

**COMSATS University, Islamabad**

# REQUIREMENTS ENGINEERING

***Prof. Dr. Sohail Asghar***  
*Department of Computer Science*  
*COMSATS University Islamabad*

# Why Requirements Engineering Practices

- The development of software has been plagued with problems since 1960s.
- Delivered late and over budget.
- Don't do what users really want and
- Often never used to their full effectiveness by the people who have paid for them.
- There is rarely a single reason (or a single solution) for these problems
- The major contributory factor is difficulties with the software requirements.

# Problems in Requirements Engineering Process and Practices

Many problems in the software world arise from shortcomings in the ways that

- people learn about,
- document,
- agree upon and
- modify the product's requirements.

Common problem areas are

- informal information gathering,
- implied functionality,
- miscommunicated assumptions,
- poorly specified requirements, and
- a casual change process.

# Importance of Requirements

- Requirements are the foundation for
  - both the software development and the project management activities,
  - all stakeholders should commit to applying requirements practices that are known to yield superior-quality products.

# What is "requirement"

- *Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.*
  - This definition acknowledges the diverse types of information that collectively are referred to as "the requirements."
  - Requirements encompass both the user's view of the external system behavior and the developer's view of some internal characteristics.
  - They include both the behavior of the system under specific conditions and those properties that make the system suitable—and maybe even enjoyable—for use by its intended operators.

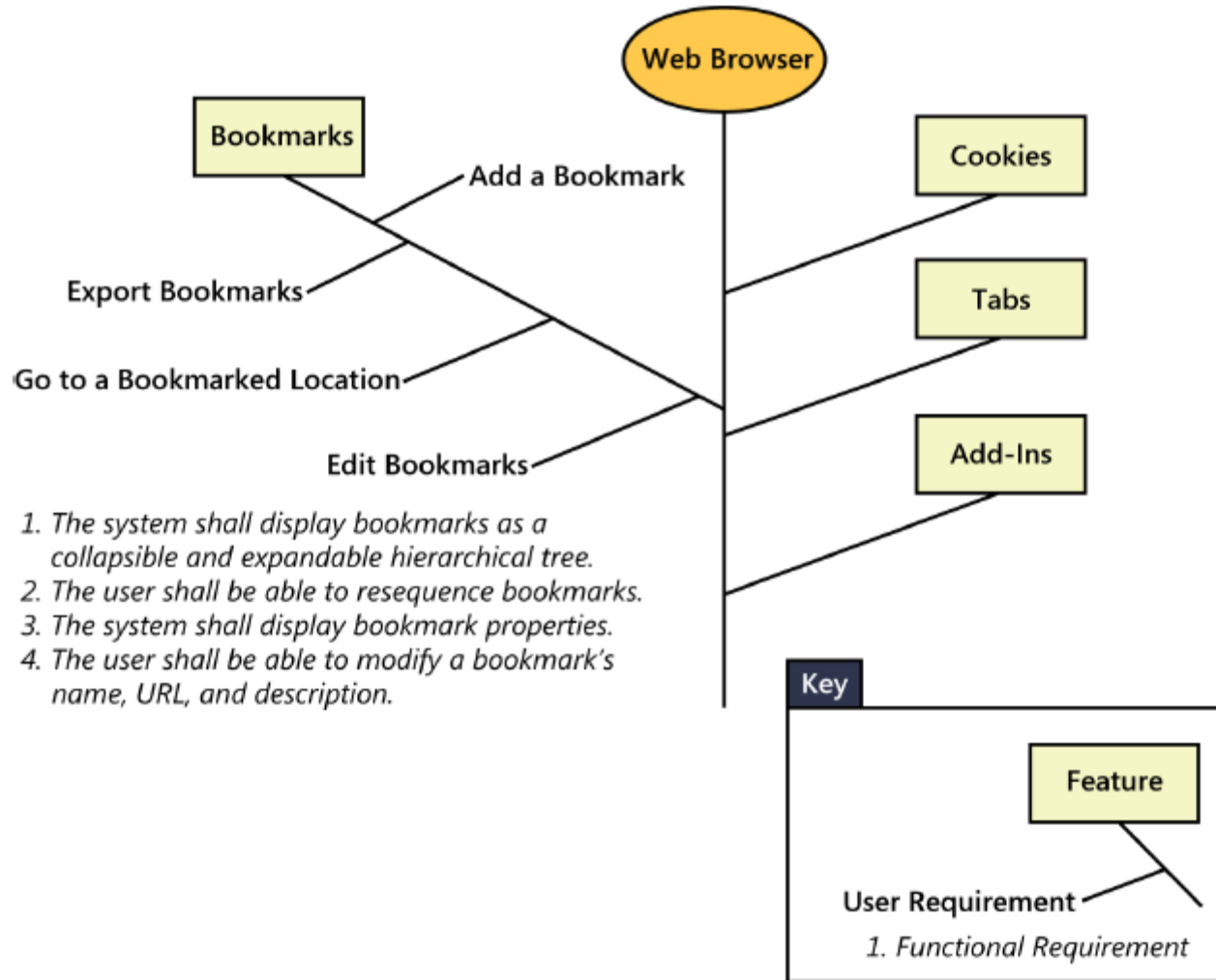
# Types of requirements

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.

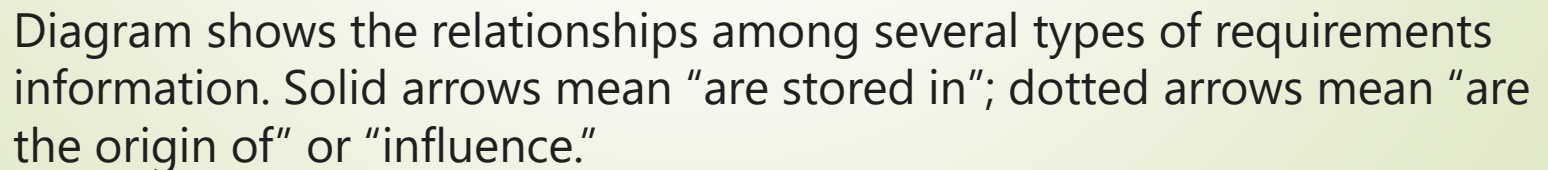


Term	Definition
Functional requirement	A description of a behavior that a system will exhibit under specific conditions.
Nonfunctional requirement	A description of a property or characteristic that a system must exhibit or a constraint that it must respect
Quality attribute	A kind of nonfunctional requirement 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.

# Relationships among features, user







# Requirements Elicitation

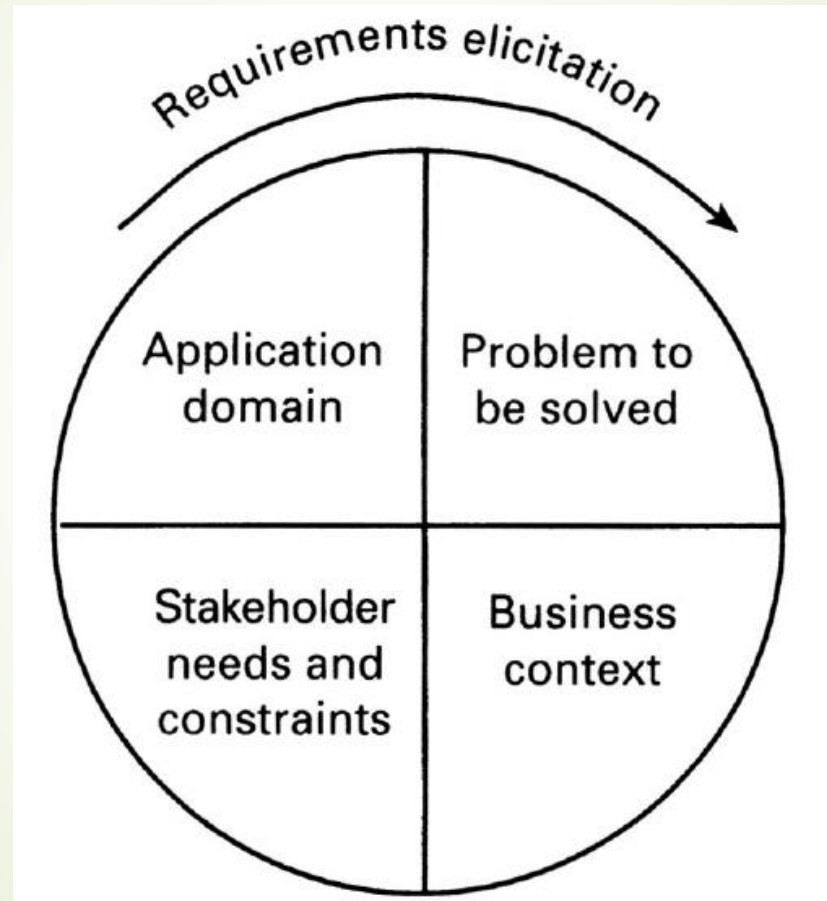
# Introduction- Requirements elicitation

- Requirements elicitation is the usual name given to activities involved in discovering the requirements of the system.
- System developers and engineers work with customers and end-users to find out about
  - The problem to be solved, the system services, the required performance of the system, hardware constraints, and so on.
- This doesn't just involve asking people what they want;
  - It requires a careful analysis of the organization, the application domain and business processes where the system will be used.

