**Requirements Analysis**

**Objectives**
- Classify categories of requirements
- Define the principles of iterative requirements analysis
- Learn about use cases and their elements
- Define system sequence diagrams for representing use cases

**Requirements Analysis is Hard**
- Major causes of project failures
  - Poor user input
  - Incomplete requirements
  - Changing requirements
- Difficulties
  - Complex problems, unknown domains, non-technical customers, communication-intensive
- Essential tools
  - Classification of requirements
  - Use cases

**Classification**
- **Functional requirements**: behavior, features, capabilities
  - "The system reads employee records and prints paychecks"
- **Usability requirements**: human factors, help, documentation
  - "The font on the display should be readable from 5 feet"
  - "Do not use colors associated with common forms of color blindness"
- **Reliability requirements**: frequency of failure, recoverability
  - "If there is a failure of the external credit card authorization system, ..."
- **Performance requirements**
  - "The server response time is < 1 sec for 90% of the accesses"
  - "The system should be capable of processing 1MB of incoming transaction data each second"
- **Supportability requirements**: adaptability, internationalization, maintainability
  - "The system should allow frequent and easy changes in the network configuration"
  - "The system should be capable of incorporating several (unknown) third-party components for tax calculation"
- **Implementation requirements**
  - "Must use Linux and Java": e.g. decision due to cost, portability, availability of trained personnel
Requirements in Iterative Development

- Often 20% - 50% of the original requirements change
  - Miscommunication; changing business needs
- One of the motivations for the iterative model
- The short duration of iterations allows quick adaptation
  - Illustration: requirements in the UP

Requirements Analysis in the UP

- Major artifacts: Use-Case Model and Supplementary Specification
  - Use-Case Model: functional requirements
    - A set of use cases
      - use case = "story" of using the system
  - Supplementary Specification: non-functional requirements
    - performance, reliability, usability, supportability, implementation, etc.

Effort Distribution

Possible Timeline

- Inception, 1 week
  - Use cases are identified and described briefly: 10% are described in detail
- Elaboration, iteration #1, 4 weeks
  - Design a small set of architectural and high-risk requirements; implement and test them
  - End of the iteration: 2 day meeting on the use cases -> 30% are described in detail

Possible Timeline

- Elaboration, iteration #2, 4 weeks
  - Design, implementation, testing of high-risk and architectural requirements
    - 5% of the code is built
    - At the end: detailed description of 50% of the use cases
- Elaboration, iteration #3, 3 weeks
  - More design, implementation, testing: 10% of the code is built
  - 70% of the use cases - fully described

Possible Timeline

- Elaboration, iteration #4, 3 weeks
  - Design, implementation, testing
    - High-risk and architecturally significant aspects should be stabilized
    - Total of 15% of the final system is built
    - 80-90% of the use cases: clarified and written in detail
- Construction
  - Very little work on the use cases
  - Design/coding for the rest of the system
The Role of Use Cases

- Currently, the most widely used approach for capturing requirements
- Central role in the Unified Process
  - Requirements are primarily discovered and recorded through use cases
  - All other activities are driven by the use cases (domain modeling, design, etc.)
- Running example: point-of-sale (POS) system [Larman]

Example of a Use Case

Process Sale: A customer arrives at checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system.

Stakeholders and Goals

- Stakeholders: customer, cashier, company, tax agencies, credit card company
- A use case is a story of using the system to fulfill stakeholder goals
  - Better than a "laundry list" of features

Some Definitions

- Actor: something with behavior – e.g. person, computer system, organization
- Scenario: a specific sequence of interactions between the actors and the system under discussion
  - Success scenario or failure scenario
- Use case: collection of related success and failure scenarios

Scenarios

Handle Returns use case

- Main success scenario: A customer arrives with items to return. The cashier uses the POS system to record ....
- Alternative Scenarios
  - If they paid by credit but reimbursement to their credit account is rejected, pay by cash
  - If the system detects a failure in the external accounting system, ....
  - ............

