirements and other system elements is in a requirements traceability matrix, also called a requirements trace matrix or a traceability table.

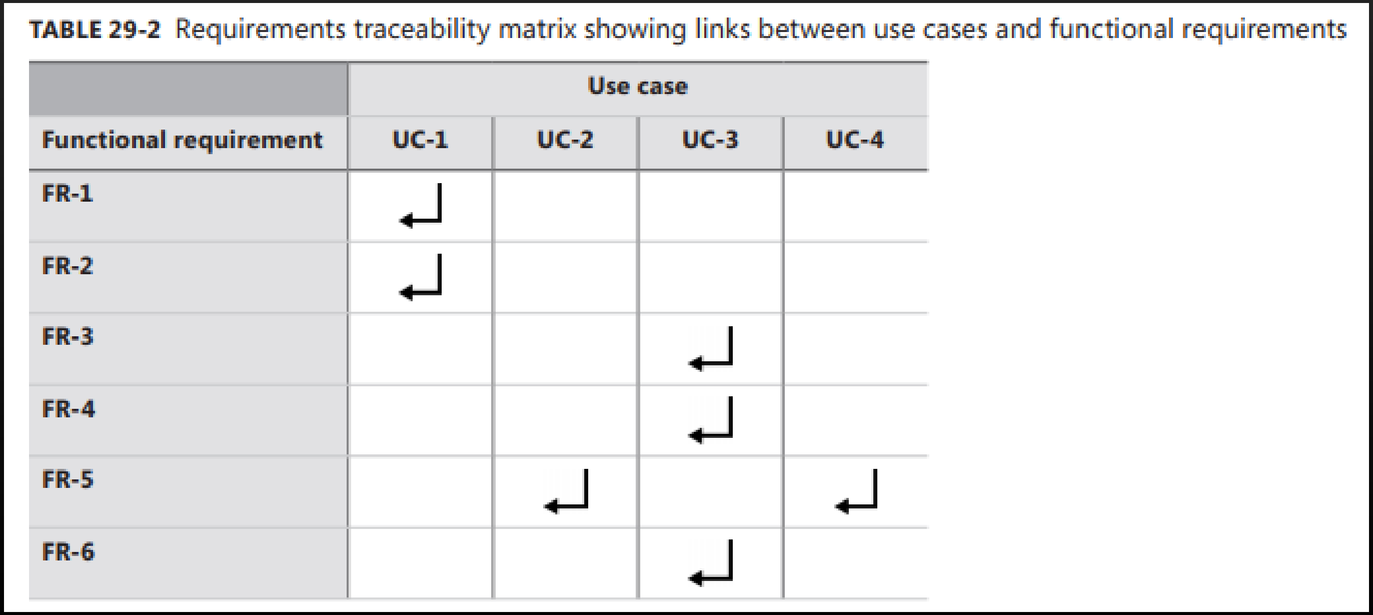


TABLE 29-2 Requirements traceability matrix showing links between use cases and functional requirements

	Use case			
Functional requirement	**UC-1**	**UC-2**	**UC-3**	**UC-4**
FR-1	↲			
FR-2	↲			
FR-3			↲	
FR-4			↲	
FR-5		↲		↲
FR-6			↲	

TABLE 3-1 Requirements engineering good practices

Elicitation	Analysis	Specification	Validation
■ Define vision and scope	■ Model the application environment	■ Adopt requirement document templates	■ Review the requirements
■ Identify user classes	■ Create prototypes	■ Identify requirement origins	■ Test the requirements
■ Select product champions	■ Analyze feasibility	■ Uniquely label each requirement	■ Define acceptance criteria
■ Conduct focus groups	■ Prioritize requirements	■ Record business rules	■ Simulate the requirements
■ Identify user requirements	■ Create a data dictionary	■ Specify nonfunctional requirements	
■ Identify system events and responses	■ Model the requirements		
■ Hold elicitation interviews	■ Analyze interfaces		
■ Hold facilitated elicitation workshops	■ Allocate requirements to subsystems		
■ Observe users performing their jobs			
■ Distribute questionnaires			
■ Perform document analysis			
■ Examine problem reports			
■ Reuse existing requirements			

Requirements management	Knowledge	Project management
■ Establish a change control process	■ Train business analysts	■ Select an appropriate life cycle
■ Perform change impact analysis	■ Educate stakeholders about requirements	■ Plan requirements approach
■ Establish baselines and control versions of requirements sets	■ Educate developers about application domain	■ Estimate requirements effort
■ Maintain change history	■ Define a requirements engineering process	■ Base plans on requirements
■ Track requirements status	■ Create a glossary	■ Identify requirements decision makers
■ Track requirements issues		■ Renegotiate commitments
■ Maintain a requirements traceability matrix		■ Manage requirements risks
■ Use a requirements management tool		■ Track requirements effort
		■ Review past lessons learned

