

47006- ANÁLISE E MODELAÇÃO DE SISTEMAS

# Requirements determination: finding what to build

Ilídio Oliveira

v2020/10/21, TP

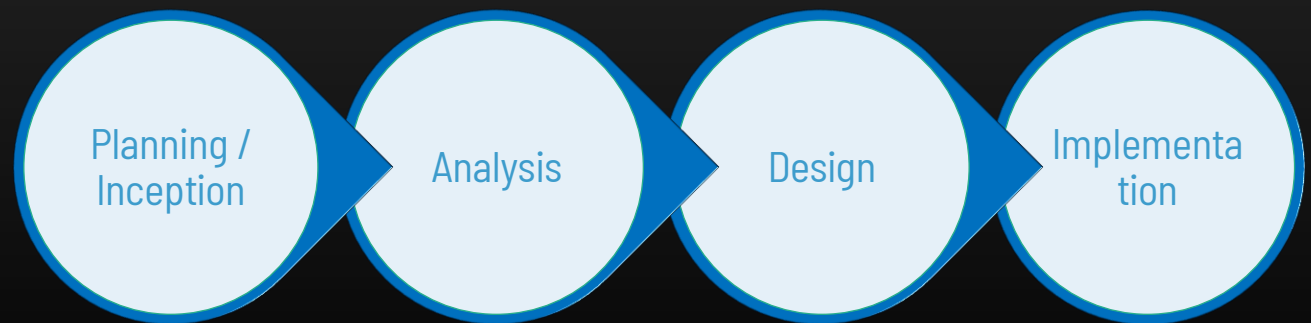
# Learning objectives for this lecture

- Distinguish functional and non-functional requirements
- Enumerate requirements gathering techniques and argument when to use each one
- Describe requirements documentation techniques
- Understand the relation between requirements and use cases
- Identify the requirements related disciplines and activities in the OpenUP

# Requirements determination

The single most critical step of the entire SDLC

- Changes can be made easily in this stage
- Most (>50%) system failures are due to problems with requirements

