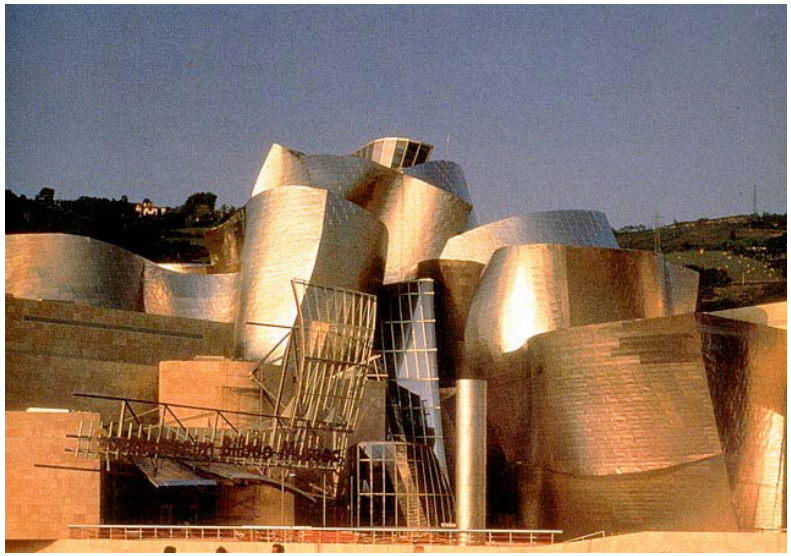
Requirements Engineering in a Nutshell

Frank O. Gehry

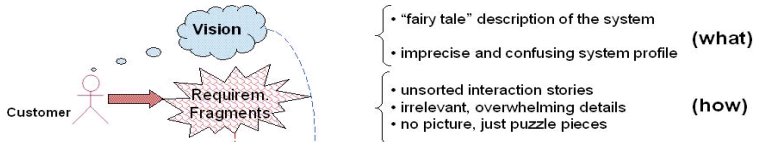


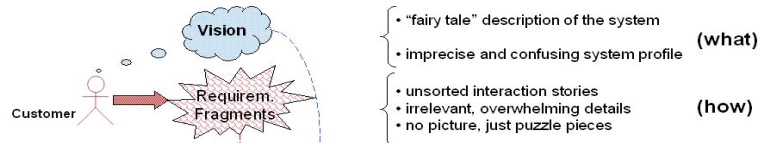
Frank O. Gehry

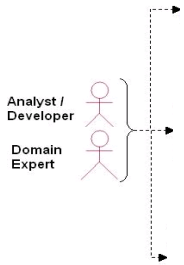
Frank O. Gehry

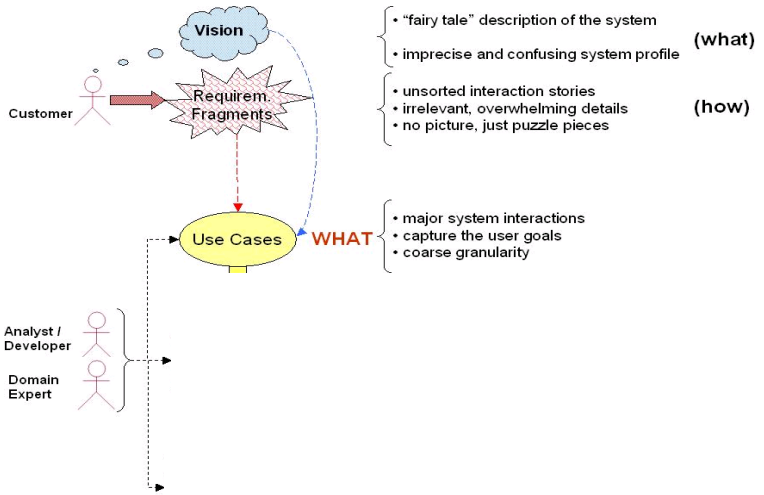
Luis von Ahn (1979)