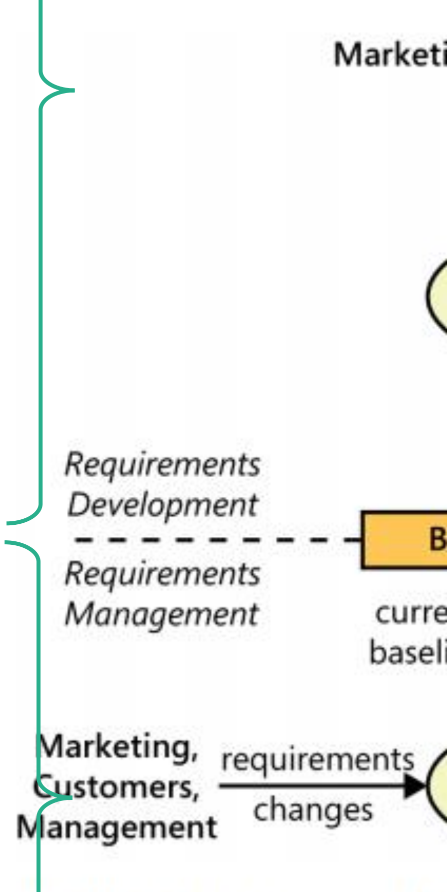


FIGURE 1-5 The boundary be

Requirement attributes to consider:

- Date the requirement was created
- Current version number of the requirement
- Author who wrote the requirement
- Priority
- Status
- Origin or source of the requirement
- Rationale behind the requirement
- Release number or iteration to which the requirement is allocated
- Stakeholder to contact with questions or to make decisions about proposed changes
- Validation method to be used or acceptance criteria

Requirements Management Tool

An RM tool provides a robust solution to the limitations of storing requirements in documents.

Small project teams can get away with just entering the requirements text and several attributes of each requirement.

Larger project teams will benefit from letting users import requirements from source documents, define attribute values, filter and display the database contents, export requirements in various formats, define traceability links, and connect requirements to items stored in other software development tools.

E.g.: IBM Rational DOORS

What are the features of a requirements management tool?

