DA-IICT

Requirements Elicitation Techniques

Saurabh Tiwari

Overview

Elicitation Techniques

- Analysis of Existing Systems
  - Documentation, Observation, and Ethnography
- Interviews
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Use Cases

- When people talk, listen completely. Most people never listen.[1]

[1] Ernest Miller Hemingway (1899-1961)

Elicitation Techniques

Elicitation techniques

- Stakeholder analysis
- Analysis of existing systems or documentation, background reading
- Discourse analysis
- Task observation, ethnography
- Questionnaires
- Interviewing
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Pilot system
- Use cases and scenarios
- Risk analysis

Comparison of Data-Gathering Techniques[1]

Technique	Good for	Kind of data	Plus	Minus
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

[1] Preece, Rogers, and Sharp "Interaction Design: Beyond human-computer interaction", p214

Analysis of Existing Systems

Analysis of Existing Systems (1)

- Useful when building a new improved version of an existing system
- Important to know:
  - What is used, not used, or missing
  - What works well, what does not work
  - How the system is used (with frequency and importance) and it was supposed to be used, and how we would like to use it

Analysis of Existing Systems (2)

Why analyze an existing system?

- Users may become disillusioned with new system or do not like the new system if it is too different or does not do what they want (risk of nostalgia for old system)
- To appropriately take into account real usage patterns, human issues, common activities, relative importance of tasks/features
- To catch obvious possible improvements (features that are missing or do not currently work well)
- To find out which "legacy" features can/cannot be left out

Review Available Documentation

- Start with reading available documentation
  - User documents (manual, guides…)
  - Development documents
  - Requirements documents
  - Internal memos
  - Change histories
  - …
- Of course, often these are out of date, poorly written, wrong, etc., but it's a good starting point
- Discourse analysis
  - Use of words and phrases is examined in written or spoken language

Observation and Related Techniques (1)

- Observation
  - Get into the trenches and observe specialists “in the wild”
  - Shadow important potential users as they do their work
  - Initially observe silently (otherwise you may get biased information)
  - Ask user to explain everything he or she is doing
  - Session videotaping
- Ethnography also attempts to discover social, human, and political factors, which may also impact requirements

Observation and Related Techniques (2)

- Can be supplemented later with questionnaires
  - Based on what you know now – the results of observation
  - To answer questions that need comparison or corroboration (confirmation)
  - To obtain some statistics from a large number of users (look for statistical significance!), e.g.:
    - *How often do you use feature X?*
    - *What are the three features you would most like to see?*
- Can be supplemented later with interviews
  - After getting a better idea of what is to be done, probably some questions require more detailed answers
  - You will not be wasting other people's time or your own
  - This is very labour intensive!

Ethnography – Overview (1)

- Comes from anthropology, literally means "writing the culture"
- Essentially seeks to explore the human factors and social organization of activities → understand work
  - Studies have shown that work is often richer and more complex than is suggested by simple models derived from interviews
- Social scientists are trained in observation and work analysis
- Discoveries are made by observation and analysis, workers are not asked to explain what they do
  - Collect what is ordinary/what is it that people do (aim at making the implicit explicit)
  - Study the context of work and watch work being done

Ethnography – Overview (2)

- Useful to discover for example
  - What does a nuclear technician do during the day?
  - What does his workspace look like?
- Less useful to explore political factors
  - Workers are aware of the presence of an outside observer

Ethnography – Example (1)

- Sommerville et al. were involved in a project where they had to elicit the requirements of an air traffic control system
- They observed the air traffic controllers in action with the existing system
- Surprising observations
  - Controllers often put aircrafts on potentially conflicting headings with the intention of fixing them later
  - System generates an audible alarm when there is a possible conflict
  - The controllers close the alarms because they are annoyed by the constant warnings
- Incorrect conclusion
  - The controllers do not like audible alarms because they close them
- More accurate observation
  - The controllers do not like being treated like idiots

Ethnography – Example (2)

- Dealers at a stock exchange write tickets to record deals with old-fashioned paper/pencil method
- It was suggested to replace this with touch screens and headphones for efficiency and to eliminate distracting noise
- Study found that the observation of other dealers is crucial to the way deals are done
  - Market position was affected if deals were not continuously monitored
  - Even if only peripheral monitoring takes place
- “Improvements" would have destroyed the very means of communication among dealers

Source: Preece, Rogers, and Sharp “Interaction Design: Beyond human-computer interaction”

Interviews

Interviews (1)

- Requires preparation and good communication management
  - Achieve interview objectives without preventing the exploration of promising leads
- Interview as many stakeholders as possible
  - Not just clients and users
- Ask problem-oriented questions
  - Ask about specific details, but…
  - Detailed and solution-specific questions may miss the stakeholder's real requirements. Example:
    - Would you like Word 2007, Excel 2007 or both?
      vs.
    - Would you like to do word processing, computations, or both?

Interviews – Objectives and Process (2)

- Three main objectives:
  - Record information to be used as input to requirements analysis and modeling
  - Discover information from interviewee accurately and efficiently
  - Reassure interviewee that his/her understanding of the topic has been explored, listened to, and valued
- Process consists of four important steps:
  - Planning and preparation
  - Interview session
  - Consolidation of information
  - Follow-up
- Many strategies for questioning

Interviews – Planning and Preparation

Important to plan and prepare interviews

- Set goals and objectives for the interview
- Acquire background knowledge of the subject matter to conduct an effective interview
  - About the domain (vocabulary, problems...) but also about the interviewee (work tasks, attitude...)
- Prepare questions in advance, by subject
- Organize the environment for conducting an effective interview
  - Determine how the elicitation notes will be taken (manually, audio, video, by whom…)

Interviews – Session

- Make the interviewee comfortable and confident
- Be polite and respectful!
- Adjust to the interviewee
  - You have your goals – be persistent but flexible
- Interview several people at once to create synergy
- Try to detect political aspects as they may influence the said and the unsaid

Interviews – Elicitation Notes

- Revise and complete the elicitation notes after the interview
  - Needs to be done soon after because one forgets the details (and everything else)
- Identify inconsistencies and address them in a follow-up interview or by email
- Keep all diagrams, charts, models created during the discussions
- You are learning, so be precise
  - Pay attention to terminology
  - Use the interviewee's terminology
  - Identify synonyms
  - Create a glossary if necessary

Common Interviewing Mistakes (1)

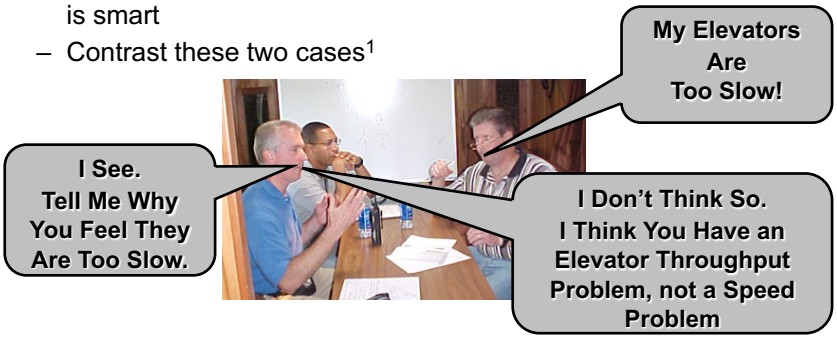
- Not interviewing all of the right people
  - Different points of view of stakeholders
- Asking direct questions too early
  - e.g., design of a transportation system:
    - How much horsepower do you need? (direct)
    - How many passengers? How far? How fast? (indirect)
  - e.g., camera design for novice photographer:
    - How important is control over shutter speed and aperture? (direct)
    - Will you be taking action shots, still shots, or both? (indirect)

Common Interviewing Mistakes (2)

- Interviewing one-at-a-time instead of in small groups
  - More people might help get juices flowing as in brainstorming
  - Users cannot think of everything they need when asked individually, but will recall more requirements when they hear others' needs
  - Reduces spotlight on individuals (may produce more interesting answers)
  - This interaction is called synergy, the effect by which group responses outperform the sum of the individuals' responses
  - Do not let one participant dominate the discussion
- Assuming that stated needs are exactly correct
  - Often users do not know exactly what they want and order "what he is eating"
  - Need to narrow what is asked for down to what is needed

Common Interviewing Mistakes (3)

- Trying to convince stakeholders that YOU are smart – wrong place to do that!
  - Instead take every opportunity to show you think the stakeholder is smart
  - Contrast these two cases[1]



[1] Alan M. Davis "Just enough requirements management"

Interviews – Start Up Questions (1)

- Context-free questions to narrow the scope a bit (Weinberg)
- Identify customers, goals, and benefits
  - Who is (really) behind the request for the system?
  - Who will use the system? Willingly?
  - Are there several types of users?
  - What is the potential economic benefit from a successful solution?
  - Is a (pre-existing) solution available from another source?

