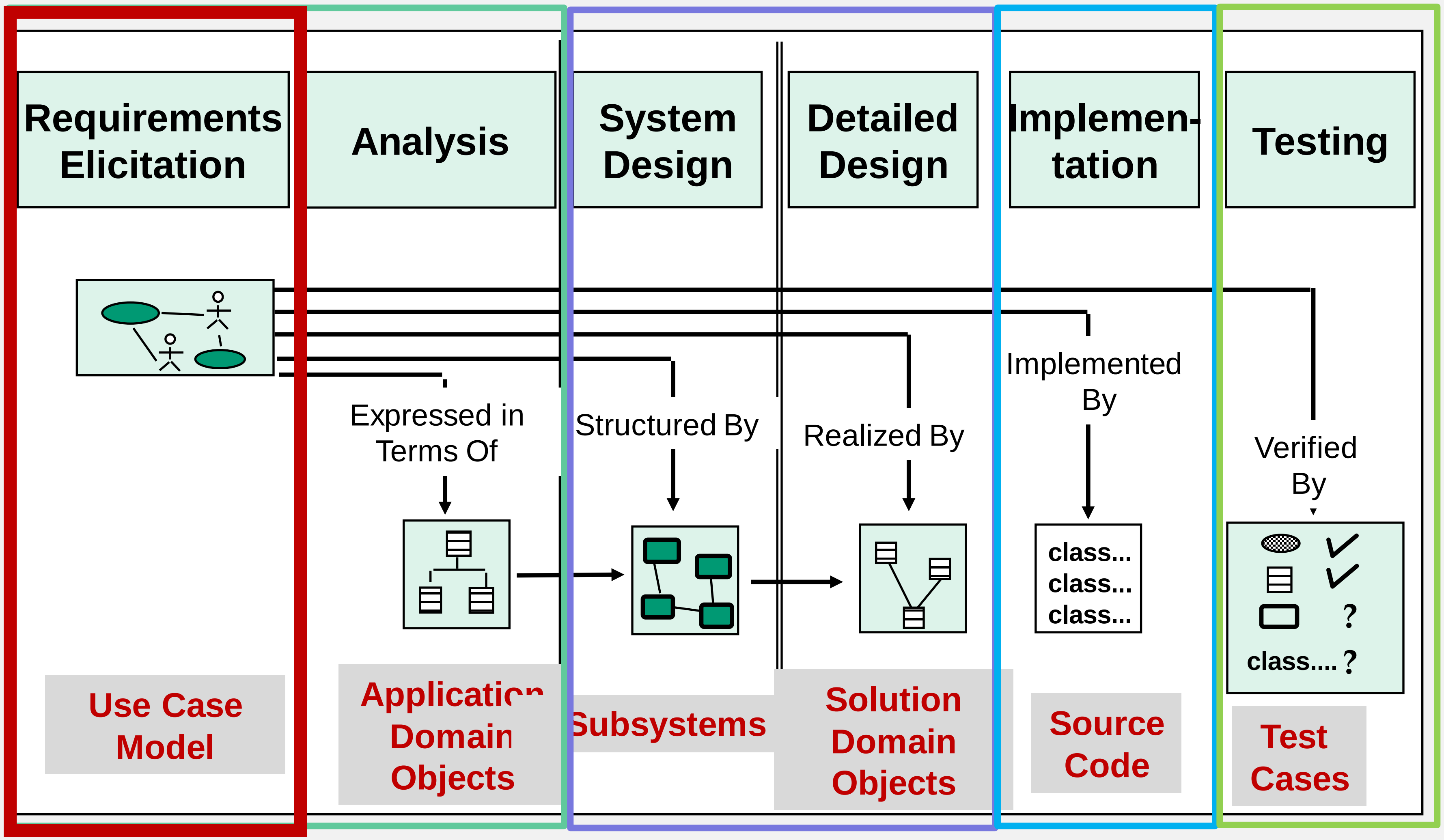
Requirement Engineering

Requirement Elicitation

Lecture 6

Software Lifecycle Activities



First step in identifying the Requirements: System identification

Two questions need to be answered:

