Announcements

- The midterm exam for CS445 will be Thurs 16 Feb 2006 8:30-9:50am in our regular classroom.

- All project documentation is now available on the course web page.

- Send your group to cs445@student.cs.uwaterloo.ca by Friday, 13 Jan 2006 2:00pm.

- Grad students need to have their topics chosen and approved by Friday, 13 Jan 2006.
Review

Tasks of Requirements Analyst:

1. Examine project viability

2. Understand problem from each stakeholder’s point of view

3. Extract the essence of the stakeholders’ requirements

4. Invent better ways to do the user’s work

5. Negotiate a consistent set of requirements with agreement from all stakeholders; set relative priorities

6. Record results in an SRS
Review: Stakeholders

Stakeholder = person “needed to ensure the success of a project” (Lauesen p. 35)

- Client
- Customer
- Users
- Domain Experts
- Software Engineer
- Inspectors
- Market Researchers
- Lawyers
- Experts on Adjacent Systems
- Value-adders
Review: Elicitation Techniques

[Lauesen, 8.2]

- Stakeholder analysis
- Interviews
- Observation
- Task Demo
- Document studies
- Questionnaires
- Brainstorm
- Focus groups
- Domain workshop
- Design workshop
- Prototyping
- Pilot experiments
- Similar companies
- Ask suppliers
- Negotiation
- Risk analysis
- Cost/benefit
- Norms
Review

The best way to avoid change requests and cost over-runs is to

- have a complete list of stakeholders,
- have a complete list of requirements from each stakeholder, and
- ensure that the lists are consistent with one another.
Today’s Agenda

- Use Cases
  - What is a “use case”? 
  - What is a “scenario”? 
  - Use case descriptions 
  - Use case diagrams 
  - Finding use cases 
  - Advanced use case modelling 
  - Rules of thumb

Required Reading: Arlow and Neustadt Ch. 4,5

Optional References:

- D. Kulak and E. Guiney, *Use Cases: Requirements in Context*, Addison-Wesley, 2004, Second Edition. (This is a good place to look for examples.)
What are use cases?

- “stories of the system”
- high-level descriptions of the system’s functionality and its environment
- “cases of use”
- describe how the system meets user goals
- way of doing “user-centered analysis”
- “first cut at the functionality of an application”
- “a sequence of related messages exchanged among the system and outside actors, together with actions performed by the system” (Lauesen, p. 126)
- black box description of the system – it’s all about the interactions (Kulak and Guiney, p. 25)
What are use cases?

“A use case represents a series of interactions between an outside entity and the system, which ends by providing business value.” (Kulak and Guiney, p. 35)

Use cases describe functionality.

Use cases were developed by Ivar Jacobson in 1986 and are part of the Rational Unified Process Development Model.

Use cases are not exclusively for object-oriented analysis.

The set of use cases need not be complete in that it documents all the requirements, rather the goal is to understand the desired behaviour of the system.
Example

Name: Buying a Book Online
Use Case Number: UC32
Authors: John Doe
Event/Precondition: Customer requests to buy one or more books. The choice of books is passed as the input.
System: Customer and Vendor computers with a Web application that implements online book selling

Actors:
- Customer (initiator)
- Credit-card authorisation service
- Bookseller

Overview/Postcondition: This use case captures the process of purchasing one or more books from an online book seller.

References: R23, R34, and R45.
Related Use Cases: UC11
## Example (con’d)

### Typical Process Description

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Customer submits a selection of books he wants to buy.</td>
<td>2. System checks if the customer has already identified himself. If customer is not identified, see UC11 (Shopping Cart Set Up).</td>
</tr>
<tr>
<td></td>
<td>3. System adds books to the Shopping Cart.</td>
</tr>
<tr>
<td></td>
<td>4. System checks the availability of items.</td>
</tr>
<tr>
<td></td>
<td>5. System prompts the customer for the payment type.</td>
</tr>
<tr>
<td></td>
<td>7. If payment type is “credit card payment”, see Section Credit Card Payment. If payment type is “cheque payment”, see Section Cheque Payment.</td>
</tr>
<tr>
<td>6. Customer chooses payment type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23. System sends a confirmation message to the customer that the books have been shipped.</td>
</tr>
</tbody>
</table>
### Example (con’d)

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Responsibility</th>
</tr>
</thead>
</table>

**Alternative 1:**
6. Customer chooses to cancel the sale.

**Exception 1:**

**Section Credit Card Payment:**
1. Customer submits credit card number.

3. Credit Card Authorisation Service sends authorization.

**Exception 1:**
2. System cannot connect to the Credit Card Authorisation Service.

⋯

7. ...