Black-Box Use Cases

- Black-box use cases: do not describe the internal workings of the system
  - Only system responsibilities
  - Goal: specify what the systems should do without deciding how it will do it
- Good: "The system records the sale"
- Bad: "The system writes the sale to a database" or "The system generates SQL INSERT statement for the sale"
Levels of Formality

- **Brief**: one-paragraph, for the main success scenario
  - Process Sale example was brief
- **Casual**: multiple paragraphs that cover several scenarios
  - Handle Return example was casual
- **Fully dressed**: all steps and variations are written down
  - Developed iteratively during elaboration; the product of requirements analysis

Fully Dressed Use Case

- **Primary actor**: principal actor that interacts with the system
  - ProcessSale: the cashier
- **Preconditions and postconditions**
- **Main success scenario**
- **Alternative scenarios**
- **Several other kinds of information**
  - Excellent example in [Larman], Section 6.6

Preconditions and Postconditions

- **Preconditions**: guaranteed to be true before the use case
  - e.g. “Cashier is authenticated”
  - Often the postconditions of a successful scenario from another use case
- **Postconditions**: what must be true after a successful completion of the use case
  - Main scenario and successful alternative scenarios

Scenarios

- **Main success scenario**: the “happy path”
  - No conditions or branching
- **Alternative scenarios (Extensions)**
  - Both successes and failures
  - The Extensions part is typically much longer than the Main Success Scenario part

Example: Process Sale, Cash Only

- **Primary actor**: Cashier
- **Precondition**: Cashier is authenticated
- **Postconditions**
  - Sale is logged, accounting and inventory are updated, receipt is generated
- **Main success scenario**: sequence of interactions with actors

Main Success Scenario

1. Customer arrives with goods
2. Cashier starts a new sale
3. Cashier enters item id
4. System records sale line item and presents description and running total
   - Repeat 3-4 until Cashier indicates “done”
5. System presents total with taxes. To determine taxes, System uses an external Tax Calculator system
Main Success Scenario (cont)

6. Cashier asks Customer for payment
7. Cashier enters cash amount tendered, and System presents change due
8. System presents receipt
9. System logs completed sale and sends sale info to the external Accounting system and to the external Inventory system

Summary

- Several actors: Cashier, Tax Calculator, Accounting, Inventory
  - Customer could be considered an actor, but has no direct interaction with the system
- Black-box point of view
  - "System logs sale" vs. "System writes sale to a file"
- Simplified interactions: no conditions, branching, etc.
- Understandable by non-technical people

System Sequence Diagram (SSD)

- Describes in more detail a scenario in a use case
  - Created from the text of the use case
- A kind of UML sequence diagram
  - SSD is a simplified version useful for requirements analysis
  - More general version of sequence diagrams: later, when talking about design
- Example: ProcessSale for POS system

Partial SSD for the Main Scenario

<table>
<thead>
<tr>
<th>Cashier</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>makeNewSale()</td>
<td></td>
</tr>
<tr>
<td>enterItem(itemID)</td>
<td></td>
</tr>
<tr>
<td>description, total</td>
<td></td>
</tr>
<tr>
<td>*(more items)</td>
<td></td>
</tr>
<tr>
<td>endSale()</td>
<td></td>
</tr>
</tbody>
</table>

SSD for “Process Sale”

2. Cashier starts a new sale
3. Cashier enters item id
4. System records sale line item and presents description and running total

Repeat 3-4 until Cashier indicates “done”
SSD for "Process Sale" (cont)

5. System presents total with taxes. To determine taxes, System uses an external Tax Calculator.

```plaintext
:Cashier
endSale()

:System
getTaxes(sale)
total with taxes

taxes

:TaxCalculator
```

SSD for "Process Sale" (cont)

7. Cashier enters cash amount tendered, and System presents change due.
8. System presents receipt.
9. System logs completed sale and sends sale info to the external Accounting system and to the external Inventory system.

```plaintext
:Cashier
:System
:Accounting
:Inventory
makePayment(amount)
change, receipt
postSale(sale)
update(sale)
```

Abstractions

- Events and return values are abstractions
  - Independent of mechanism & representation
- `makePayment(amount)`
  - Looks like a method call; shows input info
  - Not a method call: abstraction of an event
- Name: should capture the intent
  - Avoid implementation choices
    - `enterItem(itemID)` is better than `scan(itemID)`