1. How can we identify the purpose of a system?
2. What is inside, what is outside the system?

These two questions are answered during requirements elicitation and analysis

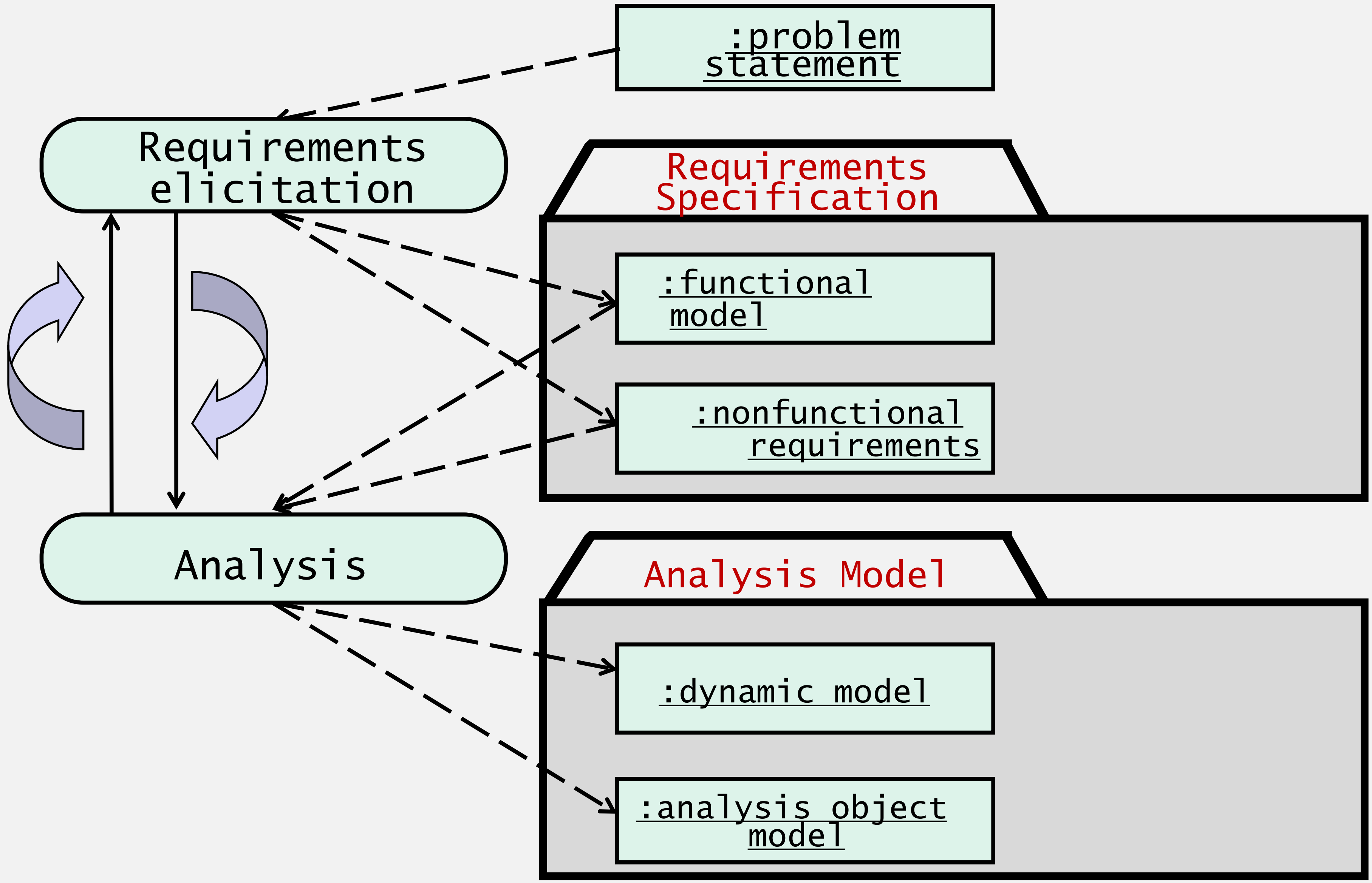
Requirements elicitation:

Definition of the system in terms understood by the customer (“Requirements specification”)

Analysis:

Definition of the system in terms understood by the developer (Technical specification, “Analysis model”)

Requirements Process



Types of Requirements

Functional requirements

Describe the interactions between the system and its environment independent from the implementation

“An operator must be able to define a new game. “

Nonfunctional requirements

Aspects not directly related to functional behavior.

“The response time must be less than 1 second”

Functional vs. Nonfunctional Requirements

Functional Requirements

Describe user tasks that the system needs to support

Interaction between the system and its environment independent of its implementation

Phrased as actions

"Advertise a new league"

"Schedule tournament"

"Notify an interest group"

Nonfunctional Requirements

Describe properties of the system or the domain

Aspects that are not directly related to the functional behavior of the system

Phrased as constraints or negative assertions

"All user inputs should be acknowledged within 1 second"

"A system crash should not result in data loss".

- Usability, reliability, performance, supportability
- implementation, interface, operation, Packaging, Legal

Requirements Elicitation Activities

1. Identifying actors
2. Identifying scenarios
3. Identifying use cases
4. Refining use cases
5. Identifying relationship among use cases
6. Identifying initial analysis objects
7. Identifying non functional requirement

Identifying Actors

Actors are external entities that interact with the system.

Actor can be human or an external system.

Actors are role abstractions and do not necessarily directly map to persons.