Interviews – Start Up Questions (2)

When do you need it by?

- Can you prioritize your needs?
- What are your constraints?
  - Time
  - Budget
  - Resources (human or otherwise)
- Expected milestones (deliverables and dates)?

Interviews – Start Up Questions (3)

Try to characterize the problem and its solution

- What would be a "good" solution to the problem?
- What problems is the system trying to address?
- In what environment will the system be used?
- Any special performance issues?
- Other special constraints?
- What is (un)likely to change?
- Future evolution?
- What needs to be flexible (vs. quick & dirty)?

Interviews – Start Up Questions (4)

Calibration and tracking questions

- Are you the right person to answer these questions?
- Are your answers "official"? If not, whose are?
- Are these questions relevant to the problem as you see it?
- Are there too many questions? Is this the correct level of detail?
- Is there anyone else I should talk to?
- Is there anything else I should be asking you? Have you told me everything you know about the problem?
- Do you have any questions?

Interviews – Start Up Questions (5)

Questions that cannot be asked directly (ask indirectly)

- Are you opposed to the system?
- Are you trying to obstruct/delay the system?
- Are you trying to create a more important role for yourself?
- Do you feel threatened by the proposed system?
- Are you trying to protect your job?
- Is your job threatened by the new system?
- Is anyone else's?

Interviews – Specific Questions (1)

Functional requirements

- What will the system do?
- When will the system do it?
- Are there several modes of operations?
- What kinds of computations or data transformations must be performed?
- What are the appropriate reactions to possible stimuli?
- For both input and output, what should be the format of the data?
- Must any data be retained for any period of time?

Interviews – Specific Questions (2)

Design Constraints

Physical environment

- Where is the equipment to be located?
- Is there one location or several?
- Are there any environmental restrictions, such as temperature, humidity, or magnetic interference?
- Are there any constraints on the size of the system?
- Are there any constraints on power, heating, or air conditioning?
- Are there constraints on the programming language because of existing software components?

Interviews – Specific Questions (3)

Design Constraints

- Interfaces
  - Is input coming from one or more other systems?
  - Is output going to one or more other systems?
  - Is there a prescribed way in which input/output need to be formatted?
  - Is there a prescribed way for storing data?
  - Is there a prescribed medium that the data must use?
- Standards
  - Are there any standards relevant to the system?

Interviews – Specific Questions (4)

Performance

- Are there constraints on execution speed, response time, or throughput?
- What efficiency measure will apply to resource usage and response time?
- How much data will flow through the system?
- How often will data be received or sent?

Interviews – Specific Questions (5)

Usability and Human Factors

- What kind of training will be required for each type of user?
- How easy should it be for a user to understand and use the system?
- How difficult should it be for a user to misuse the system?

Interviews – Specific Questions (6)

Security

- Must access to the system or information be controlled?
- Should each user's data be isolated from data of other users?
- Should user programs be isolated from other programs and from the operating system?
- Should precautions be taken against theft or vandalism?

Interviews – Specific Questions (7)

Reliability and Availability

- Must the system detect and isolate faults?
- What is the prescribed mean time between failures?
- Is there a maximum time allowed for restarting the system after failure?
- How often will the system be backed up?
- Must backup copies be stored at a different location?
- Should precautions be taken against fire or water damage?

Interviews – Specific Questions (8)

Maintainability

- Will maintenance merely correct errors, or will it also include improving the system?
- When and in what ways might the system be changed in the future?
- How easy should it be to add features to the system?
- How easy should it be to port the system from one platform (computer, operating system) to another?

Interviews – Specific Questions (9)

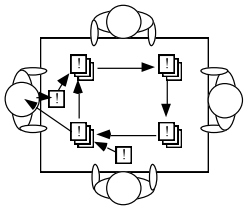
Precision and Accuracy

- How accurate must data calculations be?
- To what degree of precision must calculations be made?

Brainstorming

Brainstorming

- To invent new way of doing things or when much is unknown
  - When there are few or too many ideas
  - Early on in a project particularly when:
    - Terrain is uncertain
    - There is little expertise for the type of applications
    - Innovation is important (e.g., novel system)
- Two main activities:
  - The Storm: Generating as many ideas as possible (quantity, not quality) – wild is good!
  - The Calm: Filtering out of ideas (combine, clarify, prioritize, improve…) to keep the best one(s) – may require some voting strategy
- Roles: scribe, moderator (may also provoke), participants



Brainstorming – Objectives

- Hear ideas from everyone, especially unconventional ideas
  - Keep the tone informal and non-judgemental
  - Keep the number of participants “reasonable“ – if too many, consider a “playoff “-type filtering and invite back the most creative to multiple sessions
- Encourage creativity
  - Choose good, provocative project name.
  - Choose good, provocative problem statement
  - Get a room without distractions, but with good acoustics, whiteboards, coloured pens, provide coffee/donuts/pizza/beer
  - Provide appropriate props/mock-ups

Brainstorming – Roles

- Scribe
  - Write down all ideas (may also contribute)
  - May ask clarifying questions during first phase but without criticizing
- Moderator/Leader
  - Cannot be the scribe
  - Two schools of thought: traffic cop or agent provocateur
  - Traffic cop – enforces "rules of order", but does not throw his/her weight around otherwise
  - Agent provocateur – traffic cop plus more of a leadership role, comes prepared with wild ideas and throws them out as discussion wanes
    - May also explicitly look for variations and combinations of other suggestions

Brainstorming – Participants

- Virtually any stakeholder, e.g.
  - Developers
  - Domain experts
  - End-users
  - Clients
  - ...
- “Ideas-people” – a company may have a special team of people
  - Chair or participate in brainstorming sessions
  - Not necessarily further involved with the project

Brainstorming – The Storm

- Goal is to generate as many ideas as possible
- Quantity, not quality, is the goal at this stage
- Look to combine or vary ideas already suggested
- No criticism or debate is permitted – do not want to inhibit participants
- Participants understand nothing they say will be held against them later on
- Scribe writes down all ideas where everyone can see
  - e.g., whiteboard, paper taped to wall
  - Ideas do not leave the room
- Wild is good
  - Feel free to be gloriously wrong
  - Participants should NOT censor themselves or take too long to consider whether an idea is practical or not – let yourself go!