# Elicitation activities

- Elicitation encompasses all of the activities involved with discovering requirements, such as interviews, workshops, document analysis, prototyping, and others.
- The key actions are:
  - Identifying the product's expected user classes and other stakeholders.
  - Understanding user tasks and goals and the business objectives with which those tasks align.
  - Learning about the environment in which the new product will be used.
  - Working with individuals who represent each user class to understand their functionality needs and their quality expectations.

# Components of requirements elicitation



# Elicitation activities

- ▀ Application domain understanding
  - ▀ Application domain knowledge is knowledge of the general area where the system is applied.
    - ▀ For Example: to understand the requirements for a railway signaling system, you must have background knowledge about the operation of railways and the physical characteristics of trains.
- ▀ Problem understanding
  - ▀ The details of the specific customer problem where the system will be applied must be understood.
    - ▀ For a railway signaling system, you must know the way in which speed limits are applied to particular track segments.
- ▀ Business understanding
  - ▀ You must understand how systems interact and contribute to overall business goals. (Means, the contribution of the system in business goal)
- ▀ Understanding the needs and constraints of system stakeholders
  - ▀ You must understand, in detail, the specific needs of people who require system support in their work.



# Identifying Stakeholders

# Stakeholders

- Stakeholders are the people who are needed to ensure the success of a project.
- It is essential to find out whom they are, what their attitudes are, and what their interests are.
- Some of them have to contribute with money and effort, and they must feel they get something in return, otherwise they won't support the project and they may even obstruct it.

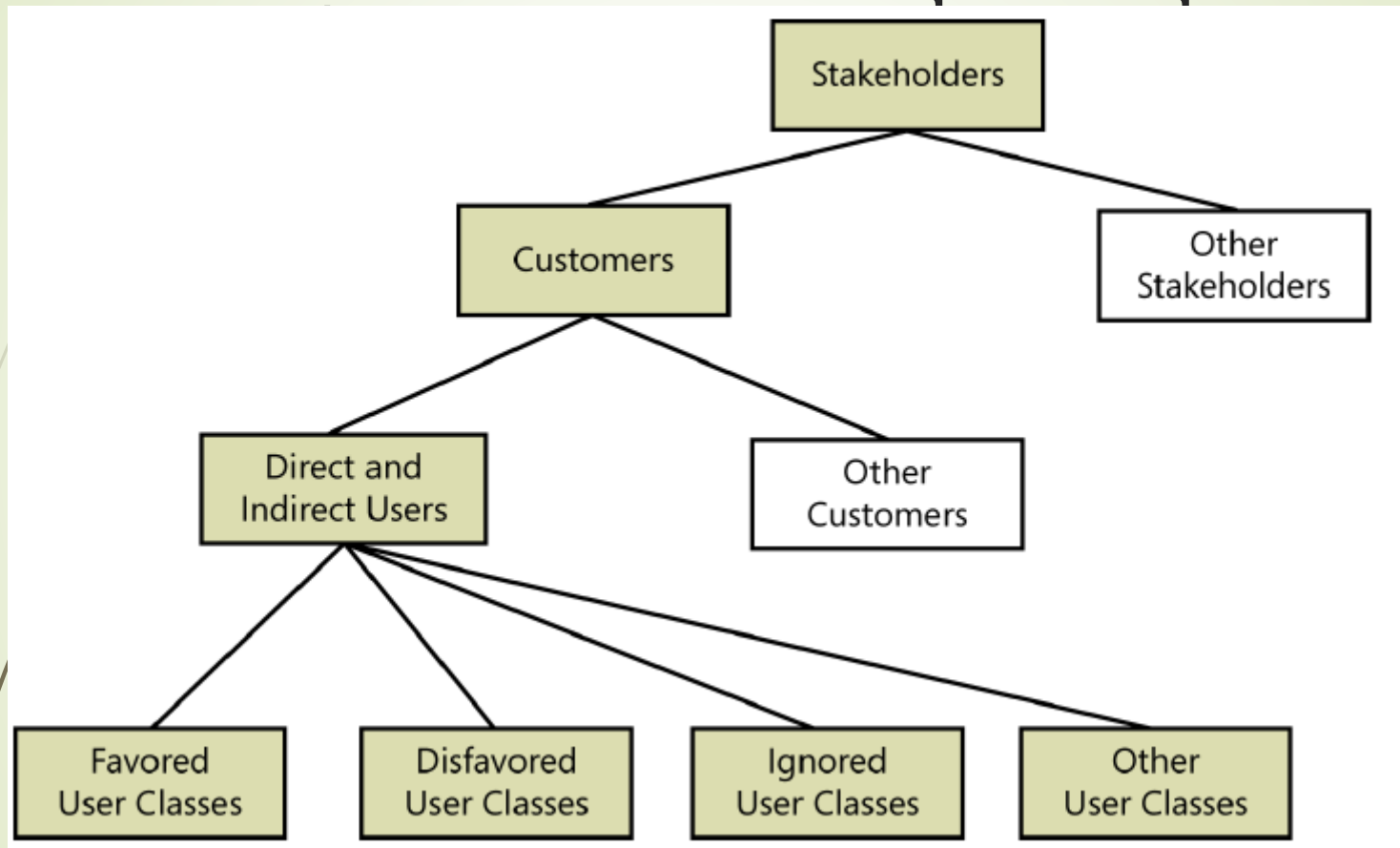
## During stakeholder analysis you try to get answers to these questions:

- Who are the stakeholders? (Initially you know only a few of them.)
- What goals do they see for the system? (Many stakeholders look not only at their own goals, but also at other stakeholder's goals.)
- Why would they like to contribute? (Their reward.)
- What risks and costs do they see?
- What kind of solutions, suppliers, and resources do they see? (Should we replace the old system or extend it? Make it ourselves or buy it?)

# Classifying users

- In the following respects, and you can group users into a number of distinct *user classes* based on these sorts of differences:
  - Their access privilege or security levels (such as ordinary user, guest user, administrator)
  - The tasks they perform during their business operations
  - The features they use
  - The frequency with which they use the product
  - Their application domain experience and computer systems expertise
  - The platforms they will be using (desktop PCs, laptop PCs, tablets, smartphones, specialized devices)
  - Their native language
  - Whether they will interact with the system directly or indirectly

# A hierarchy of stakeholders,



# Class Exercise

*List the possible stakeholders for a library cataloguing system.*



# Possible Stockholders for *library cataloguing system*

- Library users
- Library staff responsible for cataloguing
- Library management
- Library staff responsible for providing user assistance
- Book publishers (indirect)
- System developers
- Managers of other library automation systems

# Specific elicitation techniques

- ▶ Interviews
- ▶ Observations and social analysis
- ▶ User interface analysis
- ▶ Prototyping
- ▶ Document Studies
- ▶ Questionnaires
- ▶ Brainstorm
- ▶ Focus groups
- ▶ Study similar companies

# Interviews

- ▶ The requirements engineer or analyst discusses the system with different stakeholders and builds up an understanding of their requirements.
- ▶ Types of interview
- ▶ Closed interviews. The requirements engineer looks for answers to a pre-defined set of questions
- ▶ Open interviews There is no predefined agenda and the requirements engineer discusses, in an open -ended way, what stakeholders want from the system.