Actors are outside of the system boundary; they are external.

Questions for Identifying Actors

- Which user group are supported by the system to perform their work
- Which user group execute the system's main function
- Which user group perform secondary function's such as maintenance and administration
- With what external hardware or software system will the system interact

Identifying Actors

SatWatch is a wrist watch that displays the time based on its current location. SatWatch uses **GPS satellites** (Global Positioning System) to determine its location and internal data structures to convert this location into a time zone.

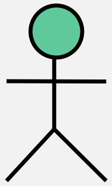
The information stored in SatWatch and its accuracy measuring time is such that the watch owner never needs to reset the time. SatWatch adjusts the time and date displayed as the watch owner crosses time zones and political boundaries. For this reason, SatWatch has no buttons or controls available to the user.

SatWatch determines its location using GPS satellites and, as such, suffers from the same limitations as all other GPS devices (e.g., inability to determine location at certain times of the day in mountainous regions). During blackout periods, SatWatch assumes that it does not cross a time zone or a politicalboundary. SatWatch corrects its time zone as soon as a blackout period ends.

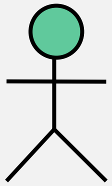
SatWatch has a two-line display showing, on the top line, the time (hour, minute, second, time zone) and on the bottom line, the date (day, date, month, year). The display technology used is such that the **watchowner** can see the time and date even under poor light conditions.

When political boundaries change, the watch owner may upgrade the software of the watch using the **WebifyWatch** device (provided with the watch) and a personal computer connected to the Internet.

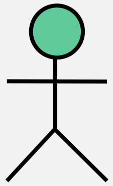
Identifying Actors



Watchowner



GPS



WebifyWatch

Identifying Actors

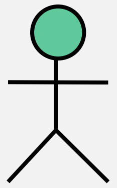
Your hairdresser salon has started a web-site for frequent customers. If you have signed up for the free membership you can login and book treatments. You can search for different employees, date, or time. The hairdressers use the system for planning, for instance, entering working hours and serving drop-in customers. The customer can erase bookings if more than 24 hours remain before the start of the treatment.

Identifying Actors

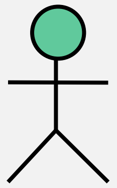
Your hairdresser salon has started a web-site for frequent customers. If you have signed up for the free membership you can login and book treatments. You can search for different employees, date, or time. The hairdressers use the system for planning, for instance, entering working hours and serving drop-in customers. The customer can erase bookings if more than 24 hours remain before the start of the treatment.

Identifying Actors

Your hairdresser salon has started a web-site for frequent customers. If you have signed up for the free membership you can login and book treatments. You can search for different employees, date, or time. The hairdressers use the system for planning, for instance, entering working hours and serving drop-in customers. The customer can erase bookings if more than 24 hours remain before the start of the treatment.



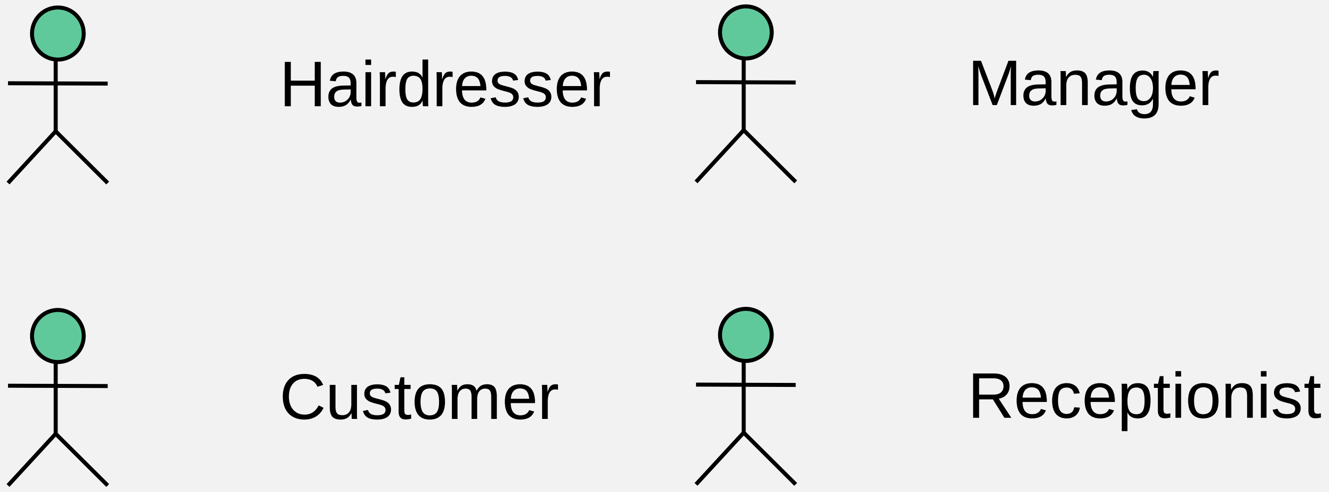
Hairdresser



Customer

Identifying Actors

Your hairdresser salon has started a web-site for frequent customers. If you have signed up for the free membership you can login and book treatments. You can search for different employees, date, or time. The hairdressers use the system for planning, for instance, entering working hours and serving drop-in customers. The customer can erase bookings if more than 24 hours remain before the start of the treatment.