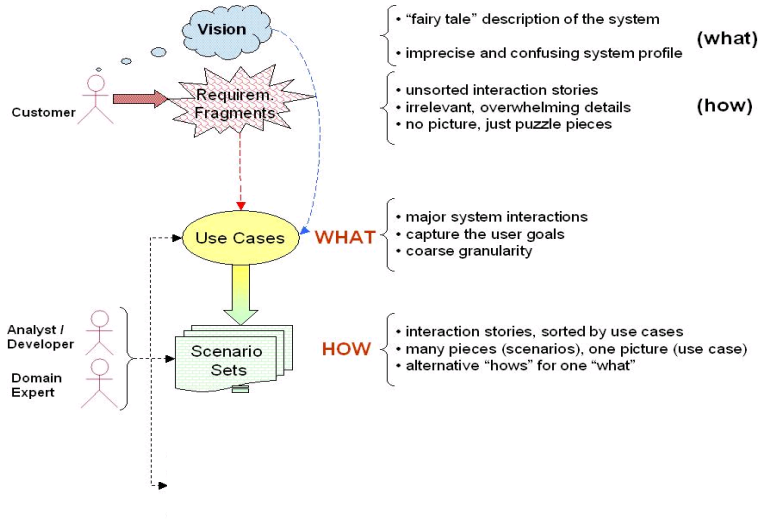
Captchas
ESP Game (label Internet images)

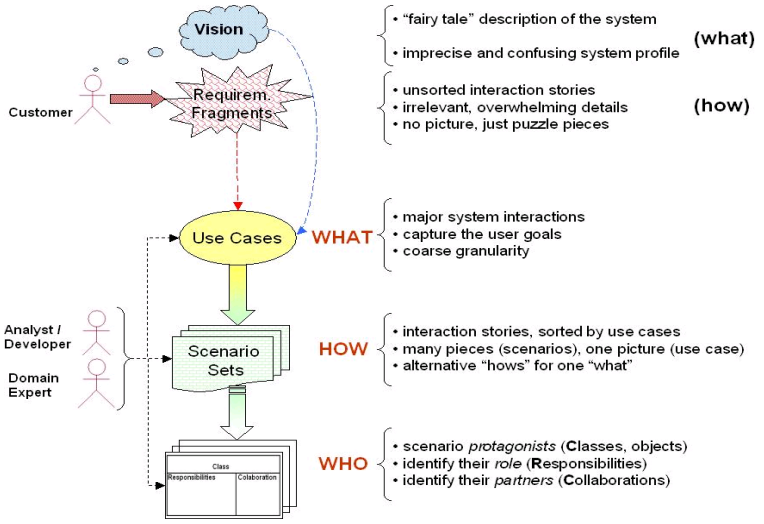












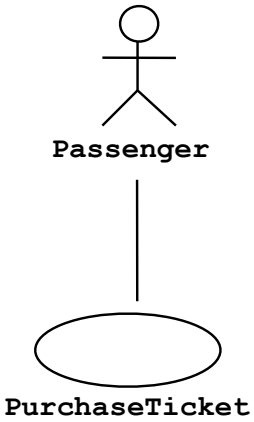
Use Cases

- create a set of **scenarios** that identify a thread of usage
  - for the system to be constructed
- scenarios are often called use-cases [Jacobson 1992]
  - how systems will be used

What is use case modeling?

- a view of a system that emphasizes the behavior as it appears to outside users.
- Partitions system functionality into transactions ('use cases') that are meaningful to users ('actors').

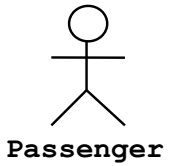
Use Case Diagrams



Used during requirements elicitation to represent external behavior

- ***Actors*** represent roles,
  - a type of user of the system
- ***Use cases*** represent a sequence of interaction for a type of functionality
- ***Use case model***
  - the set of all use cases.
  - a complete description of the functionality of the system and its environment

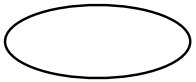
Actors



- an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- has a unique name and an optional description.
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Use Case

- a class of functionality provided by the system.