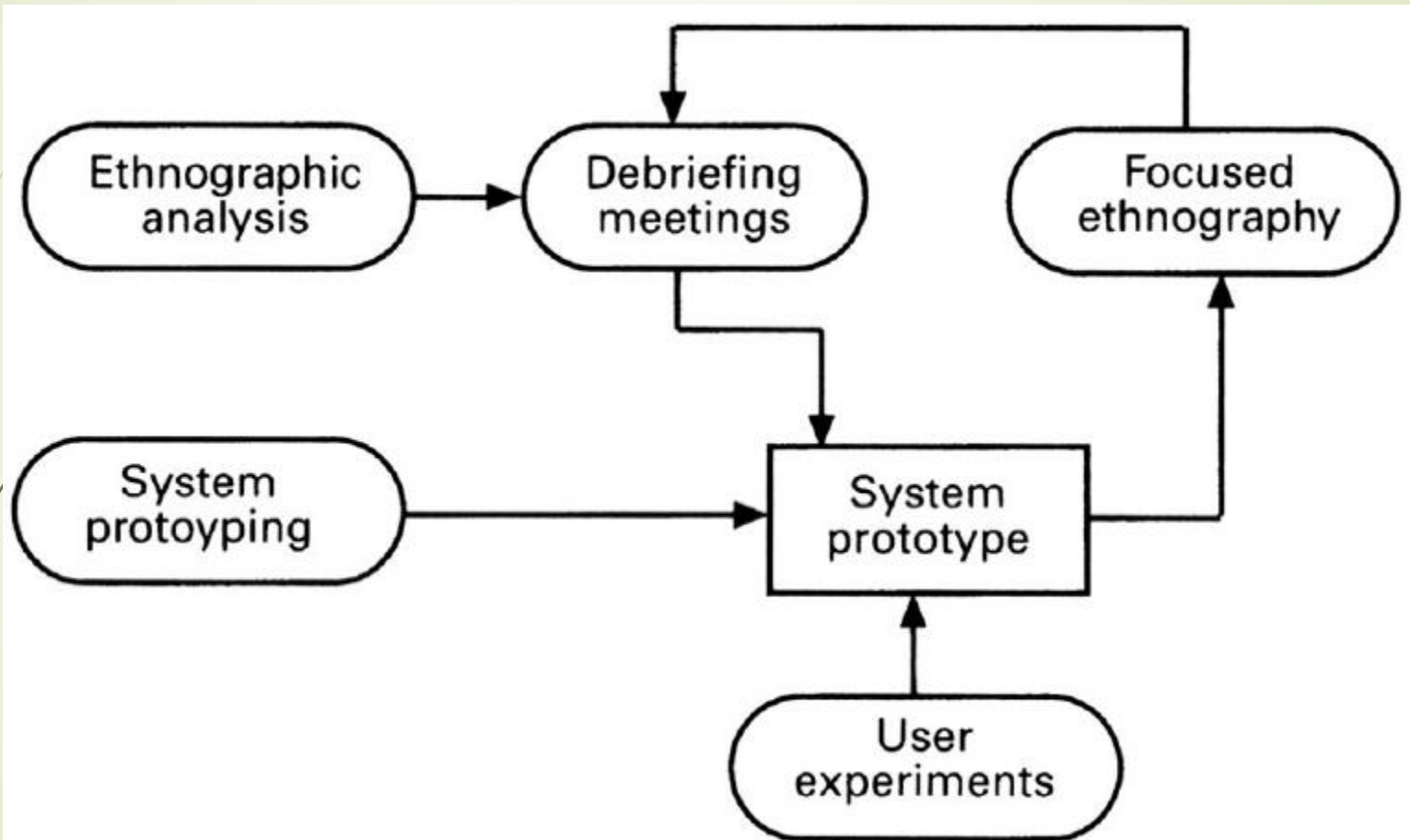
# Observation and social analysis

- People often find it hard to describe what they do because it is so natural to them. Sometimes, the best way to understand it is to observe them at work
- Ethnography is a technique from the social sciences which has proved to be valuable in understanding actual work processes
- Actual work processes often differ from formal, prescribed processes
- An ethnographer spends some time observing people at work and building up a picture of how work is done

# Ethnography guidelines

- Assume that people are good at doing their job and look for non-standard ways of working
- Spend time getting to know the people and establish a trust relationship
- Keep detailed notes of all work practices. Analyze them and draw conclusions from them
- Combine observation with open-ended interviewing
- Organize regular de-briefing session where the ethnographer talks with people outside the process
- Combine ethnography with other elicitation techniques

# Ethnography in elicitation



**Figure 3.7:** Using ethnography in requirements elicitation.



# User interface analysis

- User interface (UI) analysis is an independent elicitation technique in which you study existing systems to discover user and functional requirements.
- It's best to interact with the existing systems directly, but if necessary you can use screen shots.
- User manuals for purchased packaged-software implementations often contain screen shots that will work fine as a starting point. If there is no existing system,
- you might be able to look at user interfaces of similar products.

- When working with packaged solutions or an existing system, UI analysis can help you identify a complete list of screens to help you discover potential features.
- By navigating the existing UI, you can learn about the common steps users take in the system and draft use cases to review with users.
- UI analysis can reveal pieces of data that users need to see.
- It's a great way to get up to speed on how an existing system works (unless you need a lot of training to do so).
- Instead of asking users how they interact with the system and what steps they take, perhaps you can reach an initial understanding yourself.

# Prototyping

- A prototype is an initial version of a system which may be used for experimentation
- Prototypes are valuable for requirements elicitation because users can experiment with the system and point out its strengths and weaknesses.
- They have something concrete to criticize
- Rapid development of prototypes is essential so that they are available early in the elicitation process

## Prototyping benefits

- The prototype allows users to experiment and discover what they really need to support their work
- Establishes feasibility and usefulness before high development costs are incurred
- Essential for developing the 'look and feel' of a user interface
- Can be used for system testing and the development of documentation
- Forces a detailed study of the requirements which reveals inconsistencies and omissions

# Types of prototyping

## ➤ **Throw-away prototyping**

- Intended to help elicit and develop the system requirements.
- The requirements which should be prototyped are those which cause most difficulties to customers and which are the hardest to understand. Requirements which are well-understood need not be implemented by the prototype.

## ➤ **Evolutionary prototyping**

- Intended to deliver a workable system quickly to the customer.
- Therefore, the requirements which should be supported by the initial versions of this prototype are those which are well-understood and which can deliver useful end-user functionality. It is only after extensive use that poorly understood requirements should be implemented.

# Document studies

- Document studies are used to cross-check the interview information.
- It is a fast way to get information about data in the old “database”.
- The analyst studies existing documents such as forms, letter files, computer logs and documentation of the existing computer system.
- He may also print screen dumps from the existing system.



# Questionnaires

- It is used to get information from many people.
- Can be used in two ways
  1. To get statistical evidence for an assumption,
    - People ask closed questions such as, “How easy is it to get customer statistics with the present system: very difficult, quite difficult, easy, very easy...”
  2. To gather opinions and suggestions.
    - People ask open questions, e.g. “What are the three biggest problems in your daily work?” and, “What are your suggestions for better IT support of your daily work?”

# Brainstorming

- In a brainstorming session RE team gather together with a group of people,
- Create a stimulating and focused atmosphere, and let people come up with ideas without risk of being ridiculed.
- The facilitator notes down all ideas on a whiteboard.
- An important rule of the game is not to criticize any idea. Even seemingly stupid ideas may turn out to have a valuable “diamond” seed in them.
- The facilitator may finish the session with a joint round where participants prioritize the ideas.

# Focus groups

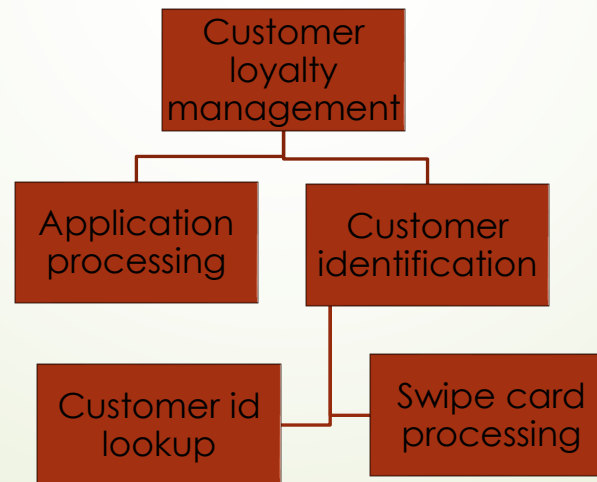
- Focus groups resemble brainstorming sessions, but are more structured.
- It starts with a phase where participants come up with problems in the current way of doing things.
- Next comes a phase where participants try to imagine an ideal way of doing things.
- The group also tries to explain why the ideas are good, which helps formulate goals and requirements for the new system.
- At the end of the session, each group identifies their high priority issues.
- When later prioritizing the requirements, it is important that each stakeholder group gets solutions to some of their high-priority issues.
  - If a stakeholder group doesn't get anything in return, they are rarely willing to contribute to the system.

# Study similar companies

- One of the best sources of realistic ideas is to see what other companies do to handle problems similar to your own.
- A study of their procedures and comparison with your own can give you many ideas.
- Most importantly, a visit to their site makes it easier to imagine how the new system could work.

# Laddering

- Requirements engineer asks the customer short prompting questions to elicit requirements.
- Follow up questions then posed to dig deeper below the surface.
- The resultant information from the responses is then organized into a tree-like structure.



# Consider the following sequence of laddering questions and responses for the POS system

- RE: Name a key feature of the system ?
- Customer: Customer identification.
- RE: How do you identify a customer?
- Customer: They can swipe their loyalty card
- RE: What if the customer forgets their card?
- Customer: They can look up by phone number.
- RE: When do you get the customer's phone number?
- Customer: When customers complete the application for the loyalty card.
- RE: How do customers complete the application? .....





# How do you know when you're done with requirements elicitation?

- The users can't think of any more use cases or user stories. Users tend to identify user requirements in sequence of decreasing importance.
- Users propose new scenarios, but they don't lead to any new functional requirements. A "new" use case might really be an alternative flow for a use case you've already captured.
- Users repeat issues they already covered in previous discussions.
- Suggested new features, user requirements, or functional requirements are all deemed to be out of scope.
- Proposed new requirements are all low priority.
- The users are proposing capabilities that might be included "sometime in the lifetime of the product" rather than "in the specific product we're talking about right now."
- Developers and testers who review the requirements for an area raise few questions.

# Identifying User Requirements

# Techniques to identify user requirements

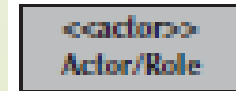
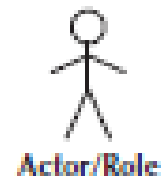
- **Use case (use case diagram + detail use case)** is an effective technique for interactive end-user applications.
- **Event- response table** is for real-time system in which most of the functionalities are performed at backend.

# Elements of Use-Case Diagrams

- The elements of a use-case diagram include
  - actors, use cases, subject boundaries, and a set of relationships among actors, actors and use cases, and use cases.
- These relationships consist of
  - association, include, extend, and generalization relationships.

# Actors

- **An actor:** Is a person or system that derives benefit from and is external to the subject.
- Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a rectangle with <<actor>> in it (alternative).
- Is labeled with its role.
- Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead.
- Is placed outside the system boundary.



# Secondary Actor

- ▶ To complete a business process when a primary actor required to interface some external system
  - ▶ for example using ATM machine, in order to perform user transaction ATM software is required to authenticate fetch user information from main banking system to complete the transaction, here the main banking system is **secondary actor**.