2. System sends credit card information to the Credit Card Authorisation Service.
“An actor specifies a role that some external entity adopts when interacting with your system directly.” (Lauesen, p. 71)

- people in certain roles
- other systems/things that use the system
- an entity with behaviour

Kinds of actors:

- **primary**: users with goals; could be another system monitoring the SuD (system under discussion); trigger the use cases
- **supporting**: provide services to the SuD; interact with the SuD after the use case has been triggered

Things/People may play multiple roles simultaneously and over time.
Actors

Choose a short descriptive name for the actors (e.g., avoid “User”).

Use of actor’s name should always be capitalised.

A use case is always started by a single actor.

“Time” may be an actor of your system.

Time is the actor for events that happen to the system at a specific point in time but aren’t triggered by an existing actor.
Observability

Use cases describe observable (i.e., externally visible) behaviour, meaning:

- Inputs from the actors
- Outputs to the actors
- Steps of the system that result in observable behaviour, i.e., steps that affect the outputs that the actor sees at some point. These steps might be:
  - **directly observable**: affect the outputs in this use case (provides value to an actor in this use case)
  - **indirectly observable**: affect the outputs in another use case (provides value to an actor participating in another use case)
Scenarios

- A scenario is a specific sequence of actions and interactions between actors and the SuD. A scenario is also called a use case instance.

- A use case is a collection of related success and failure scenarios that describe actors using a system to support a goal (an observable result of value to a particular actor).

A scenario is one path through a use case.

These terms are used with different meanings elsewhere.
Use Case Descriptions

There are a variety of use case description formats. For example:

- **Brief**: only contain main scenario
- **Casual**: paragraph format; may cover multiple scenarios
- **Fully Dressed**: sequence of interactions written in column format with sections such as preconditions and cross-referencing

Your text show different formats. In your project, you should write “fully dressed” use cases, but you may use the format you find most suitable as long as your customer agrees with you.
Use Case Descriptions

- **Use case number:** a unique number for referencing UC elsewhere in the specification; use cases are numbered “UC1”, “UC2”, etc.

- **Name:** a short, descriptive name indicating what is captured by UC
  - names should start with verbs

- **Authors:** the names of the people who discovered the use case
Names

Use of weak verbs in use case names may indicate that you don’t really know the purpose of the use case.

Examples:

<table>
<thead>
<tr>
<th>Strong Verbs</th>
<th>Weak Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Make</td>
</tr>
<tr>
<td>Remove</td>
<td>Use</td>
</tr>
<tr>
<td>Merge</td>
<td>Copy</td>
</tr>
<tr>
<td>Calculate</td>
<td>Record</td>
</tr>
<tr>
<td>Enter</td>
<td>Process</td>
</tr>
<tr>
<td>View</td>
<td>List</td>
</tr>
<tr>
<td>Search</td>
<td>Input</td>
</tr>
</tbody>
</table>
Names

Use of weak nouns in use case names may indicate that you don’t really know the purpose of the use case.

Examples:

<table>
<thead>
<tr>
<th>Strong Nouns</th>
<th>Weak Nouns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loan</td>
<td>Data</td>
</tr>
<tr>
<td>Price</td>
<td>Report</td>
</tr>
<tr>
<td>Account</td>
<td>System</td>
</tr>
<tr>
<td>Date</td>
<td>Form</td>
</tr>
<tr>
<td>Sales</td>
<td>Template</td>
</tr>
</tbody>
</table>
Use Case Descriptions (con’d)

- **Event/Precondition:** the description of the event that initiates UC; indicate information that is passed as input with the event
  - a use case should be triggered by a single event
  - preconditions are constraints on the state of the system that are assumed to be true before beginning a scenario (not tested in the scenario)

- **System:** a declaration of what you consider to be the system for UC
  - business (interaction with business), system (interaction with software)
Use Case Descriptions (con’d)

- **Actors:** a list of the actors that participate in UC, giving UC’s initiator as the first element of the list

- **Overview/Postconditions:** a brief 2-3 sentence description of UC; this overview serves also as a high-level description of UC. Describe what should be true on successful completion of the use case.