Timeline for SSDs

- SSDs are created during elaboration
  - Clarify the major events that the system should be able to handle
  - Later we design objects to handle these events (object-oriented design)
- SSDs are created for some chosen scenarios from the current iteration
  - Happy path + frequent/complex alternatives

Alternative Scenarios

In the main scenario:

3. Cashier enters item identifier

In the Extensions part of the use case:

3a. Invalid identifier:
   1. System signals error and rejects entry
   - Condition: triggers this alternative
   - Handling: one or more steps
Alternative Scenarios

In the main scenario:
6. Cashier asks customer for payment

In the Extensions part of the use case:
6a. Customer doesn’t have enough cash
1. Cashier cancels sale on System

An example of a failure scenario

Another Example

In the main scenario:
5. System presents total with taxes. To determine taxes, it uses a TaxCalculator

Alternative: What if some customers are entitled to a discount? - e.g. employees
- Suppose that each such customer has a customer_id that determines discount %
- ids and discount % are stored in an external CustomerInfo system

Another Example

In the main scenario:
5. System presents total with taxes ...

In the Extensions part of the use case:
5a. Customer is eligible for discount
1. Cashier signals discount
2. Cashier enters customer id
3. System presents updated total. System uses external CustomerInfo system to get discount percentage

SSD for this scenario

:Cashier
:total with taxes
:startDiscount()
:enterCustomerId(id)
:updated total
:makePayment(amount)

:CustomerInfo
:getDiscount(id)
:discount percentage

Additional Info in a Use Case

- Non-functional requirements
  - Usability: “The text on the screen should be visible from 5 feet”
  - Performance: “Credit authorization response should be within 30 sec, 90% of the time”
  - Technology: “Identifier entered by laser scanner or by keyboard”
- Eventually gathered in the Supplementary Specification

UML Use Case Diagram
UML Use Case Diagram

- UML notation for representing actors, use cases, and their relationships
- The diagram is secondary to the actual use cases: need to focus on text
  - Use-Case Model = text of the use cases
- In this class, the use case diagram is shown only for completeness

Constructing the Use Cases

- Iteratively refined during inception and elaboration
- Communication-intensive process
  - Developers need to talk with domain experts
    - e.g. Extreme Programming requires a user to be co-located full-time with the development team
  - Focus on user intentions and goals

Timeline

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Incep</th>
<th>Elab</th>
<th>Const</th>
<th>Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-Case Model</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suppem. Spec</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain Model</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Model</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implem. Model</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Requirements analysis: Use-Case Model + Supplementary Specification

Domain analysis: Domain Model
Design: Design Model
Coding: Implementation Model

Supplementary Specification

- Describes other requirements
  - In addition to the functional requirements described by the use cases
- Functional requirements common across many use cases
  - Logging and error handling: e.g. “Log all errors to persistent storage”
  - Security: e.g. “All usage requires user authentication”
  - Usability requirements
    - “Avoid colors associated with common forms of color blindness”
    - “The cashier is often looking at the customer, so signals and warnings should be conveyed with sound”
  - Reliability requirements
    - Recoverability: “If there is a failure to use external services (e.g. accounting system), store the information locally to complete the sale”
  - Performance requirements
    - “External payment authorization should be completed within 30 seconds, in 90% of the cases”
  - Supportability requirements
    - Adaptability: “At certain points in the scenarios, pluggable rules should be enabled to accommodate different customers”
    - Configurability: “Different customers will have different network configurations”
**Supplementary Specification**

- **Implementation constraints**
  - "Management insists on Java for long-term portability, supportability, and ease of development"

- **Requirements for purchased components**
  - The tax calculator will be purchased from a third party. The system must support pluggable tax calculators for different countries

- And many more: business rules, legal issues, standards, I/O devices, tools, ...

**The Role of the Supplementary Spec**

- Describes issues that are not easily captured by use cases
- Particularly important are system-wide attributes
  - Performance, reliability, testability, etc.
  - Central role during early design and implementation
  - E.g., a system-wide requirement for high fault tolerance has very significant influence on large-scale design decisions

**UP Timeline**

- Inception: the supplementary specification is only lightly developed
  - Focus on risky system-wide requirements

- Elaboration: continuous refinement
  - In parallel with early design/implementation
  - Feedback for the requirements analysis
  - Stabilized at the end of elaboration