# Use Case

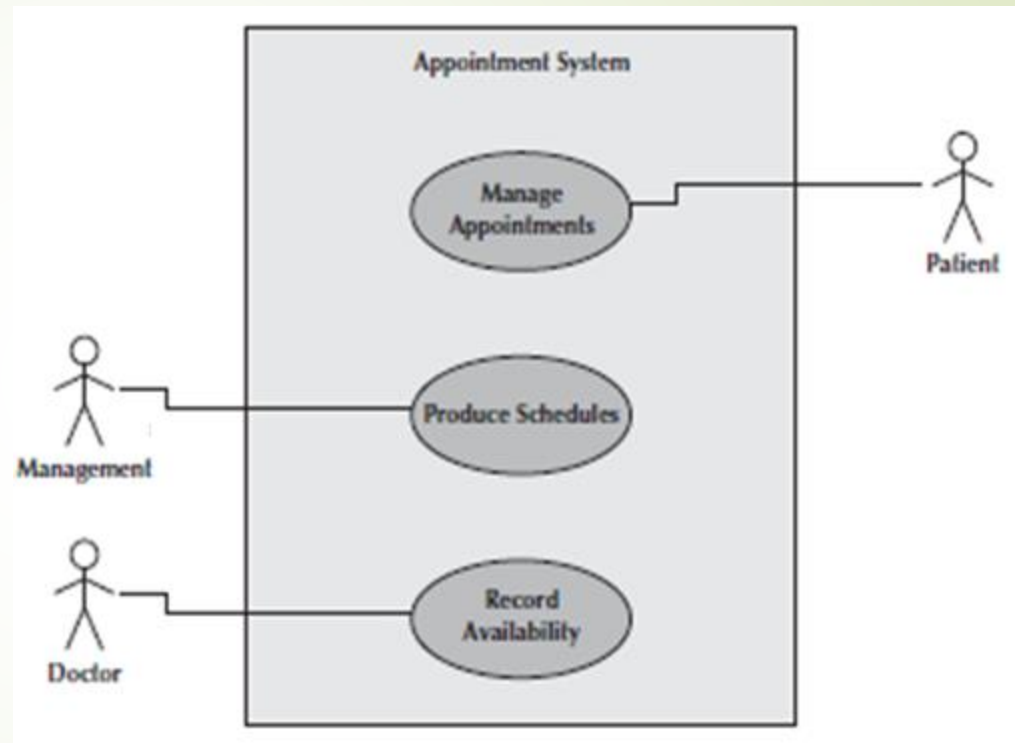
- Represents a major piece of system functionality.
- Can extend another use case.
- Can include another use case.
- Is placed inside the system boundary.
- Is labeled with a descriptive verb–noun phrase.





# A System Boundary

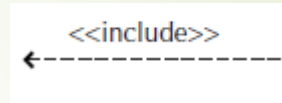
- Includes the name of the subject inside or on top.
- Represents the scope of the System.



# Use case Diagram Relationships

## ➤ An association relationship:

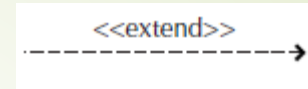
- Links an actor with the use case(s) with which it interacts.



## ➤ An include relationship:

- Represents the inclusion of the functionality of one use case within another.
- Has an arrow drawn from the base use case to the used use case.

# Use case Diagram Relationships



## ➤ An extend relationship:

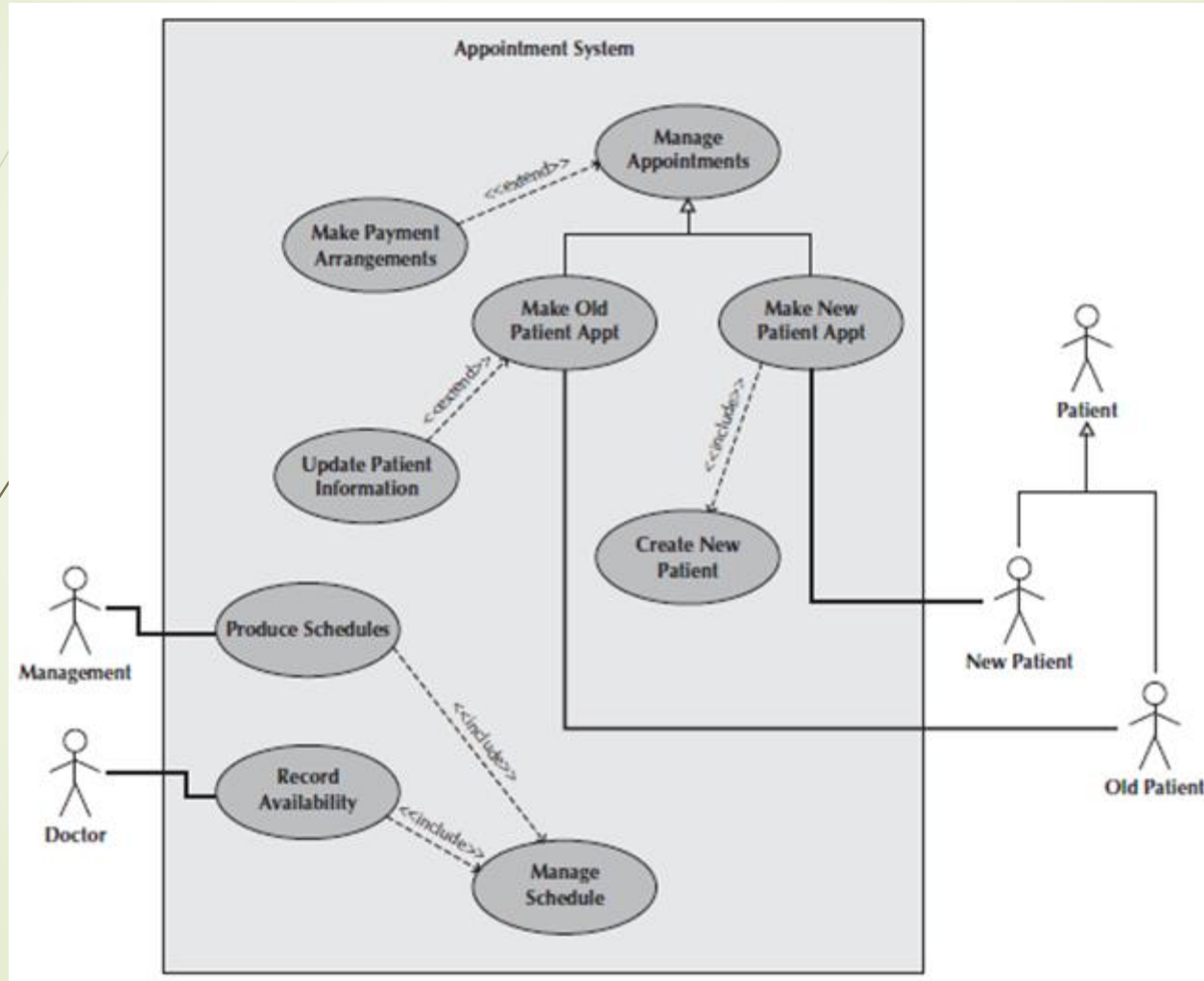
- Represents the extension of the use case to include optional behavior.
- Has an arrow drawn from the extension use case to the base use case.



## ➤ A generalization relationship:

- Represents a specialized use case to a more generalized one.
- Has an arrow drawn from the specialized use case to the base use case.

# Example



## Case Study (Analyzing User requirement)

- *The Chemical Tracking System (CTS) project was holding its first requirements elicitation workshop to learn what chemists would need to do with the system. The participants included a business analyst, Lori; the product champion for the chemists, Tim; two other chemist representatives, Sandy and Peter; and the lead developer, Ravi.*

*“Tim, Sandy, and Peter have identified 14 use cases that chemists would need to perform using the Chemical Tracking System,” Lori told the group. “You said the use case called ‘Request a Chemical’ is top priority and Tim already wrote a brief description for it, so let’s begin there. Tim, how do you visualize the process to request a chemical with the system?”*

*“First,” said Tim, “you should know that only people who have been authorized by their lab managers are allowed to request chemicals.”*

*“Okay, that sounds like a business rule,” Lori replied.*

*“I’ll start a list of business rules because we’ll probably find others. It looks like we’ll have to verify that the user is on the approved list.”*



Lori then guided the group through a discussion of how they envisioned creating a request for a new chemical. She used flipcharts and sticky notes to collect information about **preconditions**, **postconditions**, and the **interactions between the user and the system**.

*Lori asked how a session would be different if the user were requesting a chemical from a vendor rather than from the stockroom.*

*She asked what could go wrong and how the system should handle each error condition.*

*After about 30 minutes, the group had a solid handle on how a user would request a chemical. They moved on to the next use case.*

# Format and guideline to write use case(s)

<b>Use Case ID:</b>	Enter a unique numeric identifier for the Use Case. e.g. UC-1.2.1
<b>Use Case Name:</b>	Enter a short name for the Use Case using an active verb phrase. e.g. Withdraw Cash
<b>Actors:</b>	[An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles, identified from the customer community that will use the product. Name the actor that will be initiating this use case (primary) and any other actors who will participate in completing the use case (secondary).]
<b>Description:</b>	[Provide a brief description of the reason for and outcome of this use case.]
<b>Trigger:</b>	[Identify the event that initiates the use case. This could be an external business event or system event that causes the use case to begin, or it could be the first step in the normal flow.]