Identifying Actors

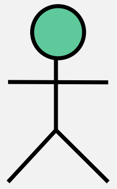
Consider normal operation of an ATM for withdrawing cash. The scenarios are like; a customer inserts the card, enters his/her PIN, enters the amount, takes the card, and takes the money. Identify the main actors and give the use-cases.

Identifying Actors

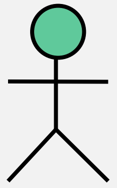
Consider normal operation of an ATM for withdrawing cash. The scenarios are like; a customer inserts the card, enters his/her PIN, enters the amount, takes the card, and takes the money. Identify the main actors and give the use-cases.

Identifying Actors

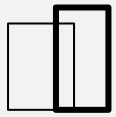
Consider normal operation of an ATM for withdrawing cash. The scenarios are like; a customer inserts the card, enters his/her PIN, enters the amount, takes the card, and takes the money. Identify the main actors and give the use-cases.



Customer



Technician



Bank

Identifying Actors

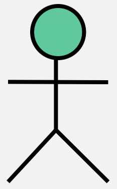
Consider a more complex example, FRIEND. A distributed information system for accident management. It includes many actors, such as FieldOfficer, who represents the police and fire officers who are responding to an incident, and Dispatcher, the police officer responsible for answering 911 calls and dispatching resources to an incident. FRIEND supports both actors by keeping track of incidents, resources, and task plans. It also has access to multiple databases, such as a hazardous materials database and emergency operations procedures. The FieldOfficer and the Dispatcher actors interact through different interfaces: FieldOfficers access FRIEND through a mobile personal assistant, Dispatchers access FRIEND through a workstation.

Identifying Actors

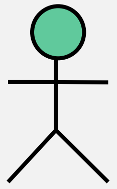
Consider a more complex example, FRIEND. A distributed information system for accident management. It includes many actors, such as FieldOfficer, who represents the police and fire officers who are responding to an incident, and Dispatcher, the police officer responsible for answering 911 calls and dispatching resources to an incident. FRIEND supports both actors by keeping track of incidents, resources, and task plans. It also has access to multiple databases, such as a hazardous materials database and emergency operations procedures. The FieldOfficer and the Dispatcher actors interact through different interfaces: FieldOfficers access FRIEND through a mobile personal assistant, Dispatchers access FRIEND through a workstation.

Identifying Actors

Consider a more complex example, FRIEND. A distributed information system for accident management. It includes many actors, such as FieldOfficer, who represents the police and fire officers who are responding to an incident, and Dispatcher, the police officer responsible for answering 911 calls and dispatching resources to an incident. FRIEND supports both actors by keeping track of incidents, resources, and task plans. It also has access to multiple databases, such as a hazardous materials database and emergency operations procedures. The FieldOfficer and the Dispatcher actors interact through different interfaces: FieldOfficers access FRIEND through a mobile personal assistant, Dispatchers access FRIEND through a workstation.



FieldOfficer



dispatcher

Heuristics for finding scenarios

Ask yourself or the client the following questions:

1. What are the primary tasks that the system needs to perform?
2. What data will the actor create, store, change, remove or add in the system?
3. What external changes does the system need to know about?
4. What changes or events will the actor of the system need to be informed about?

Finding scenarios

Example scenario

Scenario name	Check Balance	
Participating actors	Customer ATM bank	
Flow of event	1	Insert chip card
	2	Provide PIN
	3	Select service (mini statement)
	4	Take the statement
	5	Ask for more service
	6	Take the card