Brainstorming – The Calm

- Go over the list of ideas and explain them more clearly
- Categorize into "maybe" and "no" by pre-agreed consensus method
  - Informal consensus
  - 50% + 1 vote vs. “clear majority”
  - Does anyone have veto power?
- Be careful about time and people
  - Meetings (especially if creative or technical in nature) tend to lose focus after 90 to 120 minutes – take breaks or reconvene later
  - Be careful not to offend participants
- Review, consolidate, combine, clarify, improve
- Rank the list by priority somehow
- Choose the winning idea(s)

Brainstorming – Eliminating Ideas

- There are some common ways to eliminate some ideas
- Blending ideas
  - Unify similar ideas but be aware not to force fit everything into one idea
- Give each participant $100 to spend on the ideas
- Apply acceptance criteria prepared prior to meeting
  - Eliminate the ideas that do not meet the criteria
- Various ranking or scoring methods
  - Assign points for criteria met, possibly use a weighted formula
- Vote with threshold or campaign speeches
  - Possibly select top k for voting treatment

Brainstorming – Voting on Ideas

- Voting with threshold
  - Each person is allowed to vote up to n times
  - Keep those ideas with more than m votes
  - Have multiple rounds with smaller n and m
- Voting with campaign speeches
  - Each person is allowed to vote up to j < n times
  - Keep those ideas with at least one vote
  - Have someone who did not vote for an idea defend it for the next round
  - Have multiple rounds with smaller j

Brainstorming – Tool Support

- With many good ideas, some outrageous and even farfetched, brainstorming can be really fun!
- Creates a great environment that stimulates people and motivates them to perform well!
- Can be done by email, but a good moderator/leader is needed to
  - Prevent flamers to come into play
  - Prevent race conditions due to the asynchronous communication medium
  - Be careful not to go into too much detail
- Collaboration tools are also possible
  - TWiki and many other more appropriate tools such as BrainStorm and IdeaFisher

Joint Application Design (JAD)

Joint Application Design (JAD)

- A more structured and intensive brainstorming approach
- Developed at IBM in the 1970s
  - Lots of success stories
- "Structured brainstorming", IBM-style
  - Full of structure, defined roles, forms to be filled out...
- Several activities and six (human) roles to be played
- JAD session may last few days
- The whole is more than the sum of its parts. The part is more than a fraction of the whole.[1]

[1] Aristotle (384 BC – 322 BC)

Joint Application Design – Four Main Tenets

- Effective use of group dynamics
  - Facilitated and directed group sessions to get common understanding and universal buy-in
- Use of visual aids
  - To enhance understanding, e.g., props, prepared diagrams
- Defined process
  - I.e., not a random hodgepodge
- Standardized forms for documenting results

Joint Application Design – Applicability

- Used for making decisions on different aspects of a project
- Any process where consensus-based decision making across functional areas is required, e.g.,
  - Planning a project
  - Defining requirements
  - Designing a solution

Joint Application Design – Activities

- Preparation
  - Pre-session Planning
  - Pre-work
- Working Session
- Summary
  - Follow-up
  - Wrap-up

Joint Application Design – Pre-session Planning

- Preparation is essential – this is not an informal session
- Evaluate project
  - Identify contentious issues and scope of JAD session
- Select JAD participants
- Create preliminary agenda
- Determine deliverables for the working session
- Enable participants to prepare for the session

The 6 “P”s

1. Purpose - Why do we do things? (Goals, needs, motivation)
2. Participants - Who is involved? (People, roles, responsibilities)
3. Principles - How do we function? (Guidelines, working agreements, ground rules)
4. Products - What do we create? (Deliverables, decisions, plans, next steps)
5. Place - Where is it located? (Venue, logistics)
6. Process - When do we do what? (Activities, sequence)

[1] Ellen Gottesdiener, Requirements by Collaboration: Facilitating Workshops to Define Stakeholder Needs, RE Conference 2006

Joint Application Design – Pre-work

- Gather information
- Clear schedules for the working session
- Refine session agenda
- Finalize pre-session assignments
- Prepare material for session (flip-charts, presentations, markers, pizza...)

Joint Application Design – Working Session

- Set-up stage
  - Session leader welcomes participants, presents task to be discussed, establishes rules and what is on/off topic…
- Generate common understanding
  - Brainstorming…
- Achieve consensus on decisions
- Generate ownership of results
- Create the deliverables (using standard JAD forms)
- Identify open issues and questions

Joint Application Design – Follow-up and Wrap-up

- Follow-up
  - Resolve open issues and questions
  - Follow-up on action items
  - Re-evaluate project
- Wrap-up
  - Review results of follow-up items
  - Evaluate the JAD process
  - Discuss "lessons learned"
  - Finalize deliverables

Joint Application Design – Roles (1)

- Session leader
  - Organizer, facilitator, JAD expert
  - Good with people skills, enthusiastic, sets tone of meeting
- Analyst
  - Scribe++
  - Produces official JAD documents, experienced developer who understands the big picture, good philosopher/writer/organizer
- Executive sponsor
  - Manager who has ultimate responsibility for product being built
  - Provides strategic insights into company's high-level goals/practices, makes executive decisions later on as required

Joint Application Design – Roles (2)

- User representatives
  - Selection of knowledgeable end-users and managers
  - Come well-prepared with suggestions and ideas of needs, will brainstorm for new or refined ideas, eventually review completed JAD documents
- Information system representatives
  - Technical expert on ISs
  - Helps users think big, know what is easy/hard/cheap/expensive, mostly there to provide information rather than make decisions
- Specialists
  - Technical expert on particular narrow topics, e.g., security, application domain, law, UI issues…

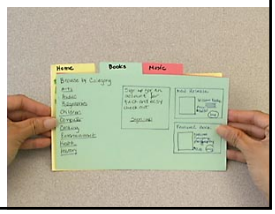
Prototyping

Prototyping

- A software requirements prototype is a mock-up or partial implementation of a software system
  - Helps developers, users, and customers better understand system requirements
  - Helps clarify and complete requirements
  - Provides early response to “I'll know it when I’ll see (or won’t see) it” attitude
  - Effective in addressing the “Yes, But” and the “Undiscovered Ruins” syndromes
  - Helps find new functionalities, discuss usability, and establish priorities
- Prototyping is effective in resolving uncertainties early in the development process
  - Focus prototype development on these uncertain parts
  - Encourages user participation and mutual understanding

Prototyping – Realizations

- Prototypes can take many forms:
  - Paper prototypes (see http://www.paperprototyping.com/)
    - Prototype on index card
    - Storyboard
  - Screen mock-ups
  - Interactive prototypes
    - Using high-level languages (e.g., Visual Basic, Delphi, Prolog)
    - Using scripting languages (e.g., Perl, Python)
    - Using animation tools (e.g., Flash/Shockwave)
  - Models (executables)
  - Pilot systems
  - …



Prototyping – Types

- Horizontal: focus on one layer – e.g., user interface
- Vertical: a slice of the real system
- Evolutive: turned into a product incrementally, gives users a working system more quickly (begins with requirements that are more understood)
- Throw-away: less precise, thrown away, focusing on the less well-understood aspects of the system to design, designed to elicit or validate requirements

Prototyping – Fidelity (1)

- Fidelity is the extent to which the prototype is real and (especially) reactive
- Fidelity may vary for throw-away prototypes
- High-fidelity
  - Applications that "work" – you press a button and something happens
  - Often involves programming or executable modeling languages
  - Advantages:
    provides an understanding of functionality, reduce design risk, more precise verdicts about requirements
  - Disadvantages:
    takes time to build, more costly to build, sometimes difficult to change, false sense of security, often focuses on details rather than on the goals and important issues

Prototyping – Fidelity (2)

Low-fidelity

- It is not operated – it is static
- Advantages:
  easy and quick to build, cheaper to develop, excellent for interfaces, offers the opportunity to engage users before coding begins, encourage creativity
- Disadvantages:
  may not cover all aspects of interfaces, are not interactive, may seem non-professional in the eyes of some stakeholders (sigh!)

Prototyping – Risks

- Prototypes that focus on user-interface tends to lose the focus of demonstrating/exploring functionality
- Prototypes can bring customers' expectations about the degree of completion unrealistically up
- Do not end-up considering a throwaway prototype as part of the production system
  - Always clearly state the purpose of each prototype before building it

Use Cases

Developing Use Case Models of Systems

- Description of a sequence of interactions between a system and external actors
- Developed by Ivar Jacobson
  - Not exclusively for object-oriented analysis
- Actors – any agent that interact with the system to achieve a useful goal (e.g., people, other software systems, hardware)
- Use case describes a typical sequence of actions that an actor performs in order to complete a given task
  - The objective of use case analysis is to model the system
    - … from the point of view of how actors interact with this system
    - … when trying to achieve their objectives
  - A use case model consists of
    - A set of use cases
    - An optional description or diagram indicating how they are related

Use Cases

- A use case should describe the user's interaction with the system ...
  - Not the computations the system performs
- In general, a use case should cover the full sequence of steps from the beginning of a task until the en
- A use case should only include actions in which the actor interacts with the computer
  - Some views differ on this one!!!