- **References:** a list of the numbers of all requirements captured by UC (See the SRS tutorial.)

- **Related Use Cases:** a list of the numbers of all related use cases; for each element of the list, describe the relationship of the identified use case to UC
Use Case Description (con’d)

- **Typical Process Description**: in a multi-column (or single column) format, a description of the most usual scenario of the use case, the so-called normal time-ordered interaction of actors and the system that leads to the successful outcome of the process that this use case captures (“happy day”/ “perfect world”). This is also called the main scenario or basic flow.

- one column for each actor or process that is visible at the user’s level.

- sometimes, there will be only two columns, at least at the highest level view of the system, (1) the actor(s) and (2) the system itself.
Use Case Description (con’d)

- Typical Process Description: (con’d)
  - In the left-most column, first row, the initiator’s actions are listed.
  - In each of the remaining rows, the reactions by one of the system’s processes to the initiator’s or other actor’s actions are listed in the appropriate column.
  - Typical actions:
    - interaction between actor and system (input/output)
    - validation by system
    - state change in the system (e.g., record some information)
  - Use active voice to describe the steps.
  - System cannot initiate a use case
Use Case Description (con’d)

- Typical Process Description: (con’d)
  - Indicate branches on certain conditions (e.g., “see Section Credit Card Payment”). Branch may refer to another use case described elsewhere or subsections of this use case. **Branches must be based on conditions that the system or an actor can detect.**
  - Subsections describe actions on branches.
  - Subsections are assumed to merge back with the main flow, unless they indicate otherwise.
Use Case Description (con’d)

- **Alternative Flows:** subsections for different actions that an actor can take in the main scenario. Start the line numbers at the point where the alternative flow diverges from the main scenario.

  Or you may begin the alternative flow “This alternative flow begins at any time” if it does not begin at a particular step in the main scenario.

  Alternatives are assumed to merge back into the main flow when they are complete.

- **Exceptions:** subsection for alternative behaviours of the system based on certain errors. These frequently do not return to the main flow.
Use Case Description (con’d)

Alternative Flows and Exceptions should rarely have alternative flows themselves.

Be careful to make it clear to which scenario (main or subsection) alternative flows or exceptions belong.

Almost every step can fail in some way. Limit the alternative flows or exceptions you include to those that are the most important!
More Complex Actions

- **For** (iteration expression)
- **While** (Boolean condition)

Indent underneath these.

**Example** [Arlow and Neustadt p. 85]:

3. **While** the Customer is browsing the company details
   - The system plays some background music.
   - The system displays special offers in a banner ad.

These are not needed very often and may be a sign that the use case is becoming too detailed or too much like pseudo-code.
Use Case Descriptions

- There may also be other fields in use case descriptions (such as non-functional requirements).
What level of description belongs in a use case?

Larman recommends focusing at the level of “elementary business processes” (EBPs):

- “a task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state.”
- should capture a user’s goal (intention)

A use case is written from the point of view of the actors.

Ivar Jacobson has said that even the largest system should not have more than 100 use cases.
Use Case Diagrams

- Describes the set of all use cases graphically
- Shows relationships (communication) between use cases and actors
- Model of the system’s functionality and environment

Actors can be shown as stick people or as a class icon stereotyped actor. Suggestion: use stick people for humans, and the class icon for non-human actors.
Use Case Diagrams

- Primary actors on the left; supporting actors on the right
- System is drawn as a box
- Use cases are drawn as ovals with labels
- Relationships between actors and use cases they participate in
Finding Use Cases

1. Find a candidate system boundary

2. Identify primary actors.
   - Who or what uses the system?
   - What roles do they play?
   - Who installs the system?
   - Who or what starts and shuts the system?
   - Who maintains the system?
   - Who or what gets and provides information to the system?
   - Does anything happen at a fixed time?

May need to generalize from specific people/things.
Finding Use Cases

3. For each actor, find their goals (EBPs). The actor is the “initiating actor” for this goal. There maybe other actors for the same goal.

4. Identify the input event for each goal. Identify any preconditions.
Finding Use Cases

5. Define a use case for each goal.
   - Think about concrete cases, maybe specific inputs, first then generalise (possibly).
   - One exception is that we can group the similar goals of “create”, “retrieve”, “update” “delete” type goals (CRUD) in a single use case.