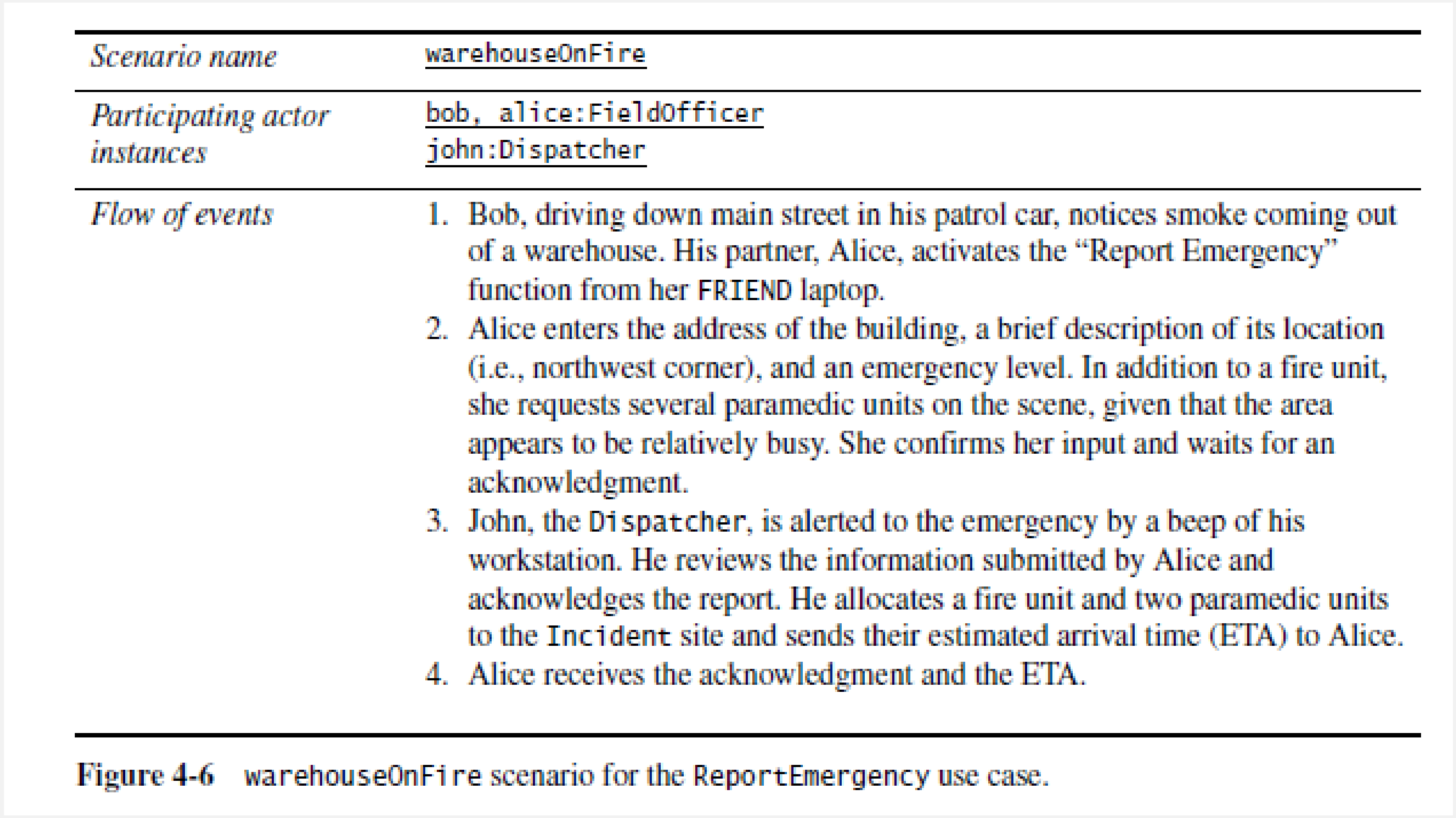
Finding scenarios

Example scenario

Scenario name	Withdraw cash	
Participating actors	Customer ATM bank	
Flow of event	1	Insert chip card
	2	Provide PIN
	3	Select amount
	4	Take the card
	5	Take the cash
	6	Take the receipt

Finding scenarios

Example scenario



Scenario name	warehouseOnFire
Participating actor instances	bob, alice:FieldOfficer john:Dispatcher
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the “Report Emergency” function from her FRIEND laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgment and the ETA.

Figure 4-6 warehouseOnFire scenario for the ReportEmergency use case.

Identifying Use Cases

A use case specifies all possible scenarios for a given piece of functionality.

A scenario is an instance of a use case

A use case is initiated by an actor. After its initiations a use case may interact with other actors.

How to write a use case

Name of Use Case

Actors

  Description of Actors involved in use case

Entry condition

  “This use case starts when…”

Flow of Events

  Free form, informal natural language

Exit condition

  “This use cases terminates when…”