Use Cases – Abstraction Level

- A use case should be written so as to be as independent as possible from any particular implementation / user interface design
- Essential use cases (Constantine & Lockwood)
  - Abstract, technology free, implementation independent
  - Defined at earlier stages
  - e.g., customer identifies herself
- Concrete use cases
  - Technology/user interface dependent
  - e.g., customer inserts a card, customer types a PIN

Scenarios (1)

- A scenario (according to the UML/UC community) is an instance of a use case
  - It expresses a specific occurrence of the use case (a specific path through the use case)
    - A specific actor ...
    - At a specific time ...
    - With specific data …
  - Many scenarios may be generated from a single use case description
  - Each scenario may require many test cases
- Rather used in a generic way in this course (as is often the case in requirements engineering)

Scenarios (2)

- A use case includes primary and secondary scenarios
- 1 primary scenario
  - Normal course of events
- 0 or more secondary scenarios
  - Alternative/exceptional course of events, variations of primary scenario
  - An alternative scenario meets the intent of the use case but with a different sequence of steps
  - An exceptional scenario addresses the conditions of main case and alternative cases that differ from the norm and cases already covered
  - Example with consensus as a goal
    - Primary scenario: vote in a session
    - Alternative scenario: voting in several sessions
    - Exceptional scenario: what to do with a non-registrant who wishes to vote

Types of Scenarios

- As-is scenario
  - Used in describing a current situation, usually used in re-engineering projects, the user describes the system
- Visionary scenario
  - Used to describe a future system, usually used in greenfield engineering and reengineering projects
  - Can often not be done by the user or developer alone
- Evaluation scenario
  - User tasks against which the system is to be evaluated
- Training scenario
  - Step by step instructions that guide a novice user through a system

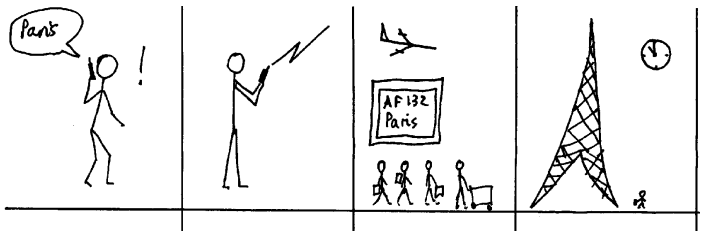
Representation of Scenarios (1)

Various approaches

- Text (informal, formal), diagrams (state, sequence ...), video, animation, comics, storyboard, collaborative workshops (pass the microphone or the ball)…
- There are specialized notation such as UML (sequence, activity, use case, interaction, and collaboration diagrams), Message Sequence Charts (MSC), Live Sequence Charts, and Use Case Maps (UCM)

Representation of Scenarios (2)

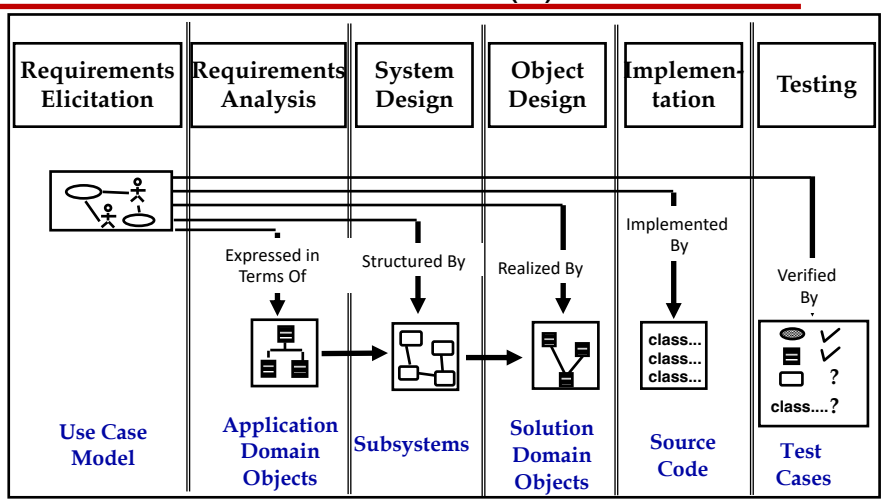
- Different representations may be useful in specific situations
  - For example, storyboards, often used in the film industry, can describe situations, roles, and sequences of tasks in a fast, compact, and polyglot way[1]



- Some scenario-based approaches are very ideological or dogmatic

[1] I. Alexander

Use Case-Driven Software Lifecycle Activities (1)

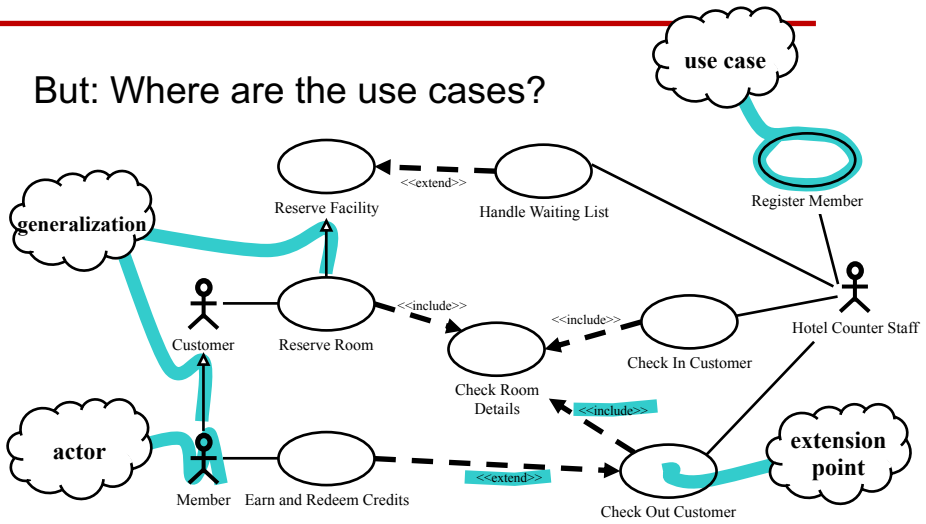


Use Case-Driven Software Lifecycle Activities (2)

- Scenarios guide elicitation, analysis, design, and testing
  - There are many scenario-based approaches
  - E.g., XP employs user stories (scenarios) to directly generate tests that will guide software design and verification
- Developers are often unable to speak directly to users
- Scenarios provide a good enough approximation of the “voice of the user”

Use Case Modeling with UML

But: Where are the use cases?