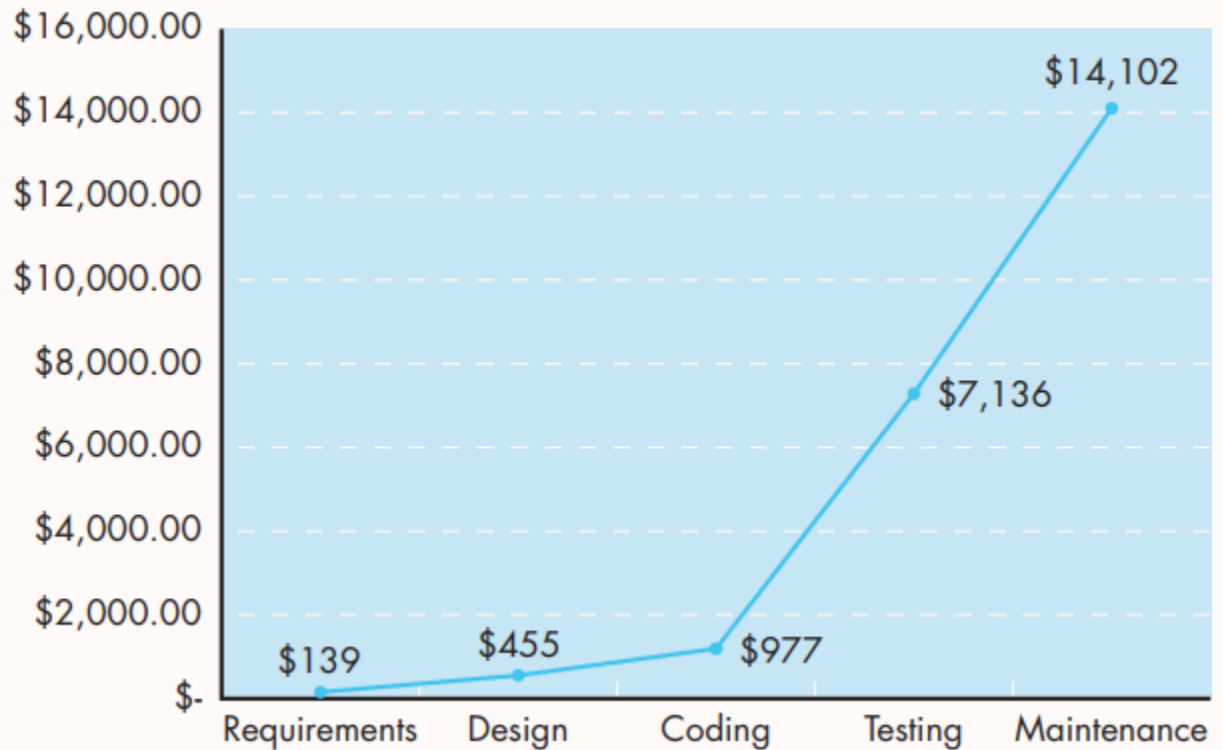


# What is the impact of correcting “errors” in the software?

**FIGURE 19.2**

Relative cost of correcting errors and defects

Source: Adapted from [Boe01b].



# Why conduct “requirements determination” activities?

## Purpose

To convert high level business requests into detailed requirements that can be used as inputs for creating models

## What is a requirement?

A statement of what the system must do or a characteristic it must have.

Will later evolve into a technical description of how the system will be implemented.

## Project “mission”

Develop a system that conforms to the established requirements (capabilities & operating conditions)

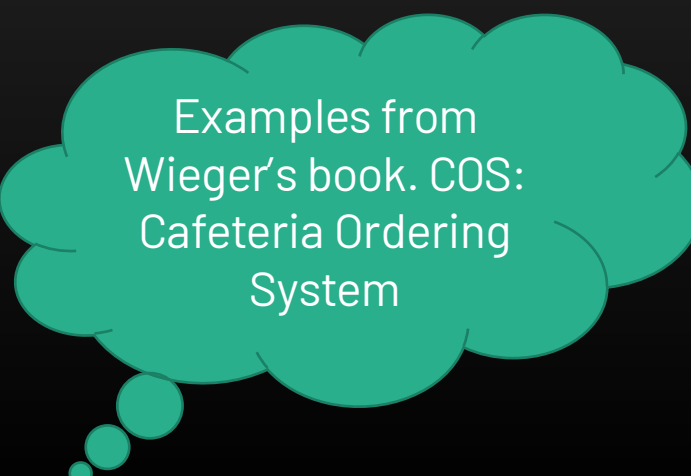
# Sample (functional) requirements

## **Ordering multiple meals and multiple food items**

The COS shall permit the user to order multiple identical meals, up to the fewest available units of any menu item in the order.

## **Delivery or pickup**

The Patron shall specify whether the order is to be picked up or delivered.



Examples from  
Wieger's book. COS:  
Cafeteria Ordering  
System

# Product specification includes quality attributes



The same functional capabilities, very different quality attribs



# What are the types of requirements?

## Types of requirements

Functional: relates to a process or data (*things the system can perform*)

Non-functional: relates to a system quality (*how well must the system do the operation*)

## "sample" requirements

- The system should search patients by name, using at least two words that should occur in sequence in the name.
- The system will retrieve the doctor's daily appointments in <1 sec.



# Function or non-functional: how to tell?

## Functional requirements

Capture the intended behavior

- **Services and functions** that the system must perform

Captured in use cases descriptions (usage scenarios)

Can be supplemented with behavior diagrams: activities, sequence,...

## Non-functional requirements

Global restrictions/operating constraints in the software

- E.g.: user friendliness, portability, robustness,...

a.k.a. quality attributes

- they refer to the expected degree of a certain desired quality

Usually, they are not limited to a function/module, rather **cross-cutting**

# Sample requirements definition

## Nonfunctional Requirements

### 1. Operational Requirements

- 1.1. The system will operate in Windows environment.
- 1.2. The system should be able to connect to printers wirelessly.
- 1.3. The system should automatically back up at the end of each day.

### 2. Performance Requirements

- 2.1. The system will store a new appointment in 2 seconds or less.
- 2.2. The system will retrieve the daily appointment schedule in 2 seconds or less.

### 3. Security Requirements

- 3.1. Only doctors can set their availability.
- 3.2. Only a manager can produce a schedule.

### 4. Cultural and Political Requirements

- 4.1. No special cultural and political requirements are anticipated.

## Functional Requirements

### 1. Manage Appointments

- 1.1. Patient makes new appointment.
- 1.2. Patient changes appointment.
- 1.3. Patient cancels appointment.