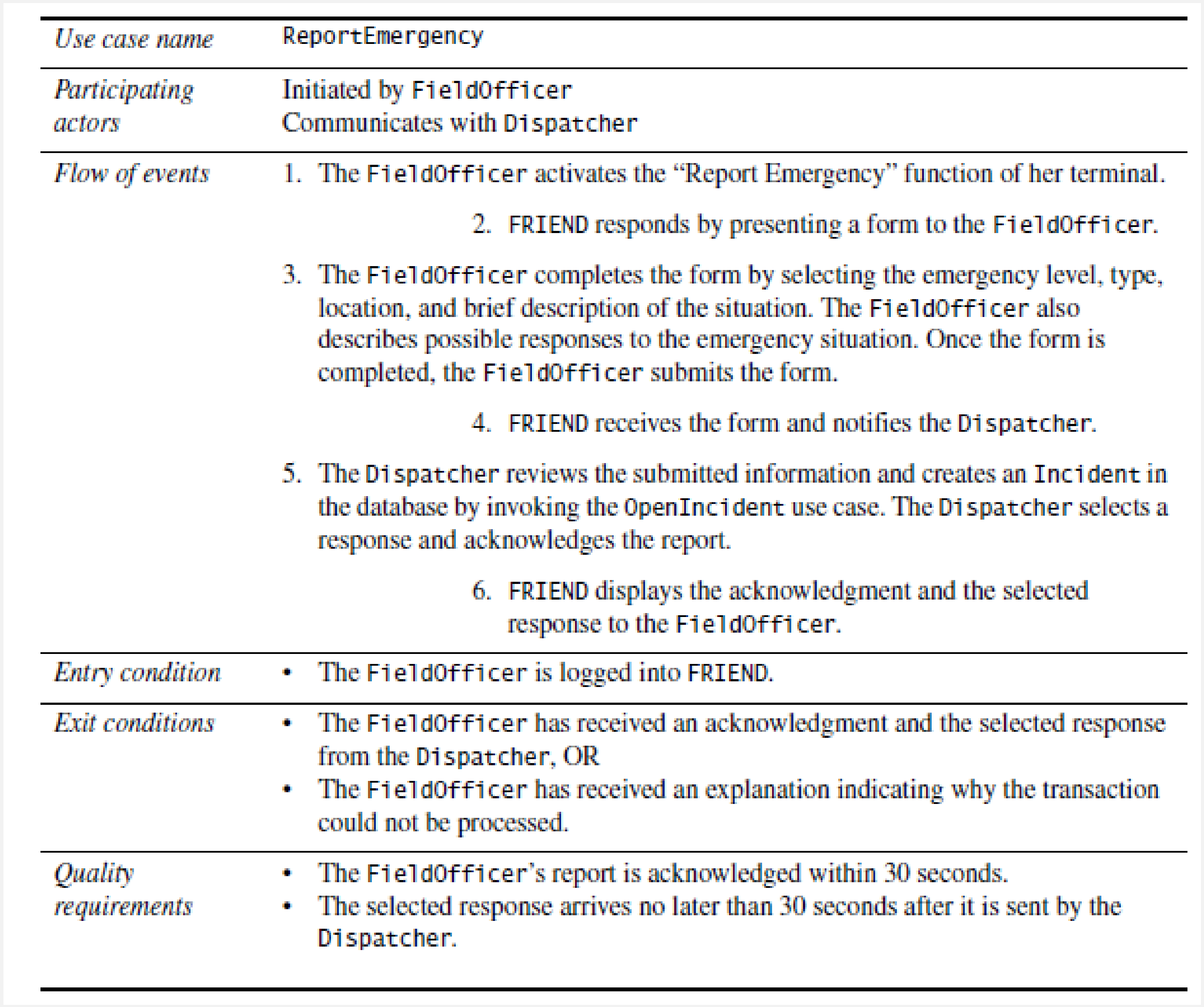
Exceptions

  Describe what happens if things go wrong

Special Requirements

  Nonfunctional Requirements, Constraints

Writing a use case



Use case name	`ReportEmergency`
Participating actors	Initiated by `FieldOfficer` Communicates with `Dispatcher`
Flow of events	1. The `FieldOfficer` activates the “Report Emergency” function of her terminal. 2. `FRIEND` responds by presenting a form to the `FieldOfficer`. 3. The `FieldOfficer` completes the form by selecting the emergency level, type, location, and brief description of the situation. The `FieldOfficer` also describes possible responses to the emergency situation. Once the form is completed, the `FieldOfficer` submits the form. 4. `FRIEND` receives the form and notifies the `Dispatcher`. 5. The `Dispatcher` reviews the submitted information and creates an `Incident` in the database by invoking the `OpenIncident` use case. The `Dispatcher` selects a response and acknowledges the report. 6. `FRIEND` displays the acknowledgment and the selected response to the `FieldOfficer`.
Entry condition	• The `FieldOfficer` is logged into `FRIEND`.
Exit conditions	• The `FieldOfficer` has received an acknowledgment and the selected response from the `Dispatcher`, OR • The `FieldOfficer` has received an explanation indicating why the transaction could not be processed.
Quality requirements	• The `FieldOfficer`’s report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the `Dispatcher`.