<b>Preconditions:</b>	<p>[List any activities that must take place, or any conditions that must be true, before the use case can be started. Number each pre-condition. e.g.</p> <ol style="list-style-type: none"><li>1. Customer has active deposit account with ATM privileges</li><li>2. Customer has an activated ATM card.]</li></ol>
<b>Postconditions:</b>	<p>[Describe the state of the system at the conclusion of the use case execution. Should include both minimal guarantees (what must happen even if the actor's goal is not achieved) and the success guarantees (what happens when the actor's goal is achieved. Number each post-condition. e.g.</p> <ol style="list-style-type: none"><li>1. Customer receives cash</li><li>2. Customer account balance is reduced by the amount of the withdrawal and transaction fees]</li></ol>

**Normal Flow:**

[Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the use case name and description.

1. Customer inserts ATM card
2. Customer enters PIN
3. System prompts customer to enter language  
performance English or Spanish
4. System validates if customer is in the bank network
5. System prompts user to select transaction type
6. Customer selects Withdrawal From Checking
7. System prompts user to enter withdrawal amount
8. ...
9. System ejects ATM card]

**Alternative Flows:**

[Alternative Flow  
1 – Not in  
Network]

[Document legitimate branches from the main flow to handle special conditions (also known as extensions). For each alternative flow reference the branching step number of the normal flow and the condition which must be true in order for this extension to be executed. e.g. Alternative flows in the Withdraw Cash transaction:

4a. In step 4 of the normal flow, if the customer is not in the bank network

1. System will prompt customer to accept network fee
2. Customer accepts
3. Use Case resumes on step 5

4b. In step 4 of the normal flow, if the customer is not in the bank network

1. System will prompt customer to accept network fee
2. Customer declines
3. Transaction is terminated
4. Use Case resumes on step 9 of normal flow

Note: Insert a new row for each distinctive alternative flow. ]



<b>Exceptions:</b>	<p>[Describe any anticipated error conditions that could occur during execution of the use case, and define how the system is to respond to those conditions. e.g. Exceptions to the Withdraw Case transaction</p> <p>2a. In step 2 of the normal flow, if the customer enters and invalid PIN</p> <ol style="list-style-type: none"><li>1. Transaction is disapproved</li><li>2. Message to customer to re-enter PIN</li><li>3. Customer enters correct PIN</li><li>4. Use Case resumes on step 3 of normal flow]</li></ol>
<b>Includes:</b>	<p>[List any other use cases that are included (“called”) by this use case. Common functionality that appears in multiple use cases can be split out into a separate use case that is included by the ones that need that common functionality. e.g. steps 1-4 in the normal flow would be required for all types of ATM transactions- a Use Case could be written for these steps and “included” in all ATM Use Cases.]</p>



<b>Special/Quality Requirements (other information):</b>	[Identify any additional requirements, such as nonfunctional requirements, for the use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.]
<b>Assumptions:</b>	[List any assumptions that were made in the analysis that led to accepting this use case into the product description and writing the use case description. e.g. For the Withdraw Cash Use Case, an assumption could be: The Bank Customer understands either English or Spanish language.]
<b>Business Rule:</b>	Some business rules constrain which roles can perform all or parts of a use case. Perhaps only users who have certain privilege levels can perform specific alternative flows. That is, the rule might impose preconditions that the system must test before letting the user proceed. Business rules can influence specific steps in the normal flow by defining valid input values or dictating how computations are to be performed.

# Example

<b>ID and Name:</b>	<b>UC-1: Order a Meal</b>		
Created By:	Prithvi Raj	Date Created:	October 4, 2013
Primary Actor:	Patron	Secondary Actors:	Cafeteria Inventory System
Description:	A Patron accesses the Cafeteria Ordering System from either the corporate intranet or external Internet, views the menu for a specific date, selects food items, and places an order for a meal to be picked up in the cafeteria or delivered to a specified location within a specified 15-minute time window.		
Trigger:	A Patron indicates that he wants to order a meal.		
Preconditions:	PRE-1. Patron is logged into COS. PRE-2. Patron is registered for meal payments by payroll deduction.		
Postconditions:	POST-1. Meal order is stored in COS with a status of "Accepted." POST-2. Inventory of available food items is updated to reflect items in this order. POST-3. Remaining delivery capacity for the requested time window is updated.		
Normal Flow:	<b>1.0 Order a Single Meal</b> <ol style="list-style-type: none"> <li>1. Patron asks to view menu for a specific date. (see 1.0.E1, 1.0.E2)</li> <li>2. COS displays menu of available food items and the daily special.</li> <li>3. Patron selects one or more food items from menu. (see 1.1)</li> <li>4. Patron indicates that meal order is complete. (see 1.2)</li> <li>5. COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge.</li> <li>6. Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2).</li> <li>7. COS displays available delivery times for the delivery date.</li> <li>8. Patron selects a delivery time and specifies the delivery location.</li> <li>9. Patron specifies payment method.</li> <li>10. COS confirms acceptance of the order.</li> <li>11. COS sends Patron an email message confirming order details, price, and delivery instructions.</li> <li>12. COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times.</li> </ol>		

# Example (Continue)

Alternative Flows:	<p><b>1.1 Order multiple identical meals</b></p> <ol style="list-style-type: none"> <li>1. Patron requests a specified number of identical meals. (see 1.1.E1)</li> <li>2. Return to step 4 of normal flow.</li> </ol> <p><b>1.2 Order multiple meals</b></p> <ol style="list-style-type: none"> <li>1. Patron asks to order another meal.</li> <li>2. Return to step 1 of normal flow.</li> </ol>
Exceptions:	<p><b>1.0.E1 Requested date is today and current time is after today's order cutoff time</b></p> <ol style="list-style-type: none"> <li>1. COS informs Patron that it's too late to place an order for today.</li> <li>2a. If Patron cancels the meal ordering process, then COS terminates use case.</li> <li>2b. Else if Patron requests another date, then COS restarts use case.</li> </ol> <p><b>1.0.E2 No delivery times left</b></p> <ol style="list-style-type: none"> <li>1. COS informs Patron that no delivery times are available for the meal date.</li> <li>2a. If Patron cancels the meal ordering process, then COS terminates use case.</li> <li>2b. Else if Patron requests to pick the order up at the cafeteria, then continue with normal flow, but skip steps 7 and 8.</li> </ol> <p><b>1.1.E1 Insufficient inventory to fulfill multiple meal order</b></p> <ol style="list-style-type: none"> <li>1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory.</li> <li>2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow.</li> <li>2b. Else if Patron cancels the meal ordering process, then COS terminates use case.</li> </ol>
Priority:	High
Frequency of Use:	Approximately 300 users, average of one usage per day. Peak usage load for this use case is between 9:00 A.M. and 10:00 A.M. local time.
Business Rules:	BR-1, BR-2, BR-3, BR-4, BR-11, BR-12, BR-33
Other Information:	<ol style="list-style-type: none"> <li>1. Patron shall be able to cancel the meal ordering process at any time prior to confirming it.</li> <li>2. Patron shall be able to view all meals he ordered within the previous six months and repeat one of those meals as the new order, provided that all food items are available on the menu for the requested delivery date. (Priority = medium) <i>[Note: You could also show this as an alternative flow for the use case.]</i></li> <li>3. The default date is the current date if the Patron is using the system before today's order cutoff time. Otherwise, the default date is the next day that the cafeteria is open.</li> </ol>
Assumptions:	Assume that 15 percent of Patrons will order the daily special (Source: previous 6 months of cafeteria data).

# Event-response tables

*Consider a complex highway intersection with numerous traffic lights and pedestrian walk signals. There aren't many use cases for a system like this. A driver might want to proceed through the light or to turn left or right. A pedestrian wants to cross the road. Perhaps an emergency vehicle wants to be able to turn the traffic signals green in its direction so it can speed its way to people who need help. Law enforcement might have cameras at the intersection to photograph the license plates of red-light violators. This information alone isn't enough for developers to build the correct functionality.*