Use case consists of:

- Unique name
- Participating actors
- Entry conditions
- Flow (sequence) of events
- Exit conditions
- Special requirements

Use Case Example

Name: `Purchase ticket`

Participating actor: `Passenger`

Entry condition:

- `Passenger` standing in front of ticket distributor.
- `Passenger` has sufficient money to purchase ticket.

Exit condition:

- `Passenger` has ticket.

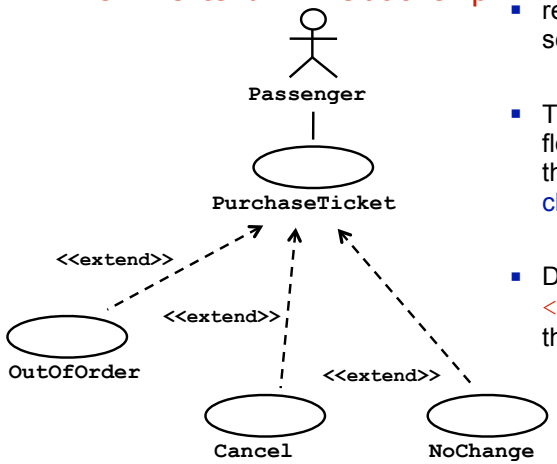
Event flow:

1. `Passenger` selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. `Passenger` inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

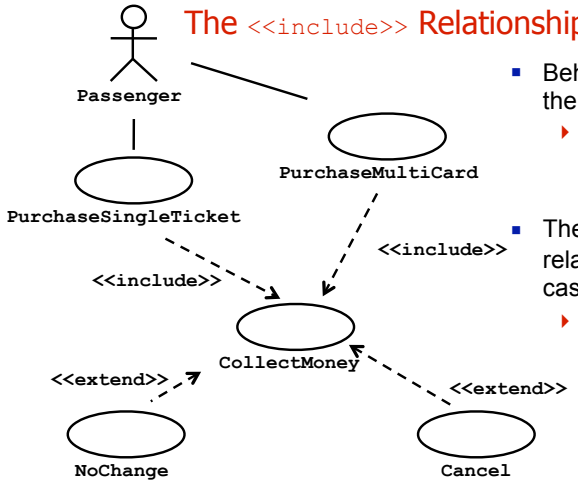
Exceptional cases!

The <<extend>> Relationship



- represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Direction of a <<extend>> relation is to the extended use case

The <<include>> Relationship



- Behavior that is **factored out** of the use case.
  - for reuse, not because it is an exception
- The direction of <<include>> relationship is to the using use case
  - unlike <<extend>>

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	UseCaseName
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	ActorName
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

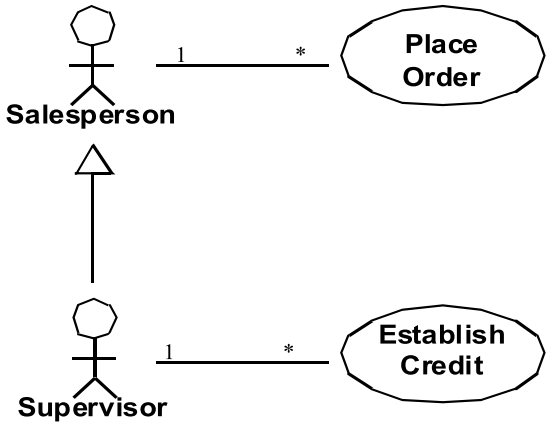
Use Case Modeling: Core Relationships

Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	
extend	A relationship from an *extension* use case to a *base* use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	<<extend>>

Use Case Modeling: Core Relationships (2)

Construct	Description	Syntax
include	An relationship from a *base* use case to an *inclusion* use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	<<include>> ---------->

Actor Relationships: Generalization



Three Levels of Abstractions for Use Cases [A.Cockburn]

- Sea Level
  - User interacts with system
  - Major interactions
  - Precise goal for using a system
- Fish Level
  - the use cases that factor out commonalities
- Kite Level (Business Use Cases)
  - Show how sea-level use cases fit into wider business interactions

Identifying Actors

- Actors are not part of the system
  - They interact with the system from outside
- An actor may
  1. Only input information to the system
  2. Only receive information from the system
  3. Input and receive information to and from the system
- Don't create an actor for every role a person can take
  - i.e. if a person fulfills a combination of roles, each role is an actor, but the combination is not!