### 2. Produce Schedule

- 2.1. Office Manager checks daily schedule.
- 2.2. Office Manager prints daily schedule.

### 3. Record Doctor Availability

- 3.1. Doctor updates schedule

Adapted from: Dennis et al,  
"Systems Analysis and Design: An  
Object Oriented Approach with  
UML", 5th ed.

# Wieger's software quality attributes (\*-ies)

**TABLE 14-1** Some software quality attributes

External quality	Brief description
Availability	The extent to which the system's services are available when and where they are needed
Installability	How easy it is to correctly install, uninstall, and reinstall the application
Integrity	The extent to which the system protects against data inaccuracy and loss
Interoperability	How easily the system can interconnect and exchange data with other systems or components
Performance	How quickly and predictably the system responds to user inputs or other events
Reliability	How long the system runs before experiencing a failure
Robustness	How well the system responds to unexpected operating conditions
Safety	How well the system protects against injury or damage
Security	How well the system protects against unauthorized access to the application and its data
Usability	How easy it is for people to learn, remember, and use the system
Internal quality	Brief description
Efficiency	How efficiently the system uses computer resources
Modifiability	How easy it is to maintain, change, enhance, and restructure the system
Portability	How easily the system can be made to work in other operating environments
Reusability	To what extent components can be used in other systems
Scalability	How easily the system can grow to handle more users, transactions, servers, or other extensions
Verifiability	How readily developers and testers can confirm that the software was implemented correctly

Credit: Wiegers '13

# FURPS categories of quality attributes

## Functionality

- assessed by evaluating the feature set and capabilities of the program, security (functions)

## Usability

- assessed by considering human factors, overall aesthetics, consistency, and documentation.

## Reliability

- evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure

## Performance

- measured using processing speed, response time, resource consumption, throughput, and efficiency.

## Supportability

- combines extensibility, adaptability, and serviceability (i.e., **maintainability**) and in addition, testability, compatibility, configurability, the ease with which a system can be installed, and the ease with which problems can be localized

# What can be involved in “performance” (as a quality attribute)?

**TABLE 14-2** Some aspects of performance

Performance dimension	Example
Response time	Number of seconds to display a webpage
Throughput	Credit card transactions processed per second
Data capacity	Maximum number of records stored in a database
Dynamic capacity	Maximum number of concurrent users of a social media website
Predictability in real-time systems	Hard timing requirements for an airplane’s flight-control system
Latency	Time delays in music recording and production software
Behavior in degraded modes or overloaded conditions	A natural disaster leads to a massive number of emergency telephone system calls

## Requirements

**STRQ1:** Want to be able to transfer funds from other accounts (not necessarily held with this firm) to a trading account.

**STRQ2:** State and federal regulations require monthly reports of account activity. Refer to specification RUFS-1234 for details of the information required.

**STRQ3:** The system should allow the use of any browser.

**STRQ4:** Customers want to manage their retirement funds.

**STRQ5:** Must be able to upgrade the system without taking it offline.

**STRQ6:** The system should allow traders to trade in multiple markets across the world.

**STRQ7:** Must be able to provide convenient answers to customer's most common questions.

**STRQ8:** The system must provide a secure environment that prohibits fraudulent access.

**STRQ9:** Need a way to train customers in the use of the system quickly and conveniently.

**STRQ10:** The system must operate on hardware that falls under the company's current maintenance contracts.

**STRQ11:** Need to be able to maintain the system with our current IT hardware and skills. Refer to enterprise architecture document EA-1234 for details.

**STRQ12:** Need account activity statements for tax reporting.

**STRQ13:** The system must provide all the basic capabilities of a normal stock broking firm.

**STRQ14:** Need to be able to perform research on any given stock.

**STRQ15:** The system must allow traders to obtain up-to-date news and alerts on nominated stock.

**STRQ16:** The system must provide current and historical information on Trading Accounts. Such as number of shares held, current price, total Trading Account value

