Requirements Engineering

• Systematic way of developing requirements through an iterative process of analyzing a problem, documenting the resulting observations, and checking the accuracy of the understanding gained
  – Observation, interview, examining documents, JAD, questionnaire, prototypes, customer focus groups

• Results in a specification of the system that stakeholders understands
  – natural language
  – easy to understand pictures (UML Diagrams)

• A stakeholder is a key representative of the groups who have a vested interest in your system and direct or indirect influence on its requirements.
**Types of Requirements**

- **Functional requirements**: requirements that specify a function that a system or system component must be able to perform
  - The watch shall display the time.

- **Nonfunctional requirements**: not specifically concerned with the functionality of a system but place restrictions on the product being developed
  - User visible aspects of the system not directly related to functional behavior
  - Usability; reliability; privacy; security; availability; performance
  - Best to translate non-functional to measurable.
    - The response time must be less than 1 second

- **Constraints** ("Pseudo requirements"): not user-visible; imposed by the client that restricts the implementation of the system or the development process
  - The implementation language must be Java.
  - Unit tests must be written in JUnit.
Types of requirements statements

- Traditional
  - “The system shall”
- Use case based (a.k.a. iTrust)
- User story – married with acceptance test to supply the detail
Formal, Use-case, User Story

- Varying degree of formality/time to create:
  - Traditional
  - Use-case based
  - User story based
Traditional Requirements

FR2.5. Tradable Cells

When the player lands on a tradable cell, including properties, railroads, and utilities, she or he shall have a chance to buy that cell given that the cell is available and the player has enough money. If the player clicks on the Buy button, the cell shall be sold to the player and the price of the cell will be deducted from the player’s money. See FR3 for the price rules of the properties, railroads, and utilities.

Origin: Interview with Mr. Gegick on May 1, 2004 (Interview #l03SC01)
Priority: 1
Implementation Completed Date: July 15, 2004

FR2.6. Draw Card

When the player lands on a card cell, including Community Chest and Chance, she or he shall click on the Draw Card button and draw a card from the Community Chest or Chance. The player shall perform the actions specified in the cards. See FR4 for the rules of the cards.

Origin: Interview with Mr. Gegick on May 1, 2004 (Interview #l03SC01)
Priority: 1
Implementation Completed Date: July 16, 2004
Use Case-Based Requirement Elicitation

- UML focuses on scenario-based requirements elicitation
- Scenario:
  - sequence of actions that illustrates behavior. A scenario may be used to illustrate an interaction or the execution of a use case instance
Actors

- Are NOT part of the system – they represent anyone or anything that must interact with the system
  - Only input information to the system
  - Only receive information from the system
  - Both input to and receive information from the system

- Represented in UML as a stickman, even when they are not “people,” such as a billing system
Use Case

• A sequence of transactions performed by a system that yields a measurable result of values for a particular actor

• 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.

• Use cases that an actor “wants” begin with verbs.
A Case Study: Eastern State University (ESU) Registration Problem: Background

- After professors decide which courses they will teach, the Registrar enters in info in the computer
- A course catalog is printed and distributed to students
- Students fill out form with their choices – usually 4 courses
- Registrar enters this info into computer
- A batch job is run overnight to assign students to courses
- In cases of conflict where the students cannot take the classes they had selected, the registrar contacts the students directly to obtain additional choices.
- Once all students have successfully assigned to courses, a hardcopy of the schedule is sent to the student.
- Professors obtain student rosters for their classes.
Eastern State University (ESU) Registration
Problem: Problem Statement

• Professors indicate which courses they will teach on-line.
• A course catalog is printed
• Allow students to select on-line four courses (and two additional choices) for upcoming semester.
• No course may have more than 10 students (this is not NCSU) or less than 3 students.
• When the registration is completed, the system sends information to the billing system.
• Professors can obtain course rosters on-line.
• Students can add or drop classes on-line.
iTrust Spec: Use Case Based
Template for Flow of Events

X Flow of Events for the <name> Use Case

X.1 Preconditions

What needs to happen (in another use case before this use case can start?)

X.2 Main Flow

X.3 Subflows

Break “normal” flow into pieces

“called” by Main Flow or another subflow

X.4 Alternative Flows

Things that happen outside of the “normal” flow

“called” by Main Flow or a subflow

⇒ Covers multiple related scenarios!!!
Clear Intersection Example

- User wants to drive through an intersection. The user can only clear through the intersection if the traffic light is green and there are no cars in the intersection. Otherwise, the car needs to join a queue.
1. Flow of Events for the Clear Intersection Use Case

1.1 Preconditions
Traffic light has been initialized.

1.2 Main Flow
This use case begins when a car enters the intersection. The car checks its status [S-1]. The use case ends when the car clears the intersection [S-4].

1.3 Subflows

S-1 Check Status
Check status [S-2, S-3]. If the light is green, and the queue is empty, the car clears the intersection [S-4]. Otherwise, it joins a queue [S-5].

S-2 Check Light
Send information on whether the light is red, yellow, or green.

S-3 Check Queue
Send information on whether the queue is empty or not

S-4 Go
The car clears the intersection and the use case ends.

S-5 Join a Queue
Car is added to queue.
**Scenario: Car approaches intersection with green light and no queue**

1.1 Preconditions
Traffic light has been initialized.

1.2 Main Flow
This use case begins when a car enters the intersection. The car checks its status [S-1]. The use case ends when the car clears the intersection [S-4].

1.3 Subflows

**S-1 Check Status**
Check status [S-2, S-3]. If the light is green, and the queue is empty, the car clears the intersection [S-4]. Otherwise, it joins a queue [S-5].