Writing a use case

Use Case name	Withdraw cash	
Participating actors	Customer ATM bank	
Flow of event	1	Insert chip card
	2	Provide PIN
	3	Select amount
	4	Take the card
	5	Take the cash
	6	Take the receipt
Entry condition	Inserted the card	
Exit conditioner	Received cash	
Special requirement	none	

Refining Use Case

Focus of refining is on completeness and correctness

Developers identify functionality not covered by scenarios, and document it by refining use case or writing new ones.

"The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through."

What is wrong with this use case?

Refining Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”

What is wrong with this use case?

It contains no actors

It is not clear which action triggers the ticket being issued
Because of the passive form, it is not clear who opens the gate (The driver? The computer? A gate keeper?)
It is not a complete transaction. A complete transaction would also describe the driver paying for the parking and driving out of the parking lot.

Refining Use Case

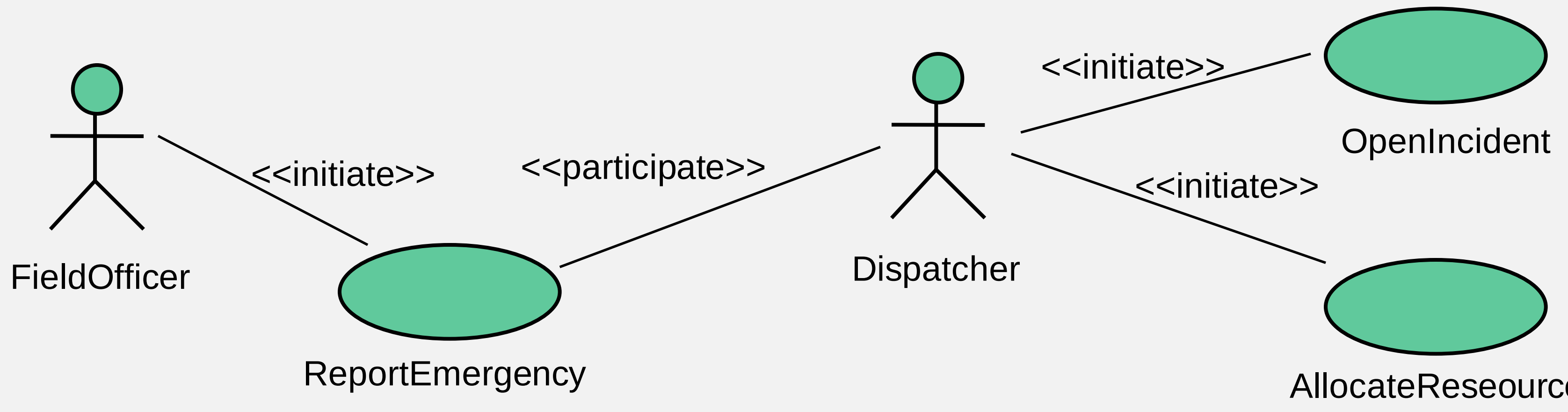
The following aspects of the use cases, initially ignored are detailed during refinement

1. The system that are manipulated by the system are detailed.
2. The low level sequence of interactions between the actor and the system are specified.
3. Access rights are specified
4. Missing exceptions are identified and their handling specified.
5. Common functionality among use cases are factored out.

Identifying relationship among Actors and Use Cases

Relationship among Actors and Use Cases enable the developers and users to reduce the complexity of the model

Communication relationship between actors and use cases represent flow of information during the use case.

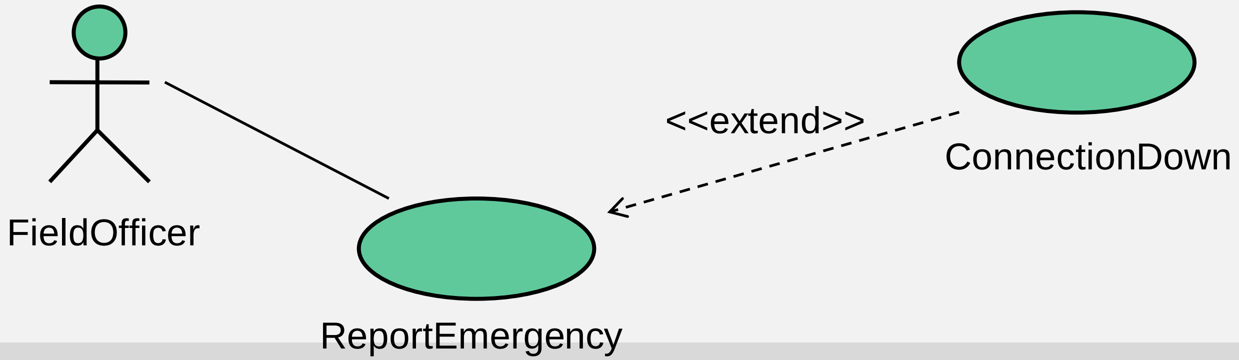


Identifying relationship among Actors and Use Cases

Relationship among Actors and Use Cases enable the developers and users to reduce the complexity of the model