Use Case Diagrams

- To define system boundary (subject), actors, and use cases
  - Subject could be: a physical system, a component, a subsystem, a class
- To structure and relate use cases
  - Associate actors with use cases
  - Include relation
  - Extend relation
  - Generalization (of actors and use cases)

Use Case Diagrams – <<include>> (Inclusions)

- Inclusions allow one to express commonality between several different use cases
- Inclusions are included in other use cases
  - Even very different use cases can share a sequence of actions (reuse)
  - Enable you to avoid repeating details in many use cases (consistency)
- An inclusion represents the execution of a lower-level task with a lower-level goal (→ decomposition of complex tasks)
- Base use case references the included use case as needed
- Base use case cannot exist without the included use case
- When included use case is done, control returns to base use case
- Disadvantage: have to look in multiple places to understand use case

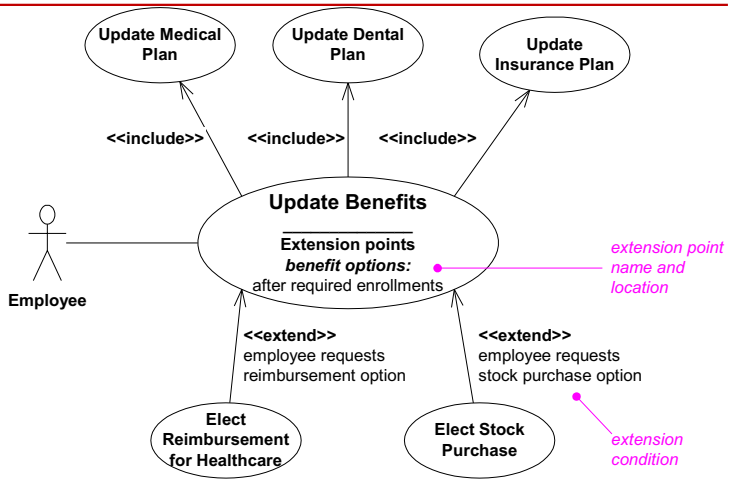
Use Case Diagrams – <<extend>> (Extensions)

- Used to make optional interactions explicit or to handle exceptional cases
- By creating separate use case extensions, the description of the basic use case remains simple
  - Note: the base use case can still be executed without the use case extension
- Extension points must be created explicitly in the base use case
- Use sparingly: there is disagreement over the semantics

Use Case Diagrams – Generalizations

- Much like super-classes in a class diagram
  - Need to satisfy “is-a” relation
- Applies to use cases and to actors
  - A generalized use case represents several similar use cases
  - One or more specializations provide details of the similar use cases
  - Inheriting use case can replace steps of inherited use case
  - Particularly useful for actors (clearer here, unlike use case generalization where the semantics are unclear – use generalization of use cases with caution)

Example: HR System



Use Case Templates (1)

- There are many different templates for use cases but they often consist of a subset of the following items:
- Identifier: unique label for use case used to reference it elsewhere
- Name: succinctly state user task independently of the structure or implementation
  - Suggested form “verb object” (e.g., Order a product)
- Authors: people who discovered use case
- Goal: short description of expected outcome from actors’ point of view
- Preconditions: what needs to be satisfied before use case can begin
- Postconditions: state of system after completion of use case
  - Minimal guarantee: state of system after completion regardless of outcome

Use Case Templates (2)

- Primary actor: initiates the use case
- Participants (secondary actors): other actors involved in use case, provide services to the system and interact with the system after the use case was initiated
- Related requirements: identifiers of functional and non-functional requirements linked to the use case
- Related use cases: identifiers of related use cases
  - Specify relationship: e.g.
    - Supposes use case UC2 has been successfully completed
    - Alternative to use case UC3
    - ...
- Description of events (scenarios)
  - Different use case description formats
  - Narrative, Simple column, Multiple columns

Use Case Templates – Narrative Form

- Paragraph focusing on the primary scenario and some secondary ones
- Very useful when the stakeholders first meet

A User inserts a card in the Card reader slot. The System asks for a personal identification number (PIN). The User enters a PIN. After checking that the user identification is valid, the System asks the user to chose an operation…

Use Case Templates – Simple Column Form

Linear sequences (main and alternatives)

1. A User inserts a card in the Card reader slot.
2. The system asks for a personal identification number (PIN).
3. The User enters a PIN.
4. The System checks that the user identification is valid.
5. The System asks the user to chose an operation

1.a The Card is not valid.
1.a.1. The System ejects the Card.
4.a The User identification is not valid.
4.a.1 The System ejects the Card.

Use Case Templates – Multiple Column Form

- One column per actor
- Allows for a more detailed view

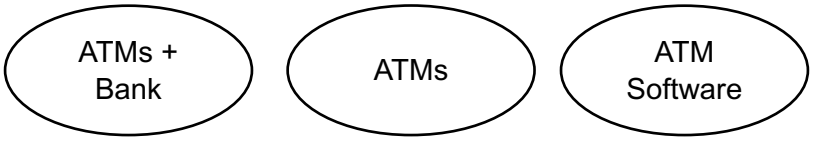
User's actions	System responses
1. Insert a card in the Card reader slot. (card is not valid see alternative 1.1)	2. Ask for a personal identification number (PIN).
3. Enter a PIN.	4. Check that the user identification is valid. (identification is not valid see alternative 4.1) 5. Ask User to chose an operation

Use Case Templates – Example Template

- **A. Name**: Give a short, descriptive name to the use case
- **B. Actors**: List the actors who can perform this use case
- **C. Goals**: Explain what the actors are trying to achieve
- **D. Preconditions**: State of the system before the use case
- **E. Description**: Give a short informal description
- **F. Trigger**: Describe the event that initiates the activity
- **G. Summary**: Summarize what occurs as the actors perform the use case
- **F. Related use cases**: (in accordance with use case diagram)
- **G. Steps**: Describe each step using some form (single/multiple columns)
- **H. Postconditions**: State of the system following completion
- **I. Special Requirements**: Non-functional requirements and constraints

Use Case Development – Example: ATM System (1)

1. Determine candidate system scope and boundaries
   - Identify stakeholders
   - Identify problem - Create problem statement
   - Use interviews and other techniques



1
- Customer

2
- Customer
- Bank

3
- Card reader
- Cash dispenser
- Key pad
- Touch screen
- Printer
- Bank

Use Case Development – Example: ATM System (2)

2. Identify actors
   - Who interacts with the system?
     - Who or what uses the system?
       - For what goals?
       - What roles do they play?
     - Who installs the system?
     - Who or what starts and shuts down the system?
     - Who maintains the system?
     - Who or what gets and provides information to the system?
     - Does anything happen at a fixed time?
   - Check for possible actor generalization

Example ATM with scope 2

Bank Customer

Installation Technician

Administrator

Administrator

Bank

Weekly Reports

Use Case Development – Example: ATM System (3)

2. Identify actors (cont'd)
   - Choose actors' names carefully
   - Actors give value, get value from system or both
   - Should reflect roles rather than actual people
     - An actor specifies a role an external entity adopts when it interacts directly with your system
     - People / things may play multiple roles simultaneously or over time
   - Use right level of abstraction

Poor actor name	Good actor name
Clerk	Pension Clerk
Third-level supervisor	Sale supervisor
Data Entry Clerk #165	Product accountant
Eddie "The Dawg" Taylor	Customer service representative

Use Case Development – Example: ATM System (4)

3. Identify use cases
   - Identify and refine actors' goals
     - Why would actor AAA use the system?
   - Identify actors' tasks to meet goals
     - What interactions would meet goal GGG of actor AAA?
   - Chose appropriate use case names
     - Verb-noun construction
     - No situation-specific data
     - Not tied to organization structure, paper forms, computer implementation, or manual process (e.g., enter form 104-B, complete approval window, get approval from immediate supervisor in Accounting Department)

Example actor: Customer

Goal: Access account 24/7 for regular banking operations (withdrawal, deposit, statement...) in a timely and secure way.

To be refined into sub-goals.

From goals we can identify tasks: Withdraw Cash, Make Deposit, Print Account Statement....

Use Case Development – Example: ATM System (5)

3. Identify use cases (cont'd)
   - Develop a brief description in narrative form
   - e.g., description of use case Withdraw Cash

   The Customer identifies herself to the system, then selects the cash withdrawal operation. The system asks for the amount to withdraw. The Customer specifies the amount. The system provides the requested amount. The Customer takes the amount and leaves.