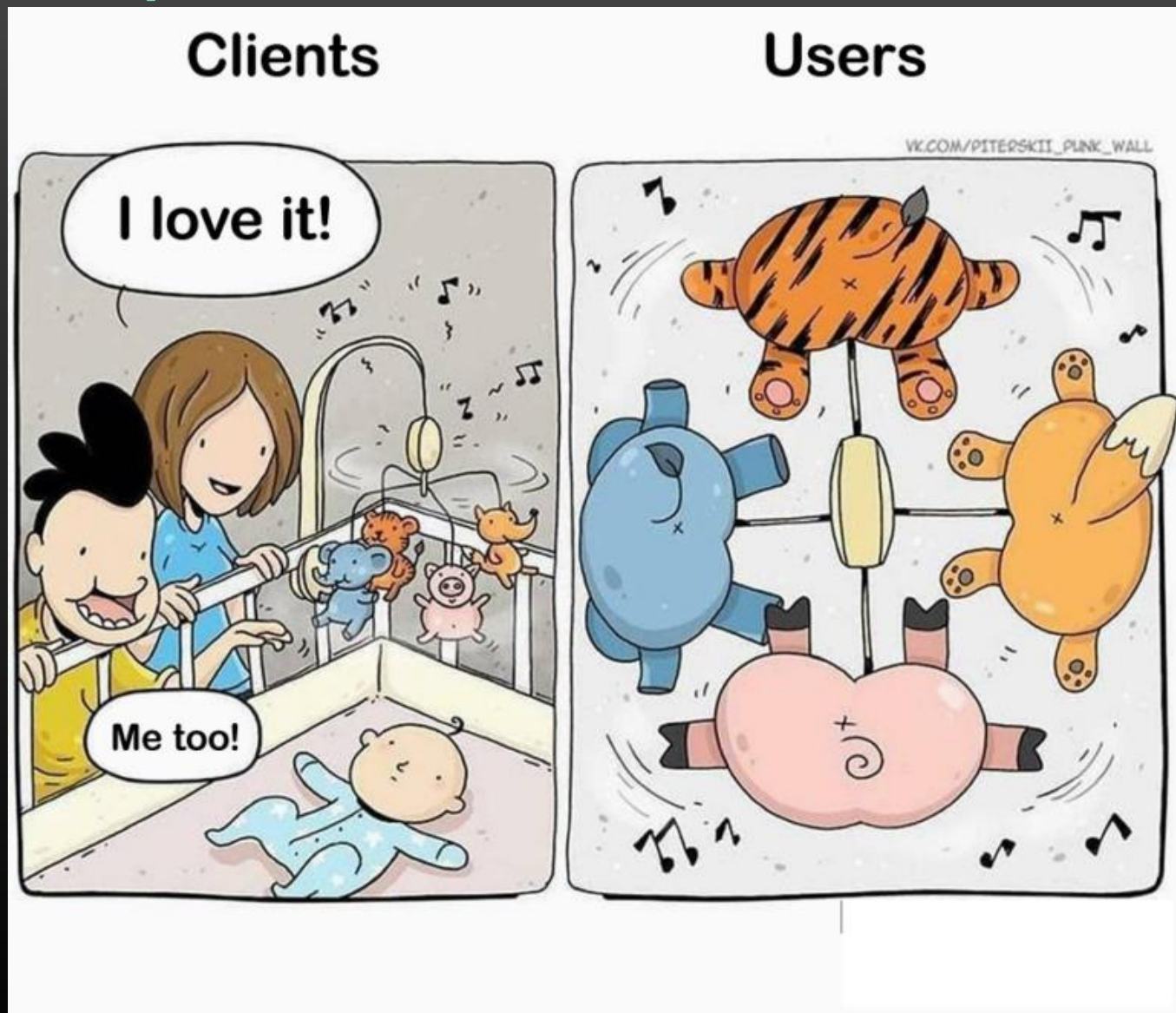
**STRQ17:** The system shall provide the following types of trades: Market Trades (buy and sell), Limit Trades (buy and sell), and transfers between mutual funds.