6. Identify the possible alternative flows and exceptions.

7. Iterate until uses cases, actors, and system boundary are stable.
Finding Use Cases

[Arlow and Neustadt p. 74-75]

- What functions will a specific actor want from the system?
- Does the system store and retrieve information?
- What happens when the system changes start (start/stop/etc)? Are any actors notified?
- Do any external events affect the system? What notifies the system about those events?
- Does the system generate any reports?
Advanced Use Case Modelling

- Actor generalization
- Use case generalization . . . skip this
- «include»
- «extends»

Remember: Use cases should be as simple as possible. Use these advanced techniques only if they improve the clarity of your model.
Actor Generalization

- Use this when actors have a lot in common (i.e., they interact with many of the same use cases)
- Factor out the common behaviour as an abstract actor
- Children inherit all relationships with use cases of the parent
Actor Generalization

- Purchaser
- Customer
- SalesAgent

The diagram shows the generalization hierarchy with Purchaser as the base class, Customer and SalesAgent as its sub-classes, and A, B, C, and D as potential roles or states associated with this hierarchy.
Include

«include»

- Include behaviour of one use case in another
- Avoids repetition of the same behaviour
- Similar to a procedure call
- Specify point of inclusion in base use case
- When included use case is done, control returns to base use case
- If included base case cannot be triggered directly by actors by itself we say it is incomplete
- Disadvantage: have to look multiple places to understand use case
Extend

«extend»

- Insert new behaviour in an existing use case
- Base use case has hooks where it can be extended
- Unlike «include», base case should flow independently of extension
- Use sparingly: there is disagreement over the semantics.
Overspecification in Use Cases

- Be careful not to over specify behaviour
  - Keep it short, keep it simple: main flow should fit on a single page.
  - Focus on what, not how:
    - Focus on externally visible behaviour
    - Are you specifying a sequence of events, in which the sequence doesn’t really matter? Ex: Order of entering data for new customer
    - Do you specify which elements from a set are selected, when any arbitrary element is needed? Ex: Selecting new arbitrary phone number
Underspecification in Use Cases

- Be careful not to under specify behaviour. Don’t forget:
  - variations on basic flow
  - exceptions
    For example, what happens to a phone call if there are no resources to allocate to it?
  - Specify behaviour for all possible inputs, both valid and invalid input
Rules of Thumb

- Keep names of data at an abstract level suitable for customers.
  For example, input and output events should have intuitive names.
  Ex: administrative i/o events: addcustomer(name, address)
  Ex: phone control i/o events: set_dial_tone(EN)

- Avoid functional decomposition: don’t try to structure the use cases as nested functions with «includes»

- Avoid deep hierarchies with «includes»
Rules of Thumb

- Think of use cases before use case diagram.
- Don’t spend too much time on the use case diagram. The textual description is the most important part.
- Avoid too much use of “extends” and “includes” in use case diagrams.
- Don’t describe the user interface.
- You don’t want too many use cases; if you have too many, you’ve probably included too much detail.

“If in doubt, leave it out”

- In practise, we don’t have to enumerate all the possible scenarios of a use case. A use case is an illustration of behaviour that we will further elaborate later in our SRS.
- Remember that these are informal descriptions.
Classic Mistakes

From Kulak and Guiney:

- Creating use cases from the application perspective rather than the user’s
- Including user interface details in the use case (for high-level use cases)
- Expanding the system boundary
- Writing pseudo-code for use case text
- Creating CRUD use cases first – rather look for use cases that are of most value to the user first
Use Cases

Advantages:

- Simple, easy to create
- All stakeholders understand them
- Often reflect user’s essential requirements
- Separates normal behaviour from exceptional behaviour

Disadvantages:

- Don’t scale well in size or complexity
Summary

- A use case describes observable behaviour triggered by a single external event. It may contain multiple scenarios.

- A scenario is a specific sequence of actions between the system and entities outside the system.
Summary

- Use Cases
  - What is a “use case”?
  - What is a “scenario”?
  - Use case descriptions
  - Use case diagrams
  - Finding use cases
  - Advanced use case modelling
  - Rules of thumb

Next Lecture Topic: Basic Notations

Required Readings: none