# Continue

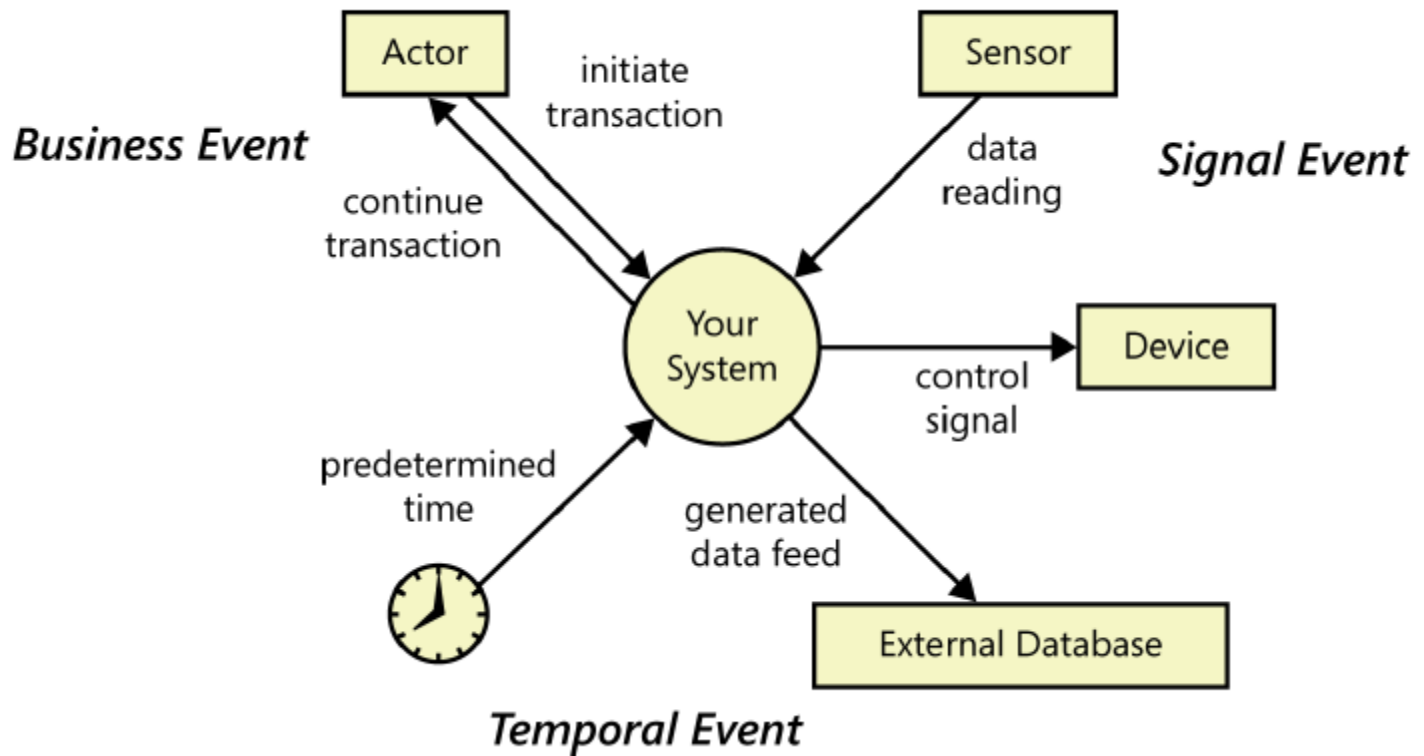
- Another way to approach user requirements is to identify the external events to which the system must respond.
- An *event* is some change or activity that takes place in the user's environment that stimulates a response from the software system (Wiley 2000).
- An *event-response table* (also called an *event table* or an *event list*) itemizes all such events and the behavior the system is expected to exhibit in reaction to each event.



# Types of Events

- **Business event** A business event is an action by a human user that stimulates a dialog with the software, as when the user initiates a use case.
  - The event-response sequences correspond to the steps in a use case or swimlane diagram.
- **Signal event** A signal event is registered when the system receives a control signal, data reading, or
- interrupt from an external hardware device or another software system, such as
  - when a switch closes, a voltage changes, another application requests a service, or a user swipes his finger on a tablet's screen.
- **Temporal event** A temporal event is time-triggered, as when the computer's clock reaches a specified time (say, to launch an automatic data export operation at midnight) or
- when a preset duration has passed since a previous event
  - as in a system that logs the temperature read by a sensor every 10 seconds.

# Systems respond to business, signal, and temporal events





# Identifying Events

- To identify events, consider all *the states associated with the object* you are analyzing,
- and *identify any events that might transition the object* into those states.
- Review context diagrams for any external entities that might initiate an action (trigger an event) or
- require an automatic response (need a temporal event triggered).

# Example

- The highway intersection system described earlier has to deal with various events, including these:
  - A sensor detects a car approaching in one of the through lanes.
  - A sensor detects a car approaching in a left-turn lane.
  - A pedestrian presses a button to request to cross a street.
- One of many timers counts down to zero.
- Exactly what happens in response to an external event depends on the state of the system at the time it detects the event.

# Exceptions

- The system might initiate a timer to prepare to change a light from green to amber and then to red.
- The system might activate a Walk sign for a pedestrian
  - (if the sign currently reads Don't Walk), or change it to a flashing
  - Don't Walk (if the sign currently says Walk), or
  - change it to a solid Don't Walk (if it's currently flashing).
- The BA needs to write the functional requirements to specify both how to detect the events and
- the decision logic to use when combining events with states to produce system behaviors.

# Event Response Table

<i>Event</i>	<i>System State</i>	<i>Response</i>
Road sensor detects vehicle entering left-turn lane.	Left-turn signal is red. Cross-traffic signal is green.	Start green-to-amber countdown timer for cross-traffic signal.
Green-to-amber countdown timer reaches zero.	Cross-traffic signal is green.	1. Turn cross-traffic signal amber. 2. Start amber-to-red countdown timer.
Amber-to-red countdown timer reaches zero.	Cross-traffic signal is amber.	1. Turn cross-traffic signal red. 2. Wait 1 second. 3. Turn left-turn signal green. 4. Start left-turn-signal countdown timer.

# Continue

<i><b>Event</b></i>	<i><b>System State</b></i>	<i><b>Response</b></i>
Pedestrian presses a specific walk-request button.	Pedestrian sign is solid Don't Walk. Walk-request countdown timer is not activated.	Start walk-request countdown timer.
Pedestrian presses walk-request button.	Pedestrian sign is solid Don't Walk. Walk-request countdown timer is activated.	Do nothing.
Walk-request countdown timer reaches zero plus the amber display time.	Traffic signal in walk direction is green.	Change all green traffic signals to amber.
Walk-request countdown timer reaches zero.	Traffic signal in walk direction is amber.	<ol style="list-style-type: none"><li>1. Change all amber traffic signals to red.</li><li>2. Wait 1 second.</li><li>3. Set pedestrian sign to Walk.</li><li>4. Start don't-walk countdown timer.</li></ol>

# Additional Information

- Other information you might want to add to an event-response table includes:
  - The event frequency (how many times the event takes place in a given time period, or a limit to how many times it can occur).
  - Data elements that are needed to process the event.
  - The state of the system after the event responses are executed.

## Note

- Listing the events that cross the system boundary is a useful scoping technique (Wiegiers 2006).
- An event-response table that defines every possible combination of event, state, and response, including exception conditions, can serve as part of the functional requirements for that portion of the system.
- You might model the event-response table in a decision table to ensure that all possible combinations of events and system states are analyzed.
- However, the BA must supply additional functional and nonfunctional requirements.



# Specifying Functional Requirements




# Use cases vs functional requirements

- Software developers don't implement business requirements or user requirements.
- They implement functional requirements, specific bits of system behavior.
- Some practitioners regard the use cases as being the functional requirements.
- However, we have seen many organizations get into trouble when they simply pass their use cases to developers for implementation.
- Why we need functional requirements?
- Use cases describe the user's perspective, looking at the externally visible behavior of the system.
- They don't contain all the information that a developer needs to write the software.



# Example

- The user of an ATM doesn't know about any back-end processing involved, such as communicating with the bank's computer.
- This detail is invisible to the user, yet the developer needs to know about it.
- Developers who receive even fully dressed use cases often have many questions.
- To reduce this uncertainty, consider having a BA explicitly specify the functional requirements necessary to implement each use case (Arlow 1998).



# Deriving functional requirements from the use case

- **Directly derivable**
- Many functional requirements fall right out of the dialog steps between the actor and the system.
- Some are obvious, such as
  - “The system shall assign a unique sequence number to each request.”
- There is no point in duplicating those elsewhere if they're clear from the use case.

# Deriving functional requirements from the use case

- **Derive through further analysis**
- Other functional requirements don't appear in the use case description.
- Example
  - For instance, the way use cases are typically documented does not specify what the system should do if a precondition is not satisfied.
- The BA must derive those missing requirements and communicate them to developers and testers (Wiegers 2006).
- This analysis to get from the user's view of the requirements to the developer's view is one of the many ways the BA adds value to a project.



# Guidelines for writing requirements

- Two important goals of writing requirements are that:
  - Anyone who reads the requirement comes to the same interpretation as any other reader.
  - Each reader's interpretation matches what the author intended to communicate.

# System or user perspective

- You can write functional requirements from the perspective of either something the system does or something the user can do.
- State requirements in a consistent fashion, such as “The system shall” or “The user shall,” followed by an action verb, followed by the observable result.
- Specify the trigger action or condition that causes the system to perform the specified behavior.
- A generic template for a requirement written from the system’ s perspective is (Mavin et al. 2009):

*[optional precondition] [optional trigger event] the system shall [expected system response]*





# Example (System Perspective)

- *If the requested chemical is found in the chemical stockroom, the system shall display a list of all containers of the chemical that are currently in the stockroom.*
  - This example includes a precondition, but not a trigger.
- Sometimes, though, it's more natural to phrase the requirement in terms of a user's action, not from the system's perspective.
- Including the "shall" and writing in the active voice makes it clear what entity is taking the action described.



# Example (User Perspective)

- When writing functional requirements from the user's perspective, the following general structure works well (Alexander and Stevens 2002):

*The [user class or actor name] shall be able to [do something] [to some object] [qualifying conditions, response time, or quality statement].*

- Alternative phrasings are “The system shall let (or allow, permit, or enable) the [a particular user class name] to [do something].”



# Example

- *The Chemist shall be able to reorder any chemical he has ordered in the past by retrieving and editing the order details.*
  - Notice how this requirement uses the name of the user class—Chemist—in place of the generic “user.”
  - Making the requirement as explicit as possible reduces the possibility of misinterpretation.