*Software designers tend to focus on the problem to be solved. Just don't forget that the FURPS attributes are always part of the problem. They must be considered.*

## Gathering requirements



# Customer/promoter vs users





# Requirements Gathering Techniques

## Process used to:

Uncover all requirements (those uncovered late in the process are more difficult to incorporate)

Build support and trust among users/stakeholders

## Which technique(s) to use?


1. Interviews
2. Joint Application Development (JAD)
3. Questionnaires
4. Document analysis
5. Observation

# OpenUP requirements practices

## OpenUP practices

[OpenUP](#) > Practices >  
Technical Practices >  
Shared Vision >  
Requirements Gathering  
Techniques

Details available on the  
OpenUp documentation



OpenUP

---

### Requirements Gathering Techniques

After you have identified these sources, there are a number of techniques that the following will describe the various techniques, followed by a brief discussion of

To get the requirements down on paper, you can do one or more of the following:

- Conduct a brainstorming session
- Interview users
- Send questionnaires
- Work in the target environment
- Study analogous systems
- Examine suggestions and problem reports
- Talk to support teams
- Study improvements made by users
- Look at unintended uses
- Conduct workshops
- Demonstrate prototypes to stakeholders

The best idea is to get the requirements down quickly and then to encourage the users to make in those corrections, and repeat the cycle. Do it now, keep it small, and correct it as you can devise, but expect to keep on correcting it throughout the process. Success is to correct it immediately.

# Requirements-Gathering Techniques Compared

A combination of techniques may be used

Document analysis & observation require little training; JAD sessions can be very challenging

	Interviews	Joint Application Design	Questionnaires	Document Analysis	Observation
Type of information	As-is, improvements, to-be	As-is, improvements, to-be	As-is, improvements	As-is	As-is
Depth of information	High	High	Medium	Low	Low
Breadth of information	Low	Medium	High	High	Low
Integration of information	Low	High	Low	Low	Low
User involvement	Medium	High	Low	Low	Low
Cost	Medium	Low-Medium	Low	Low	Low to Medium

Credit: Dennis et al, "Systems Analysis and Design: An Object Oriented Approach with UML", 5th ed.

# Alternative Techniques

## Domain analysis

Study the characteristics of the domain

Learn from others

Increase the readiness for scaling to more customers

## Concept Maps

Represent meaningful relationships between concepts

Focus individuals on a small number of key ideas

## User Stories, Story Cards & Task Lists

Associated with agile development methods

Very low tech, high touch, easily updatable, and very portable

Captured using story cards (index cards)

Capture both functional and nonfunctional requirements.



“house wish-list...”



# Requirements elicitation

System to build

Requirements documented in analysis

Inconsistent

Not feasible  
(technology)

Missing  
requirements

superfluous

Initial scope



# What are the requirements elicitation activities?

The heart of requirements development is **elicitation**

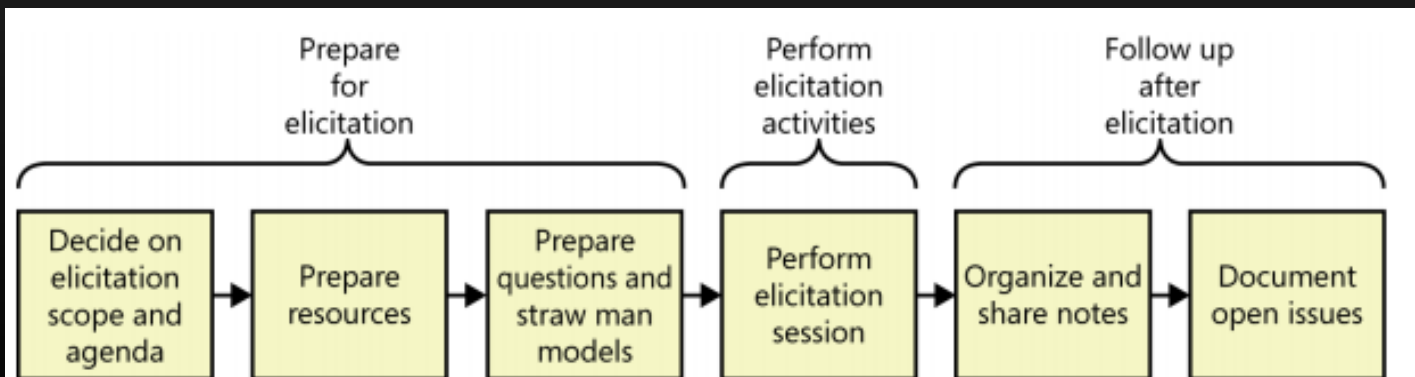
- the process of identifying the needs and constraints of the various stakeholders for a software system.

Elicitation is not the same as “gathering requirements.”

- Nor is it a simple matter of transcribing exactly what users say.

Elicitation is a collaborative and analytical process that includes activities to collect, discover, extract, and define requirements. Elicitation is used to discover business, user, functional, and nonfunctional requirements.