- <<extend>> relationship is used to separate exceptional and common flow of events

- extended and extension both are complete use case of their own

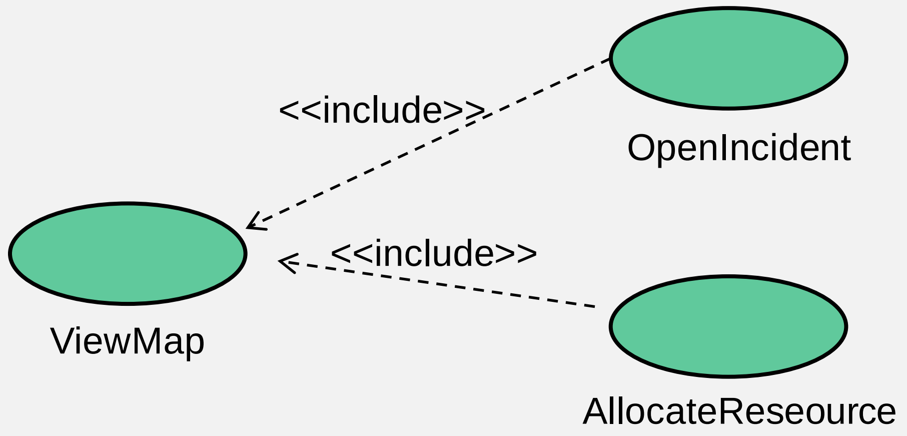


1. base use case is shorter and easier to understand
2. common case is distinguished from exceptional cases

Identifying relationship among Actors and Use Cases

Relationship among Actors and Use Cases enable the developers and users to reduce the complexity of the model

- <<include>> relationship is used to reduce redundancy among use cases



Redundancies among use cases can be factored out using include relationship.

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
- Performance
  - Response time
  - Scalability
  - Throughput
  - Availability
- Supportability
  - Adaptability
  - Maintainability

Quality requirements

Types of Nonfunctional Requirements

- Usability
- Reliability
  - Robustness
  - Safety
- Performance
  - Response time
  - Scalability
  - Throughput
  - Availability
- Supportability
  - Adaptability
  - Maintainability

Quality requirements

- Implementation
- Interface
- Operation
- Packaging
- Legal
  - Licensing
  - Certification
  - Regulation

Constraints or
Pseudo requirements

Finding Nonfunctional Requirements

Find definitions for all the nonfunctional requirements on the previous slide and learn them by heart

Understand their meaning and scope (their applicability).

Some Quality Requirements Definitions

Usability

- The ease with which actors can use a system to perform a function
- Usability is one of the most frequently misused terms ((“The system is easy to use”)
- **Usability** must be **measurable**, otherwise it is **marketing**
  - Example: Specification of the number of steps – the measure! - to perform a internet-based purchase with a web browser

Robustness: The ability of a system to maintain a function

- even if the user enters a wrong input
- even if there are changes in the environment
  - Example: The system can tolerate temperatures up to 90 C

Availability: The ratio of the expected uptime of a system to the aggregate of the expected up and down time

- Example: The system is down not more than 5 minutes per week.

Nonfunctional Requirements: Examples

"Spectators must be able to watch a match without prior registration and without prior knowledge of the match."

- *Usability Requirement*

"The system must support 10 parallel tournaments"

- *Performance Requirement*

"The operator must be able to add new games without modifications to the existing system."

- *Supportability Requirement*

What should not be in the Requirements?

- System structure, implementation technology
- Development methodology
- Development environment
- Implementation language
- Reusability

It is desirable that none of these above are constrained by the client.

Requirements Validation

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis

Correctness:

The requirements represent the client's view

Completeness:

All possible scenarios, in which the system can be used, are described

Consistency:

There are no requirements that contradict each other.

Requirements Validation (2)

Clarity:

- Requirements can only be interpreted in one way

Realism:

- Requirements can be implemented and delivered

Traceability:

- Each system behavior can be traced to a set of functional requirements

Problems with requirements validation:

- Requirements change quickly during requirements elicitation
- Inconsistencies are easily added with each change
- Tool support is needed!

We can specify Requirements for "Requirements Management"

Functional requirements:

- Store the requirements in a shared repository
- Provide multi-user access to the requirements
- Automatically create a specification document from the requirements
- Allow change management of the requirements
- Provide traceability of the requirements throughout the artifacts of the system.

Prioritizing requirements

- High priority
  - Addressed during analysis, design, and implementation
  - A high-priority feature must be demonstrated
- Medium priority
  - Addressed during analysis and design
  - Usually demonstrated in the second iteration
- Low priority
  - Addressed only during analysis
  - Illustrates how the system is going to be used in the future with not yet available technology

Thank You