4. Identify preconditions
   - e.g., preconditions of use case Withdraw Cash
     - System is in operation, cash is available

Use Case Development – Example: ATM System (6)

5. Definition of primary scenario
   - Happy day story where everything goes fine
6. Definition of secondary scenarios (alternatives/exceptions)
   - A method for finding secondary scenarios is to go through the primary scenarios and ask:
     - Is there some other action that can be taken at this point? (alternate scenario)
     - Is there something that could go wrong at this point? (exception scenario)
     - Is there some behavior that could happen at any time? (alternative scenario unless it is an error, then it would be an exception scenario)

Example use case: Withdraw Cash

Not enough cash available

Incorrect identification

Customer forgets card in card reader

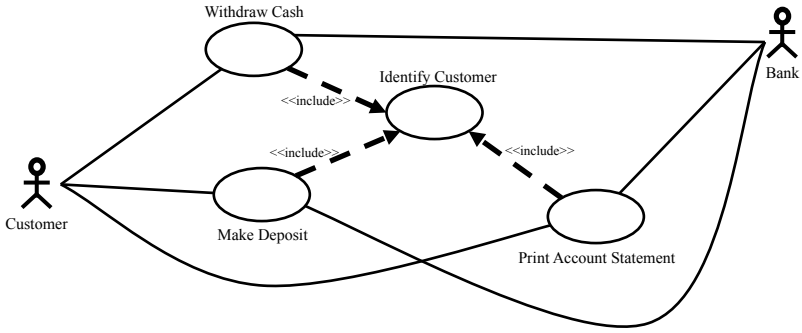
Incorrect amount entered

Customer forgets cash in dispenser

Use Case Development – Example: ATM System (7)

7. Structure use case diagram

- Identify commonalities and specify included use cases
- Identify variants and specify extended use cases



Use Cases Development (1)

Heuristics for finding use cases

- Select a narrow vertical slice of the system (i.e., one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
  - Could target high value or high risk first
- Select a horizontal slice (i.e., many scenarios) to define the scope of the system
  - Discuss the scope with the user
- Use illustrative prototypes (e.g., mock-ups) as visual support
- Find out what the user does
  - Task observation (preferable to questionnaires)

Use Cases Development (2)

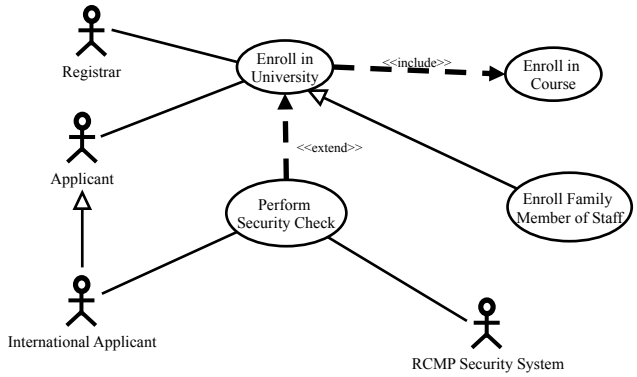
Alternative ways of identifying use cases (Ham, Larman)

- Identify external events to which the system must respond, and then relate these events to participating actors and specific use cases
- Express business processes in terms of specific scenarios, generalize the scenarios into use cases, and identify the actors involved in each use case
- Derive likely use cases from existing functional requirements – if some requirements do not trace to any use case, consider whether you really need them

Use Cases Development (3)

- Often one use case (or a very small number) can be identified as central to the system
  - The entire system can be built around this particular use case
- There are other reasons for focusing on particular use cases:
  - Some use cases will represent a high risk because for some reason their implementation is problematic
  - Some use cases will have high political or commercial value
- Approach is iterative
  - System scope and boundaries may change as more information is known about actors, their goals, and use cases

Example: University Registration System (1)



Example: University Registration System (2)

- Name: Enroll in University
- Identifier: UC 19
- Goal: Enroll applicant in the university
- Preconditions:
  - The Registrar is logged into the system.
  - The Applicant has already undergone initial checks to verify that they are eligible to enroll.
- Postconditions:
  - The Applicant will be enrolled in the university as a student if they are eligible.

Example: University Registration System (3)

Main Flow:

1. An applicant wants to enroll in the university.
2. The applicant hands a filled out copy of form UI13 University Application Form to the registrar.
3. The registrar visually inspects the forms.
4. The registrar determines that the forms have been filled out properly.
5. The registrar selects to Create New Student.
6. The system displays the Create Student Screen.
7. The registrar inputs the name, address, and phone number of the applicant.

[Extension Point: XPCheck]

8. The system determines that the applicant does not already exist within the system.
9. The system determines that the applicant is on the eligible applicants list.
10. The system adds the applicant to its records.
11. The registrar enrolls the student in courses via use case UC 17 Enroll in Course.
12. The system prepares a bill for the applicant enrollment fees.

Alternate Flows:

- 4.a The forms have not been adequately filled…

Example: University Registration System (4)

- Name: Perform Security Check
- Identifier: UC 34

At Extension Point XPCheck:

1. The registrar asks for security check results about applicant.
2. The system asks RCMP Security System for applicant security check results.
3. The RCMP Security System responds that applicant has been cleared.

3.a The Security System responds that applicant has not been cleared

Example: University Registration System (5)

- Name: Enroll Family Member of Staff
- Identifier: UC 20
- Inherits From: UC 19

Main Flow:

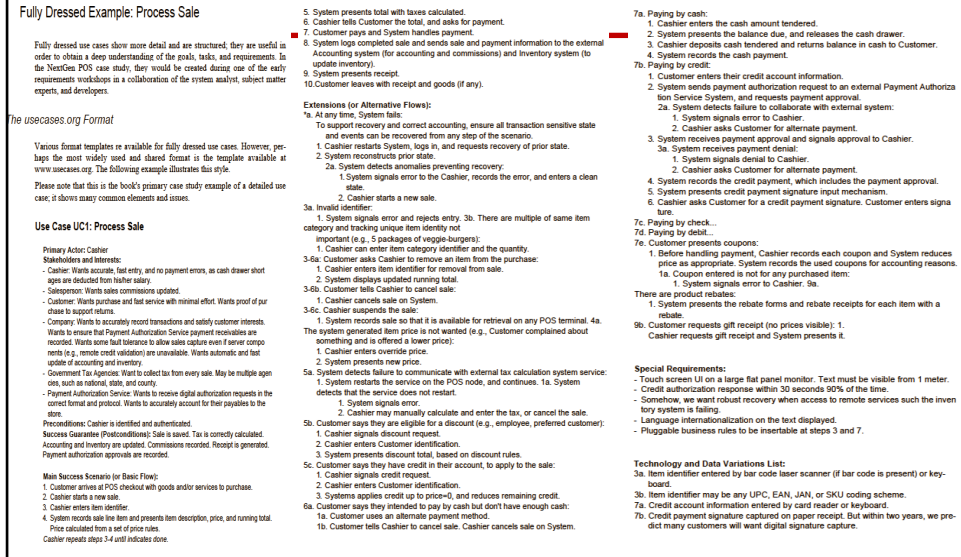
1. An applicant family member wants to enroll in the university.
2. The applicant hands a filled out copy of form UI15 University Application Form for Family Members to the registrar.

…

12. The system prepares a bill for the applicant enrollment fees based on staff family members rate.

Alternate Flows: …

Example: Point of Sale Application



Fully Dressed Example: Process Sale

Fully dressed use cases show more detail and are structured; they are useful in order to obtain a deep understanding of the goals, tasks, and requirements. In the NextGen POS case study, they would be created during one of the early requirements workshops in a collaboration of the system analyst, subject matter experts, and developers.

The usecases.org Format

Various format templates re available for fully dressed use cases. However, perhaps the most widely used and shared format is the template available at www.usecases.org. The following example illustrates this style.

Please note that this is the book's primary case study example of a detailed use case; it shows many common elements and issues.

Use Case UC1: Process Sale

Primary Actor: Cashier
Stakeholders and Interests:
- Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer short ages are deducted from his/her salary.
- Salesperson: Wants sales commissions updated.
- Customer: Wants purchase and fast service with minimal effort. Wants proof of pur chase to support returns.
- Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server compo nents (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
- Government Tax Agencies: Want to collect tax from every sale. May be multiple agen cies, such as national, state, and county.
- Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.
Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Main Success Scenario (or Basic Flow):
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):
*a. At any time, System fails:
To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.
2a. System detects anomalies preventing recovery:
1. System signals error to the Cashier, records the error, and enters a clean state.
2. Cashier starts a new sale.