Requirements elicitation is perhaps the most challenging, critical, error-prone, and communication-intensive aspect of software development.



**FIGURE 7-2** Activities for a single requirements elicitation session.

Figure 7-3 suggests the elicitation techniques that are most likely to be useful for various types of projects. Select the row or rows that represent characteristics of your project and read to the right to see which elicitation techniques are most likely to be helpful (marked with an X). For instance, if you're developing a new application, you're likely to get the best results with a combination of stakeholder interviews, workshops, and system interface analysis. Most projects can make use of interviews and workshops. Focus groups are more appropriate than workshops for mass-market software because you have a large external user base but limited access to representatives. These suggestions for elicitation techniques are just that—suggestions. For instance, you might conclude that you do want to apply user interface analysis on mass-market software projects.

	Interviews	Workshops	Focus groups	Observations	Questionnaires	System interface analysis	User interface analysis	Document analysis
Mass-market software	x		x		x			
Internal corporate software	x	x	x	x		x		x
Replacing existing system	x	x		x		x	x	x
Enhancing existing system	x	x				x	x	x
New application	x	x				x		
Packaged software implementation	x	x		x		x		x
Embedded systems	x	x				x		x
Geographically distributed stakeholders	x	x			x			

**FIGURE 7-3** Suggested elicitation techniques by project characteristic.



# Requirements documentation

## Requisitos são muitas vezes elencados em listas “o sistema deve...”

#	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:

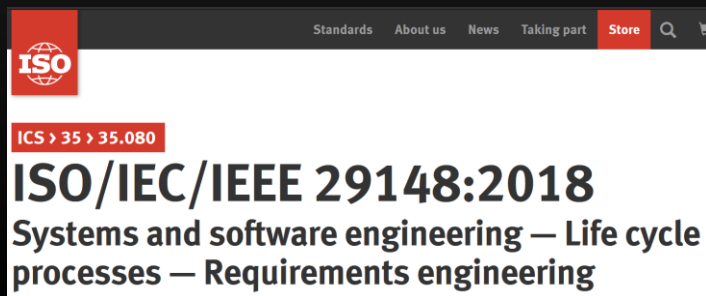
- can be verified,
- has to be met or possessed by a system to solve a stakeholder problem or to achieve a stakeholder objective,
- is qualified by measurable conditions and bounded by constraints, and
- defines the performance of the system when used by a specific stakeholder or the corresponding capability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

## 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.



The screenshot shows the top of a web page for the ISO/IEC/IEEE 29148:2018 standard. At the top, there is a navigation bar with links: Standards, About us, News, Taking part, and Store. Below this is the ISO logo. Under the logo, the text 'ICS > 35 > 35.080' is displayed. The main title of the page is 'ISO/IEC/IEEE 29148:2018' in a large, bold font. Below the title, the subtitle 'Systems and software engineering — Life cycle processes — Requirements engineering' is written in a smaller font.

# S. M. A. R. T. (Specific, Measurable, Attainable, Relevant and time-sensitive)

## Step 5: Specify well-structured quality requirements

Simplistic quality requirements such as “The system shall be user-friendly” or “The system shall be available 24x7” aren’t useful. The former is far too subjective and vague; the latter is rarely realistic or necessary. Neither is measurable. **Such requirements provide little guidance to developers.** So the final step is to craft specific and verifiable requirements from the information that was elicited regarding each quality attribute. When writing quality requirements, keep in mind the useful SMART mnemonic—make them *Specific, Measurable, Attainable, Relevant, and Time-sensitive*.

"The system shall be 100% reliable and 100% available".

"The system shall have a minimum response to a query of 1 second irrespective of system load".

*AVL-1. The system shall be at least 95 percent available on weekdays between 6:00 A.M. and midnight Eastern Time, and at least 99 percent available on weekdays between 3:00 P.M. and 5:00 P.M. Eastern Time.*

*IOP-1. The Chemical Tracking System shall be able to import any valid chemical structure from the ChemDraw (version 13.0 or earlier) and MarvinSketch (version 5.0 or earlier) tools.*

*PER-1. Authorization of an ATM withdrawal request shall take no more than 2.0 seconds.*

*PER-2. The anti-lock braking system speed sensors shall report wheel speeds every 2 milliseconds with a variation not to exceed 0.1 millisecond.*

*PER-3. Webpages shall fully download in an average of 3 seconds or less over a 30 megabits/second Internet connection.*

*PER-4. At least 98 percent of the time, the trading system shall update the transaction status display within 1 second after the completion of each trade.*

# Requirements and use cases

# Which main elicitation approaches exist?

What is the goal the **user** wants to achieve? vs.