Identifying Use Cases

A use case typically represents a major piece of functionality that is complete from beginning to end.
A use case must deliver something of value to an actor.

- Good Questions:
  - Can all functional requirements be performed by the use cases?
  - What are the tasks of each actor?

Use Case Modeling Tips

- Each use case describes a significant chunk of system usage
  - that is understandable by both domain experts and programmers
- When defining use cases in text, use nouns and verbs accurately and consistently
  - to help derive later objects and messages for interaction diagrams
- Factor out common usages that are required by multiple use cases
  - If the usage is required use <<include>>
  - If the base use case is complete and the usage may be optional, consider use <<extend>>
- A use case diagram should
  - contain only use cases at the same level of abstraction
  - include only actors who are required

Software Requirements Analysis

Software Requirements Analysis

- Build models of data and behavioral domain
  - that will be treated by software
- Result: representation of data, function
  - behavior that can be translated into design:
- Means to assess quality once the system is built

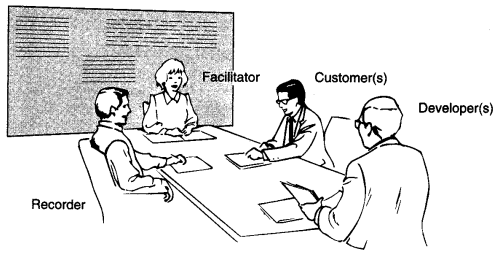
Initiate the Requirements Analysis Process

- First Meetings like ... dating :)
  - both want it to be a success, yet they don't know what to say and are afraid that it might be misinterpreted
- Ask context-free questions
  - Who requested this work? Who will use the solution? What's the benefit?
  - What is a good output? What's the environment where it will work?
- Ask meta-questions
  - Are my question relevant? Should I be asking you something else?
  - Do you feel comfortable with answering my questions? Is there someone else who could answer these questions more easy?

Facilitated Application Specification Techniques (FAST)

- Defeat the "*us and them*" mentality
  - communicate through more than memos and documents
- **Goal**:
  - to identify problem, specify a preliminary set of requirements
- **Guidelines**:
  - Meeting at neutral site with both sw. engineers and customers
  - Establish meeting rules
  - Agenda to cover important points
    - formal-informal: cover major points, yet stimulate free flow of ideas
  - Facilitator controls the meeting
    - best if it's an outsider
  - Definition mechanism
- Most well-known instance is **JAD**
  - JAD (Joint Application Design) - Chuck Morris (IBM) late '70s

Facilitated Application Specification Techniques (FAST)



from **R.S.Pressman, 2005**

FAST Process (1)

- Start meeting from a 1-2 pg. "*product request*"
  - as a result of initial discussions
- Each participant comes to the meeting with **4 lists:**
  1. List of **Objects**
     a. produced objects
     b. used objects, during processing
     c. environmental objects (surround the system)
  2. List of **Services**
     - functions and processes that manipulate the objects
  3. List of **Constraints**
     - cost, size, business rules
  4. List of **Performance Criteria**
     - speed, accuracy
- Lists don't have to be exhaustive, just reflect various p.o.v

FAST Process (2)

- Start with an agreement on the justification of the product
- Pin lists on wall
- Create a combined list on each topic area
  - eliminate redundancies
  - **don't delete anything!**
- Develop a consensus list
  - after all lists have been combined...
  - lists get shortened, lengthened, rephrased
- Divide team into sub-teams and develop mini-specifications
  - Alternative: develope Use Cases
  - an elaboration of the word or phrase representing a list item
- Joined meeting and discuss emerged changes
  - also build a *list of issues*

FAST Process (3)

- Each participant builds a list of validation criteria
  - followed by aggregating them in a consensus list of validation criteria
- Write the complete draft specification using all inputs from the FAST meeting
  - one or more participants assigned for this