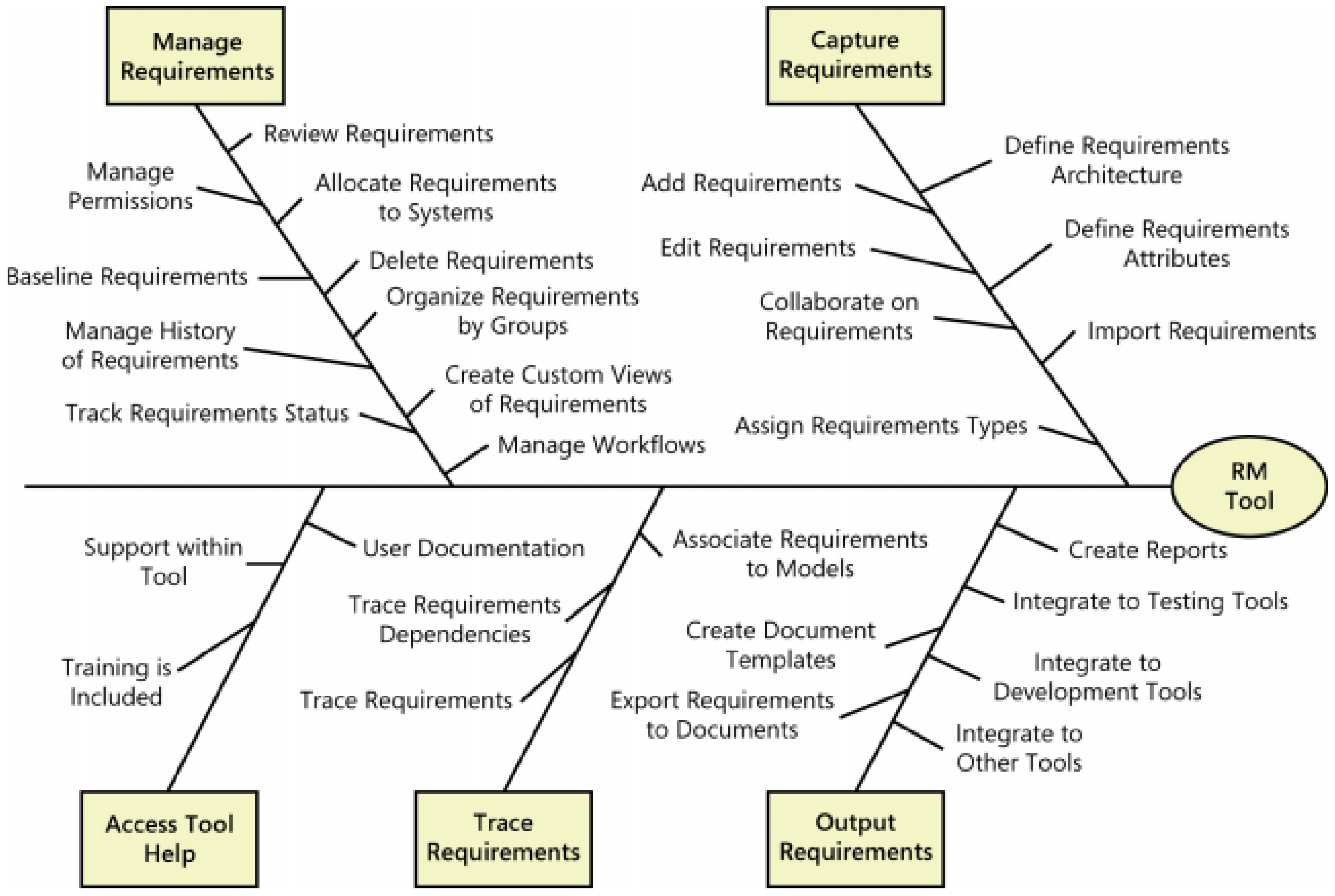
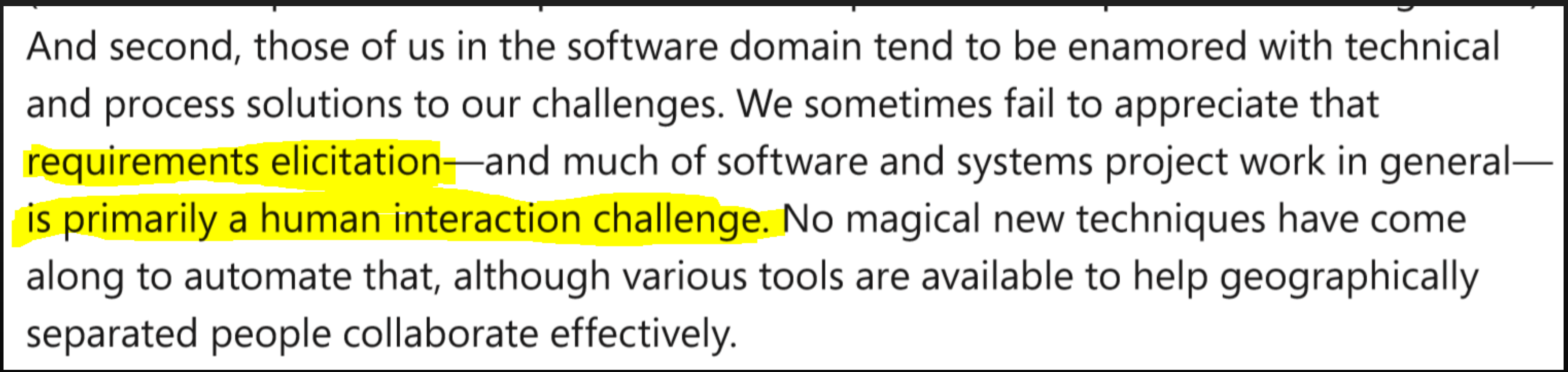

Manage Requirements
Review Requirements
Manage Permissions
Allocate Requirements to Systems
Baseline Requirements
Delete Requirements
Organize Requirements by Groups
Manage History of Requirements
Create Custom Views of Requirements
Track Requirements Status
Manage Workflows
Capture Requirements
Define Requirements Architecture
Add Requirements
Define Requirements Attributes
Edit Requirements
Collaborate on Requirements
Import Requirements
Assign Requirements Types
RM Tool
Support within Tool
User Documentation
Training is Included
Access Tool Help
Associate Requirements to Models
Trace Requirements Dependencies
Trace Requirements
Trace Requirements
Create Document Templates
Export Requirements to Documents
Create Reports
Integrate to Testing Tools
Integrate to Development Tools
Integrate to Other Tools
Output Requirements

FIGURE 30-1 Common RM tool features.

In: https://www.topresume.com/



And second, those of us in the software domain tend to be enamored with technical and process solutions to our challenges. We sometimes fail to appreciate that requirements elicitation—and much of software and systems project work in general—is primarily a human interaction challenge. No magical new techniques have come along to automate that, although various tools are available to help geographically separated people collaborate effectively.

In: Wiegers, "Software Requirements"

Readings & references

Core readings	Suggested readings
• [Wiegers13] – Multiple chapters used.	• [Dennis15] – Chap. 3 – Requirements Determination • [Pressman15] – Chap. 8 – Understanding Requirements