What capability should the **product**/system possess?

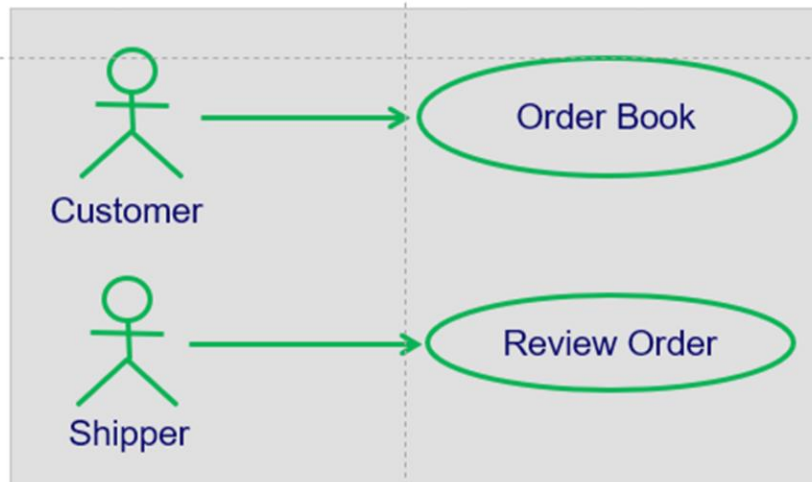


# OpenUP recommended practices

## *Use Case Driven Development* 🏆



- 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.

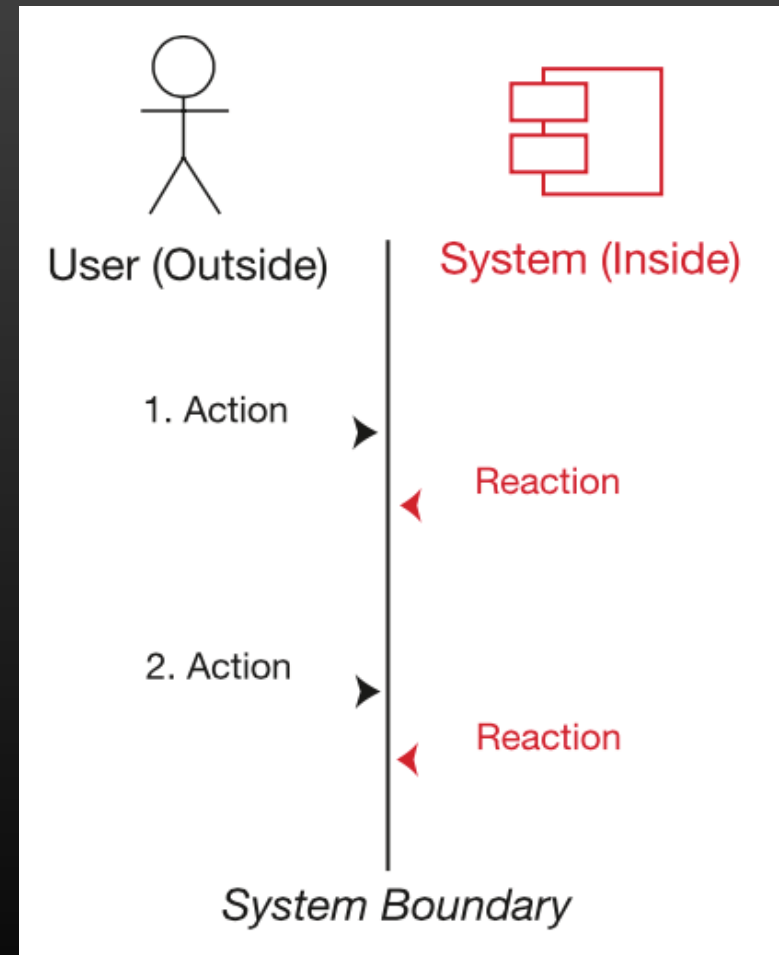




# The use case captures a dialog between the actor(s) and the system

## CaU: Pay at checkout

1. Customer arrives at POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price is calculated from a set of price rules. Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any)



# The use case details describe an interaction

## HOW TO WRITE A USE CASE: THE THREE MAGIC QUESTIONS

Well, OK, this whole chapter describes how to write a use case. But when writing use cases, you need to keep asking the following three fundamental questions:<sup>1</sup>

1. What happens?

(This gets your “sunny-day scenario” started.)