3a. Invalid identifier:
1. System signals error and rejects entry. 3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):
1. Cashier can enter item category identifier and the quantity.

3-6a. Customer asks Cashier to remove an item from the purchase:
1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:
1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:
1. System records sale so that it is available for retrieval on any POS terminal. 4a. The system generated item price is not wanted (e.g., Customer complained about something and is offered a lower price):
1. Cashier enters override price.
2. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:
1. System restarts the service on the POS node, and continues. 1a. System detects that the service does not restart.
1. System signals error.
2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):
1. Cashier signals discount request.
2. Cashier enters Customer identification.
3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:
1. Cashier signals credit request.
2. Cashier enters Customer identification.
3. Systems applies credit up to price=0, and reduces remaining credit.

6a. Customer says they intended to pay by cash but don't have enough cash:
1a. Customer uses an alternate payment method.
1b. Customer tells Cashier to cancel sale. Cashier cancels sale on System.
7a. Paying by cash:
1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:
1. Customer enters their credit account information.
2. System sends payment authorization request to an external Payment Authoriza tion Service System, and requests payment approval.
2a. System detects failure to collaborate with external system:
1. System signals error to Cashier.
2. Cashier asks Customer for alternate payment.
3. System receives payment approval and signals approval to Cashier.
3a. System receives payment denial:
1. System signals denial to Cashier.
2. Cashier asks Customer for alternate payment.
4. System records the credit payment, which includes the payment approval.
5. System presents credit payment signature input mechanism.
6. Cashier asks Customer for a credit payment signature. Customer enters signa ture.

7c. Paying by check…
7d. Paying by debit…
7e. Customer presents coupons:
1. Before handling payment, Cashier records each coupon and System reduces price as appropriate. System records the used coupons for accounting reasons.
1a. Coupon entered is not for any purchased item:
1. System signals error to Cashier. 9a.

There are product rebates:
1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible): 1.
Cashier requests gift receipt and System presents it.

Special Requirements:
- Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90% of the time.
- Somehow, we want robust recovery when access to remote services such the inven tory system is failing.
- Language internationalization on the text displayed.
- Pluggable business rules to be insertable at steps 3 and 7.

Technology and Data Variations List:
3a. Item identifier entered by bar code laser scanner (if bar code is present) or key- board.
3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.
7a. Credit account information entered by card reader or keyboard.
7b. Credit payment signature captured on paper receipt. But within two years, we pre- dict many customers will want digital signature capture.

Source: Craig Larman "Applying UML and Patterns"

Use Case Development – Rules of Thumb (1)

Be careful not to over specify behavior

- Keep it short, keep it simple: main flow should fit on a single page
- Focus on what, not how:
  - Focus on externally visible behavior
  - Are you specifying a sequence of events, in which the sequence does not really matter?
    - Example: Order of entering data for new customer
  - Do you specify which elements from a set are selected, when any arbitrary element is needed?
    - Example: Selecting new arbitrary phone number

Use Case Development – Rules of Thumb (2)

- Be careful not to under specify behavior
  - Do not forget variations on basic flow
  - Do not forget exceptions
    - For example, what happens to a phone call if there are no resources to allocate to it?
- Specify behavior for all possible inputs, both valid and invalid input

Use Case Development – Rules of Thumb (3)

- Keep names, data at an abstract level suitable for users
  - For example, input and output events should have intuitive names
  - Avoid data definition in use cases
- Use cases need to be understandable by users
  - They must validate them
- Avoid functional decomposition
  - Do not try to structure the use cases as nested functions with «includes»
  - Avoid deep hierarchies with «includes»

Use Case Development – Rules of Thumb (4)

- Think of use cases before use case diagram
- Do not 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
- Do not describe the user interface
- You do not want too many use cases; if you have too many, you have probably included too much detail (“If in doubt, leave it out”)
  - Do not attempt to describe everything – too many variations – too many things that can go wrong
  - The requirements specification captures a more complete picture

Benefits of Use Case-Based Software Development

- They can help to define the scope of the system
- They are often used to plan the development process
- They are used to both develop and validate the requirements
  - Simple, easy to create
  - All stakeholders understand them
  - Often reflect user's essential requirements
  - Separates normal behavior from exceptional behavior
- They can form the basis for the definition of test cases
- They can be used to structure user manuals

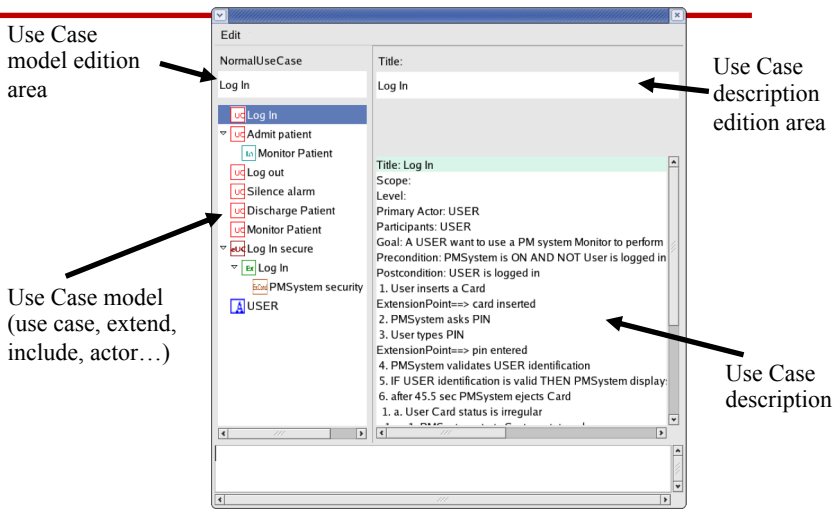
Use Cases are Not a Panacea…

- The use cases themselves must be validated
  - Using the requirements validation methods
  - Question/observe many types of users
- There are some aspects of software that are not covered by use case analysis
  - How to integrate nun-functional requirements?
- Innovative solutions may not be considered
- Scalability and maintainability
- Others discussed by Stephen Ferg in “What's Wrong with Use Cases”

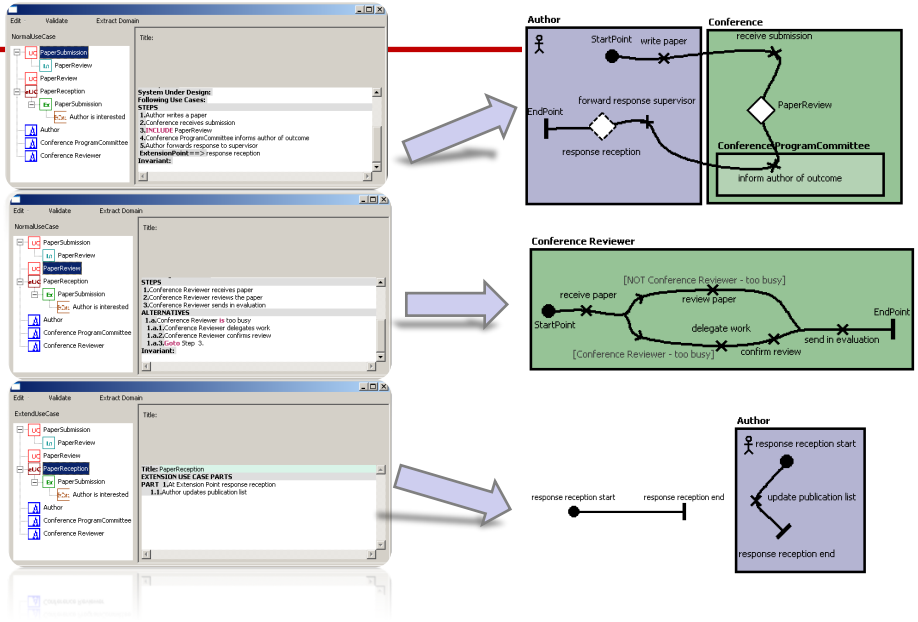
Tools

- Many UML tools support use case diagrams, without really supporting use cases well
- UCEd tool (Prof. Somé), to help capture/validate use cases
  - Use Case edition (structured English)
  - Domain model edition (and automatic extraction)
  - Scenario edition
  - Use Case & Domain model validation
  - Use Cases combination in state models
  - Simulation of executable model derived from Use Cases
  - Scenario generation
  - http://www.site.uottawa.ca/~ssome/Use_Case_Editor_UCEd.html

Use Case Edition with UCEd



UCEd Model Import in jUCMNav



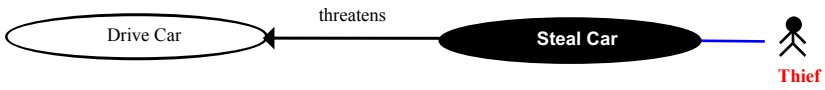
Don’t be Negative, But…

- Be ready to face the music!
  - In business, some people might wish to see you fail…
  - There are unforeseen events in any project
  - Open systems are subject to various threats
  - Software contains various kinds of bugs
- Remember Murphy’s Law…
  - “Anything that can go wrong will go wrong.”