**S-2 Check Light**
Send information on whether the light is red, yellow, or green.

**S-3 Check Queue**
Send information on whether the queue is empty or not

**S-4 Go**
The car clears the intersection and the use case ends.

**S-5 Join a Queue**
Car is added to queue.
Scenario: Car approaches intersection with red light and no queue

1.1 Preconditions
Traffic light has been initialized.

1.2 Main Flow
This use case begins when a car enters the intersection. The car checks its status [S-1]. The use case ends when the car clears the intersection [S-4].

1.3 Subflows

S-1 Check Status
Check status [S-2, S-3]. If the light is green, and the queue is empty, the car clears the intersection [S-4]. Otherwise, it joins a queue [S-5].

S-2 Check Light
Send information on whether the light is red, yellow, or green.

S-3 Check Queue
Send information on whether the queue is empty or not.

S-4 Go
The car clears the intersection and the use case ends.

S-5 Join a Queue
Car is added to queue.
1. Flow of Events for the Clear Intersection Use Case

1.1 Preconditions
Traffic light has been initialized.

1.2 Main Flow
This use case begins when a car enters the intersection. The car checks its status (S-1). The use case ends when the car clears the intersection [S-4].

1.3 Subflows
   S-1 Check Status
   Check status [S-2, S-3, E-1, E-2]. If the light is green, and the queue is empty, the car clears the intersection [S-4]. Otherwise, it joins a queue [S-5].
   
   S-2 Check Light
   Send information on whether the light is red, yellow, or green.
   
   S-3 Check Queue
   Send information on whether the queue is empty or not
   
   S-4 Go
   The car clears the intersection and the use case ends.
   
   S-5 Join a Queue
   Car is added to queue.

1.4 Alternative Flows
   E-1 Light Out
   The light is not red, yellow, or green. Wait for clear intersection and gun it.
   
   E-2 Accident
   An accident is blocking the intersection. Rubber neck and slowly drive around it.
Flow of Events vs Scenario

- Flow of events enumerates all subflows and exception flows.
- **Scenario** is one path through your flow of events.
- When you’re testing, make sure you cover a reasonable (80%??) set of scenarios.
Stereotypes

- Use Case X includes Use Case Y:
  - X has a multi-step subtask Y and subtask Y is also used by one or more other use cases. In the course of doing X or a subtask of X, Y will always be completed.

- Use Case X extends Use Case Y:
  - Y performs a sub-task and X is a similar but more specialized way of accomplishing that subtask. X only happens in an exception situation. Y can complete without X ever happening.
Story

- A story describes functionality that will be valuable to either a user or purchaser of a system or software.
  - May be called “features” in processes like FDD and Scrum
- Stories are written with the customer . . . In language of their business, not technical jargon
Card-Conversation-Confirmation

• Card
  – Stories are written on “cards”
  – The card is a token representing the requirement.
  – Notes are written on it, reflecting priority and cost.

• Conversation
  – The requirement itself is communicated from customer to programmers through conversation.
  – Best when the “customer” represents as many user types as possible
  – This conversation takes place throughout the process.
  – The conversation is largely verbal with some documents.

• Confirmation
  – At the beginning of the iteration, the customer communicates to the programmers what she wants, by telling them how she will confirm that they've done what is needed.
  – She defines the acceptance tests that will be used to show that the story has been implemented correctly.
**Example User Story**

Add a Recipe

As the owner, I would like to be able to add up to three coffee recipes.

Additional notes: A recipe consists of a name, price, units of coffee, units of milk, units of sugar, and units of chocolate. Each recipe name must be unique in the recipe list.
Setup: Five units of coffee, milk, and sugar have been added to inventory.

Operation: Jamie inserts $.75 and chooses black coffee which sells for $.50.

Verify: Jamie gets her coffee and $.25 change; coffee inventory is decreased by 1 unit, money is increased by $.50.
### Personas

**Imaginary representation of a user role.**

<table>
<thead>
<tr>
<th><strong>JAMIE</strong></th>
<th><strong>Status and Trust Level:</strong> Trusted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Role:</strong> Customer</td>
<td><strong>Demographics:</strong> Female, teenager</td>
</tr>
<tr>
<td>Knowledge, skills, and abilities: Jamie is an expert at using our machine. However, she always purchases a complicated coffee drink and sometimes doesn’t bring enough money.</td>
<td></td>
</tr>
<tr>
<td>Goals, motives, and concerns: To get coffee quickly before class begins using money found from her father’s change jar before rushing into the car.</td>
<td></td>
</tr>
<tr>
<td>Usage Patterns: Visits the vending machine each morning and chooses latte.</td>
<td></td>
</tr>
</tbody>
</table>
**Requirements Validation Criteria**

- Critical step in the development process,
  - Usually after requirements engineering or requirements analysis.
  - Also at delivery

- Requirements validation criteria:
  - **Correctness:**
    » The requirements represent the client’s view.
  - **Completeness:**
    » All possible scenarios through the system are described, including exceptional behavior by the user or the system
  - **Consistency:**
    » There are functional or nonfunctional requirements that contradict each other
  - **Clarity:**
    » There are no ambiguities in the requirements.
Requirements Validation Criteria (continued)

- Feasible:
  - Requirements can be implemented and delivered

- Traceability:
  - Each system function can be traced to a corresponding set of functional requirements

- Understandable

- Non-prescriptive
  - everything about what the customer wants and nothing about how the programmer(s) will do it.

- Consistent language
  - Shall, should, may
  - “the physician” vs. “the doctor”

- Testable