2. And then what happens?

(Keep asking this question until your “sunny-day scenario” is complete.)

3. What else might happen?

(Keep asking this one until you’ve identified all the “rainy-day scenarios” you can think of, and described the related behavior.)


# "Request a Chemical" use case specification

ID and Name:	UC-4 Request a Chemical		
Created By:	Lori	Date Created:	8/22/13
Primary Actor:	Requester	Secondary Actors:	Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.		
Trigger:	Requester indicates that he wants to request a chemical.		
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.		
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.		
Normal Flow:	<b>4.0 Request a Chemical from the Chemical Stockroom</b> <ol style="list-style-type: none"> <li>1. Requester specifies the desired chemical.</li> <li>2. System lists containers of the desired chemical that are in the chemical stockroom, if any.</li> <li>3. System gives Requester the option to View Container History for any container.</li> <li>4. Requester selects a specific container or asks to place a vendor order (see 4.1).</li> <li>5. Requester enters other information to complete the request.</li> <li>6. System stores the request and notifies the Chemical Stockroom.</li> </ol>		
Alternative Flows:	<b>4.1 Request a Chemical from a Vendor</b> <ol style="list-style-type: none"> <li>1. Requester searches vendor catalogs for the chemical (see 4.1.E1).</li> <li>2. System displays a list of vendors for the chemical with available container sizes, grades, and prices.</li> <li>3. Requester selects a vendor, container size, grade, and number of containers.</li> <li>4. Requester enters other information to complete the request.</li> <li>5. System stores the request and notifies the Buyer.</li> </ol>		
Exceptions:	<b>4.1.E1 Chemical Is Not Commercially Available</b> <ol style="list-style-type: none"> <li>1. System displays message: No vendors for that chemical.</li> <li>2. System asks Requester if he wants to request another chemical (3a) or to exit (4a).</li> </ol>		

Use case:	Brief description:
Create new assignment	The Teaching Staff creates a new Activity of type Assignment, directly inserting it in the page layout. The assignment must define a title and a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from registered students.

<b>Use case:</b>	<u>Add new assignment</u>
<b>Brief description:</b>	The Faculty creates assignments for students, directly inserting it in the course page. The assignment defines a time period for submissions and can be configured to work with individual or group submissions. The assignment is listed in the student view and on the specified date (or immediately, if none is given) accepts submissions from students.
<b>Basic flow:</b>	<ol style="list-style-type: none"><li>1. Log-in using corporate IdP.</li><li>2. Select desired course.</li><li>3. Turn editing mode on.</li><li>4. Add Assignment activity in the page layout.</li><li>5. Configure Assignment activity.</li><li>6. Commit changes.</li></ol>
<b>Alternative flows:</b>	Step 1: IdP unavailable. Step 4/5: Instead of a new, empty assignment, the user may reuse an existing one.
<b>Open issues:</b>	Step 3/4. The course is closed. Are changes allowed to past courses? Step 5. The browser does not accept the rich text editor. Default to plain text?

# Putting it together

 OpenUP

Glossary | Feedback | About

Print


Where am I | Tree Sets |

Team



- Introduction to OpenUP
- Getting Started
- Delivery Processes
- Practices
  - Management Practices
  - Technical Practices
    - Concurrent Testing
    - Continuous Integration
    - Evolutionary Architecture
    - Evolutionary Design
    - Shared Vision
    - Test Driven Development
    - Use Case Driven Development
      - How to Adopt the Use Case Driven Development
    - Key Concepts

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](#)

 Expand All Sections  Collapse All Sections

**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.

 [Back to top](#)



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
<ul style="list-style-type: none"><li>• [Dennis15] – Chap. 3 – Requirements Determination</li></ul>	<ul style="list-style-type: none"><li>• [Pressman15] – Chap. 8 – Understanding Requirements</li><li>• [Wiegers13] – Chap. 1-3</li></ul>