Think about Negative Scenarios?

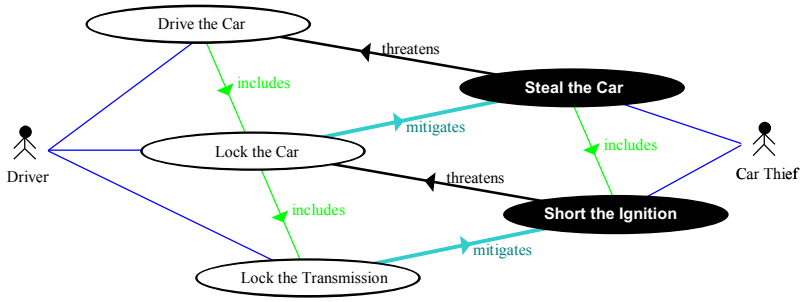
- A negative scenario is a scenario whose goal is
  - Undesirable from the business organization's viewpoint
  - Desirable from a hostile agent's viewpoint (not necessarily human)
- Consider them beforehand
  - As well as potential solutions
- Or…
  - Wait until it is too late to react…

Misuse Case



- Proposed by Sindre et Opdahl (2000)
  - Capture use cases that a system must be protected against
  - Goal is to threaten the system
  - Obvious applications for security and risk analysis
- The misuse case's misactor is a hostile agent
- The colors of the ellipse are inverted

A MiniMax[1] for Security



Use Cases for 'Car Security'

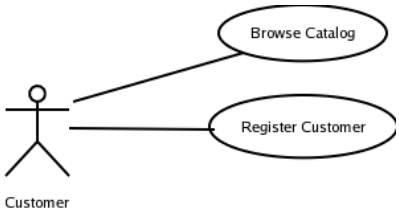
- Similar to a chess match…
  - White's best move is to find Black's best move and to counter it
- New relations:
  - threatens (from misuse case to use case)
  - mitigates (from use case to misuse case)

[1] Minimax: minimizing the maximum possible loss

Finding Misuse Cases – Step 1

- Start from a normal use case diagram
- Find misactors (hostile roles)
  - Who are the misactors, who want:
  - To harm the system, its stakeholders, or their resources intentionally

    or
  - To achieve goals that are incompatible with the system's goals



Competitor

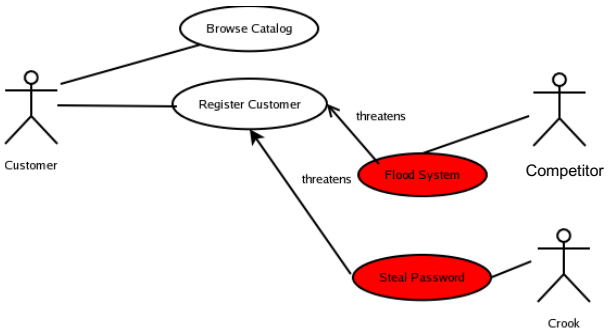
Crook

…

Finding Misuse Cases – Step 2

Find misuse cases

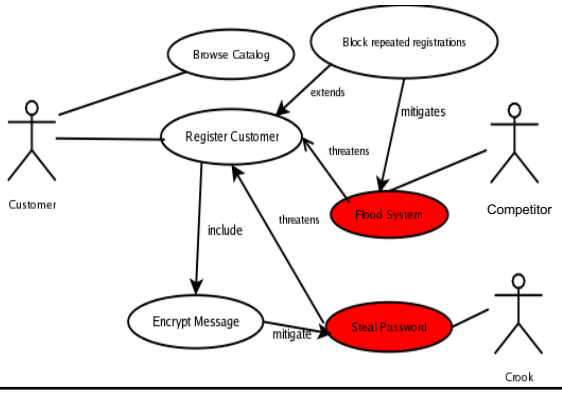
- Ask what would a misactor do to harm system
- Express goals of misactors (if needed elaborate with scenarios)
- Add relationships (threaten)



Finding Misuse Cases – Step 3

Mitigate misuse cases

- Ask what would neutralize the threats
- New included use case, new extension use case, or new secondary scenario to existing use case might be added



Benefits and Risks of Misuse Cases

- Benefits
  - Elicitation of security and safety requirements
  - Early identification of threats, mitigations, and exceptions that could cause system failure
  - Early identification of test cases
  - Documentation of rationales
- Risks
  - Get into premature design solutions in step 3 (mitigation)
    - Goal should be to find requirements (safety, security…)
  - Missing misactors and threats in a partial view

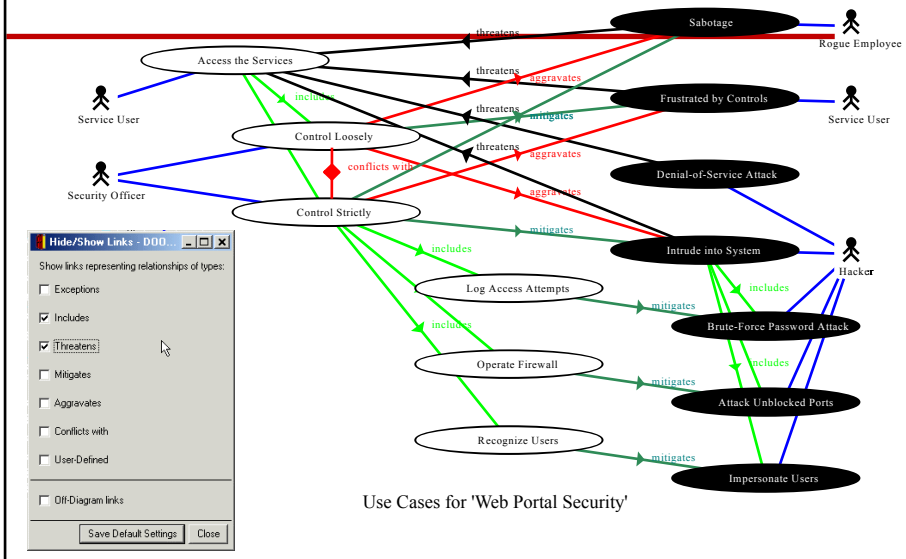
Tool: DOORS Plug-in

- Scenario Plus (for Telelogic DOORS)
- Textual / Graphical output (HTML)
- Automatic links, metrics, etc.
- Upon referencing: automatic creation of use/misuse cases

ID	Use Cases	Links to Included Use/Misuse Cases
UC-30	**2.1.3 Lock the Car**	
UC-31	**2.1.3.1 Primary Scenario**	
UC-35	System automatically Locks the Transmission to prevent the Car thief from Stealing the Car.	UC-49 Lock the Transmission UC-70 Steal the Car

- Automatic creation of links between misuse and use cases, by searching for underlined use case names with simple fuzzy matching

Conflict and Trade-off Analysis



Use Cases for 'Web Portal Security'