# Writing style

- Adjust your writing style to put the punch line—the statement of need or functionality—

<b>Identifier</b>	Requirement ID
<b>Title</b>	Title of requirement
<b>Requirement</b>	Description of requirement e.g <i>If written in system perspective</i> <i>[optional precondition] [optional trigger event]</i> <i>the system shall [expected system response]</i> <i>If written in user perspective</i> <i>The [user class or actor name] shall be able to</i> <i>[do something] [to some object] [qualifying</i> <i>conditions, response time, or quality</i> <i>statement].</i>
<b>Source</b>	Where requirement come from (who demand it)
<b>Rationale</b>	Motivation behind the requirement
<b>Restrictions and Risk</b>	Any restriction and risk that requirement must be fulfilled
<b>Dependencies</b>	Requirements ID that are dependent on this requirement
<b>Priority</b>	High/Medium/Low

# Template to write functional requirement

Identifier	Requirement ID
Title	Title of requirement
Requirement	Description of requirement
Source	Where requirement come from (who generate it)
Rationale	Motivation behind the requirement
Restrictions and Risk	Any restriction and risk that requirement must be fulfilled
Dependencies	Requirements ID that are dependent on this requirement
Priority	High/Medium/Low



# Some bad requirements example



# Example 1

- *The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds.*
- What are the status messages?
- Under what conditions and in what fashion are they provided to the user?
- If displayed on the screen, how long do they remain visible?
- Is it okay if they just flash up for half a second?
- The timing interval is not clear, and the word “every” just muddles the issue.



# After we get some more information from the customer:

- *The Background Task Manager (BTM) shall display status messages in a designated area of the user interface.*
- *The BTM shall update the messages every 60 plus or minus 5 seconds after background task processing begins.*
- *The messages shall remain visible continuously during background processing.*
- *The BTM shall display the percent of the background task that is completed.*
- *The BTM shall display a “Done” message when the background task is completed.*
- *The BTM shall display a message if the background task has stalled.*

## Example 2

- *Corporate project charge numbers should be validated online against the master corporate charge number list, if possible.*
- The phrase “if possible” is ambiguous.
  - Does it mean “if it's technically feasible” (a question for the developer) or
  - “if the master charge number list can be accessed at run time”?
- This requirement doesn't specify what to do when the validation either passes or fails.
- Also, avoid imprecise words such as “should.”
- Here's a revised version of this requirement:
- *At the time the requester enters a charge number, the system shall display an error message if the charge number is not in the master corporate charge number list.*

# Example 3

- *The device tester shall allow the user to easily connect additional components, including a pulse generator, a voltmeter, a capacitance meter, and custom probe cards.*
- The word “easily” implies a usability requirement, but it is neither measurable nor verifiable.
- “Including” doesn’t make it clear whether this is the complete list of external devices that must be connected to the tester.
- Perhaps there are many others that we don’t know about. *ltmeter, a capacitance meter, and custom probe cards.*
- *The device tester shall incorporate a USB port to allow the user to connect any measurement device that has a USB connection.*
- *The USB port shall be installed on the front panel to permit a trained operator to connect a measurement device in 10 seconds or less.*



# Deriving Functional Requirements from a Use case



# Recap

- Software developers don't implement business requirements or user requirements.
- They implement functional requirements, specific bits of system behavior.
- Why we need functional requirements?
- Use cases describe the user's perspective, looking at the externally visible behavior of the system.
- They don't contain all the information that a developer needs to write the software
- Example
  - The user of an ATM doesn't know about any back-end processing involved, such as communicating with the bank's computer.
  - This detail is invisible to the user, yet the developer needs to know about it.



# Continue

- **Directly derivable**
- Many functional requirements fall right out of the dialog steps between the actor and the system.
- **Derive through further analysis**
- Other functional requirements don't appear in the use case description.
- Example
  - For instance, the way use cases are typically documented does not specify what the system should do if a precondition is not satisfied.
- The BA must derive those missing requirements and communicate them to developers and testers (Wiegers 2006).

<b>ID and Name:</b>	<b>UC-1: Order a Meal</b>
<b>Primary Actor:</b>	Patron                      Secondary Actors:    Cafeteria Inventory System
<b>Description:</b>	A Patron accesses the Cafeteria Ordering System from either the corporate intranet or external Internet, views the menu for a specific date, selects food items, and places an order for a meal to be picked up in the cafeteria or delivered to a specified location within a specified 15-minute time window.
<b>Trigger:</b>	A Patron indicates that he wants to order a meal.
<b>Preconditions:</b>	PRE-1. Patron is logged into COS. PRE-2. Patron is registered for meal payments by payroll deduction.
<b>Postconditions:</b>	POST-1. Meal order is stored in COS with a status of "Accepted." POST-2. Inventory of available food items is updated to reflect items in this order. POST-3. Remaining delivery capacity for the requested time window is updated.
<b>Normal Flow:</b>	<b>1.0 Order a Single Meal</b> 1. Patron asks to view menu for a specific date. (see 1.0.E1, 1.0.E2) 2. COS displays menu of available food items and the daily special. 3. Patron selects one or more food items from menu. (see 1.1) 4. Patron indicates that meal order is complete. (see 1.2) 5. COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge. 6. Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2). 7. COS displays available delivery times for the delivery date. 8. Patron selects a delivery time and specifies the delivery location. 9. Patron specifies payment method. 10. COS confirms acceptance of the order. 11. COS sends Patron an email message confirming order details, price, and delivery instructions. 12. COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times.
<b>Alternative Flows:</b>	<b>1.1 Order multiple identical meals</b> 1. Patron requests a specified number of identical meals. (see 1.1.E1) 2. Return to step 4 of normal flow. <b>1.2 Order multiple meals</b> 1. Patron asks to order another meal. 2. Return to step 1 of normal flow.
<b>Exceptions:</b>	<b>1.0.E1 Requested date is today and current time is after today's order cutoff time</b> 1. COS informs Patron that it's too late to place an order for today. 2a. If Patron cancels the meal ordering process, then COS terminates use case. 2b. Else if Patron requests another date, then COS restarts use case. <b>1.0.E2 No delivery times left</b> 1. COS informs Patron that no delivery times are available for the meal date. 2a. If Patron cancels the meal ordering process, then COS terminates use case. 2b. Else if Patron requests to pick the order up at the cafeteria, then continue with normal flow, but skip steps 7 and 8. <b>1.1.E1 Insufficient inventory to fulfill multiple meal order</b> 1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory. 2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow. 2b. Else if Patron cancels the meal ordering process, then COS terminates use case.

# Order a Meal

Possible Functional Requirements

## Precondition

- PRE-1. Patron is logged into COS.
- PRE-2. Patron is registered for meal payments by payroll deduction.

## ➤ Order.Place:

The system shall let a Patron who is logged in to the Cafeteria Ordering System place an order for one or more meals.

## ➤ Order.Place.Register:

The system shall confirm that the Patron is registered for payroll deduction to place an order.

## ➤ Order.Place.Register.No:

If the Patron is not registered for payroll deduction, the system shall give the Patron options to register now and continue placing an order, to place an order for pickup in the cafeteria (not for delivery), or to exit from the COS.

# Order a Meal

Possible Functional Requirements

## Normal Flow

### ➤ 1.0 Order a Single Meal

- 1. Patron asks to view menu for a specific date. (see 1.0.E1, 1.0.E2)
- **1.0.E1 Requested date is today and current time is after today's order cutoff time**
- 1. COS informs Patron that it's too late to place an order for today.
- 2a. If Patron cancels the meal ordering process, then COS terminates use case.
- 2b. Else if Patron requests another date, then COS restarts use case.

### ➤ Order.Place.Date:

The system shall prompt the Patron for the meal date (see BR-8).

*BR-8: Meals must be ordered within 14 calendar days of the meal date.*

### ➤ Order.Place.Date.Cutoff:

If the meal date is the current date and the current time is after the order cutoff time, the system shall inform the patron that it's too late to place an order for today. The Patron may either change the meal date or cancel the order.

# Order a Meal

## Possible Functional Requirements

### Normal Flow

3. Patron selects one or more food items from menu. (see **1.1 of Alternative Flow**)

#### 1.1 Order multiple identical meals

1. Patron requests a specified number of identical meals. (see 1.1.E1)

##### ➤ 1.1.E1 Insufficient inventory to fulfill multiple meal order

➤ 1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory.

➤ 2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow.

➤ 2b. Else if Patron cancels the meal ordering process, then COS terminates use case..

##### ➤ Order.Units.Food:

The system shall allow the Patron to indicate the number of units of each menu item that he wishes to order.

##### ➤ Order.Units.Multiple:

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

##### ➤ Order.Units.TooMany:

If the Patron orders more units of a menu item than are presently in the cafeteria's inventory, the system shall inform the Patron of the maximum number of units of that food item that he can order.

##### ➤ Order.Units.Change:

If the available inventory cannot fulfill the number of units ordered, the Patron may change the number of units ordered, change the number of identical meals being ordered, or cancel the meal order.



# Order a Meal

## Possible Functional Requirements

### Normal Flow

- 5. COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge.
- 6. Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2 of NF).
- 7. COS displays available delivery times for the delivery date.
- 8. Patron selects a delivery time and specifies the delivery location.

### ➤ Order.Confirm.Display:

When the Patron indicates that he does not wish to order any more food items, the system shall display the food items ordered, the individual food item prices, and the payment amount, calculated per BR-12.

*BR-12: Order price is calculated as the sum of each food item price times the quantity of that food item ordered, plus applicable sales tax, plus a delivery charge if a meal is delivered outside the free delivery zone.*

### ➤ Order.Confirm.Prompt:

The system shall prompt the Patron to confirm the meal order.

### ➤ Order.Confirm.Not:

If the Patron does not confirm the meal order, the Patron may either edit or cancel the order.

### ➤ Order.Deliver.Select:

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

### ➤ Order.Deliver.Location:

If the order is to be delivered and there are still available delivery times for the meal date, the Patron shall provide a valid delivery location.



# Order a Meal

## Possible Functional Requirements

### Normal Flow

- 9. Patron specifies payment method.
- 10. COS confirms acceptance of the order.

- Order.Pay.Method:

When the Patron indicates that he is done placing orders, the system shall ask the user to select a payment method.

- Order.Pay.Pickup:

If the meal is to be picked up in the cafeteria, the system shall let the Patron choose to pay by payroll deduction or by paying cash at the time of pickup.

- Order.Pay.Details:

The system shall display the food items ordered, payment amount, payment method, and delivery instructions.

- Order.Pay.Confirm:

The Patron shall either confirm the order, request to edit the order, or request to cancel the order.

- Order.Pay.Confirm.Deduct:

If the Patron confirmed the order and selected payment by payroll deduction, the system shall issue a payment request to the Payroll System.

# Order a Meal

## Possible Functional Requirements

### Normal Flow

- 11. COS sends Patron an email message confirming order details, price, and delivery instructions.
- 12. COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times.

- Order.Done:

When the Patron has confirmed the order, the system shall do the following as a single transaction:

- Order.Done.Store:

Assign the next available meal order number to the meal and store the meal order with an initial status of "Accepted."

- Order.Done.Inventory:

Send a message to the Cafeteria Inventory System with the number of units of each food item in the order.

- Order.Done.Menu:

Update the menu for the current order's order date to reflect any items that are now out of stock in the cafeteria inventory.

- Order.Done.Times:

Update the remaining available delivery times for the date of this order.

- Order.Done.Patron:

Send an e-mail message to the Patron with the meal order and meal payment information.

- Order.Done.Cafeteria:

Send an e-mail message to the Cafeteria Staff with the meal order information.

# Introduction

- There's more to software success than just delivering the right functionality.
- Users also have expectations, often unstated, about *how well* the product will work.
- Such expectations include
  - how easy it is to use, how quickly it executes, how rarely it fails, how it handles unexpected conditions.
- Such characteristics, collectively known as *quality attributes*, *quality factors*, *quality requirements*, *quality of service requirements*, or the “-ilities,” constitute a major portion of the system's nonfunctional requirements.

# Non-Functional Requirements

# Classes of nonfunctional requirements

- Two other classes of nonfunctional requirements are constraints and external interface requirements.
- A **constraint** places restrictions on the design or implementation choices available to the developer.
  - CON-1. *The application must use Microsoft .NET framework 4.5. [architecture constraint]*
  - CON-2. *Online payments may be made only through PayPal. [design constraint]*
  - CON-3. *All textual data used by the application shall be stored in the form of XML files. [data constraint]*
- External interface provides information to ensure that the system will communicate properly with users and with external hardware or software elements.

# Importance of quality attributes

- Quality attributes can distinguish a product that merely does what it's supposed to from one that delights its users.
- Excellent products reflect an optimum balance of competing quality characteristics.
- If you don't explore the customers' quality expectations during elicitation, you're just lucky if the product satisfies them.
- Disappointed users and frustrated developers are the more typical outcome.
- Quality attributes serve as the origin of many functional requirements.
- They also drive significant architectural and design decisions



# Classification of quality attributes

- **External quality attributes** describe characteristics that are observed when the software is executing.
- They profoundly influence the user experience and the user's perception of system quality.
- **Internal quality attributes** are not directly observable during execution of the software.
- They are properties that a developer or maintainer perceives while looking at the design or code to modify it, reuse it, or move it to another platform.
- Internal attributes can indirectly affect the customer's perception of the product's quality if it later proves difficult to add new functionality or if internal inefficiencies result in performance degradation.

# Some Software Quality Attribute

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

# Certain attribute for certain projects

- Certain attributes are of particular importance on certain types of projects
- Embedded systems
  - performance, efficiency, reliability, robustness, safety, security, usability.
- Internet and corporate applications
  - availability, integrity, interoperability, performance, scalability, security, usability
- Desktop and mobile systems
  - performance, security, usability

# Exploring quality attributes

# Exploring quality attributes

- Step 1: Start with a broad taxonomy

Attribute	Score	Availability	Integrity	Performance	Reliability	Robustness	Security	Usability	Verifiability
Availability	2		^	^	^	<	^	^	<
Integrity	6			<	<	<	^	<	<
Performance	4				<	<	^	^	<
Reliability	2					<	^	^	^
Robustness	1						^	^	<
Security	7							<	<
Usability	5								<
Verifiability	1								

## Step 2: Reduce the list

- Engage a cross-section of stakeholders to assess which of the attributes are likely to be important to the project.
- An airport check-in kiosk needs to emphasize
  - Usability (because most users will encounter it infrequently) and
  - Security (because it has to handle payments).
- Attributes that don't apply to your project need not be considered further.
- Record the rationale for deciding that a particular quality attribute is either in or out of consideration.
- Recognize, though, that if you don't specify quality goals, no one should be surprised if the product doesn't exhibit the expected characteristics.



## Step 3: Prioritize the attributes

- ▶ Prioritizing the applicable attributes sets the focus for future elicitation discussions.
- ▶ Pair-wise ranking comparisons can work efficiently with a small list of items like this.
- ▶ For each cell at the intersection of two attributes, ask yourself,
  - “If I could have only one of these attributes, which would I take?”
  - Entering a less-than sign (<) in the cell indicates that the attribute in the row is more important;
  - a caret symbol (^) points to the attribute at the top of the column as being more important.

Attribute	Score	Availability	Integrity	Performance	Reliability	Robustness	Security	Usability	Verifiability
Availability	2		^	^	^	^	^	^	^
Integrity	6			<	<	<	<	<	<
Performance	4				<	<	<	<	<
Reliability	2					<	<	<	<
Robustness	1						<	<	<
Security	7							<	<
Usability	5								<
Verifiability	1								

## Step 3: Continue

- For instance, comparing availability and integrity, I conclude that integrity is more important.
- The passenger can always check in with the desk agent if the kiosk isn't operational (albeit, perhaps with a long line of fellow travelers).
- But if the kiosk doesn't reliably show the correct data, the passenger will be very unhappy.
- So I put a caret in the cell at the intersection of availability and integrity, pointing up to integrity as being the more important of the two.

Attribute	Score	Availability	Integrity	Performance	Reliability	Robustness	Security	Usability	Verifiability
Availability	2		^	^	^	<	^	^	<
Integrity	6			<	<	<	^	<	<
Performance	4				<	<	^	^	<
Reliability	2					<	^	^	^
Robustness	1						^	^	<
Security	7							<	<
Usability	5								<
Verifiability	1								

## Step 3: Continue

Attribute	Score	Availability	Integrity	Performance	Reliability	Robustness	Security	Usability	Verifiability
Availability	2		^	^	^	<	^	^	<
Integrity	6			<	<	<	^	<	<
Performance	4				<	<	^	^	<
Reliability	2					<	^	^	^
Robustness	1						^	^	<
Security	7							<	<
Usability	5								<
Verifiability	1								

- The spreadsheet calculates a relative score for each attribute, shown in the second column.
- Though the other factors are indeed important to success—
  - it's not good if the kiosk isn't available for travelers to use or if it crashes halfway through the check-in process

# Advantage of prioritization

- The prioritization step helps in two ways.
- First, it lets you focus elicitation efforts on those attributes that are most strongly aligned with project success.
- Second, it helps you know how to respond when you encounter conflicting quality requirements.
- Example
  - In the airport check-in kiosk example, elicitation would reveal a desire to achieve specific performance goals, as well as some specific security goals.
  - These two attributes can clash, because adding security layers can slow down transactions.
  - However, because the prioritization exercise revealed that security is more important (with a score of 7) than performance (with a score of 4), you should bias the resolution of any such conflicts in favor of security.

## Step 4: Elicit specific expectations for each attribute

- The comments users make during requirements elicitation supply some clues about the quality characteristics.
- The trick is to pin down just what the users are thinking when they say the software must be user-friendly, fast, reliable, or robust.
- Questions that explore the users' expectations can lead to specific quality requirements that help developers create a delightful product.

## Step 4: Continue

- Users won't know how to answer questions such as "What are your interoperability requirements?" or "How reliable does the software have to be?"
- Following are a few questions a BA might ask to understand user expectations about the performance of a system that manages applications for patents that inventors have submitted:
  1. What would be a reasonable or acceptable response time for retrieval of a typical patent application in response to a query?
  2. What would users consider an unacceptable response time for a typical query?
  3. How many simultaneous users do you expect on average?
  4. What's the maximum number of simultaneous users that you would anticipate?
  5. What times of the day, week, month, or year have much heavier usage than usual?
- Sending a list of questions like these to elicitation participants in advance
  - A good final question to ask during any such elicitation discussion is, "Is there anything I haven't asked you that we should discuss?"



# 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.
- When writing quality requirements, keep in mind the useful SMART mnemonic—make them *Specific, Measurable, Attainable, Relevant, and Time-sensitive*.
- Quality requirements need to be measurable to establish a precise agreement on expectations among the BA, the customers, and the development team.
  - If it’s not measurable, there is little point in specifying it, because you’ll never be able to determine if you’ve achieved a desired goal.
  - If a tester can’t test a requirement, it’s not good enough