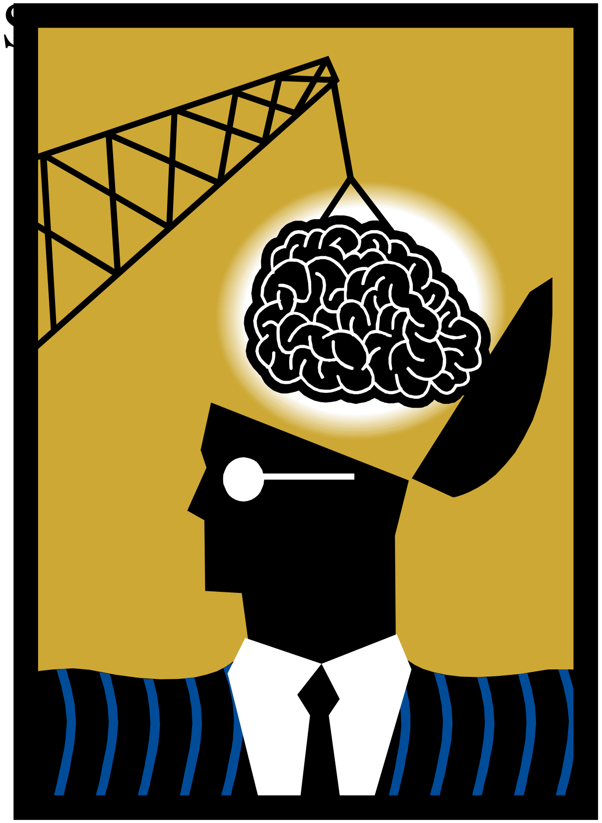
Requirements Elicitation

With material from: Jo Atlee, Nancy Day, Dan Berry University of Waterloo

What is elicitation?

- Elicitation means “to bring out, to evoke, to call forth”
- Requirements elicitation is “the process of discovering the requirements for a system by communication with customers, system users and others who have a stake in the system development” [Ian Sommerville and Pete S
- Human activity involving interaction

  between human beings:
  - Clients
  - Users
  - Systems analysts
  - Systems developers
  - …

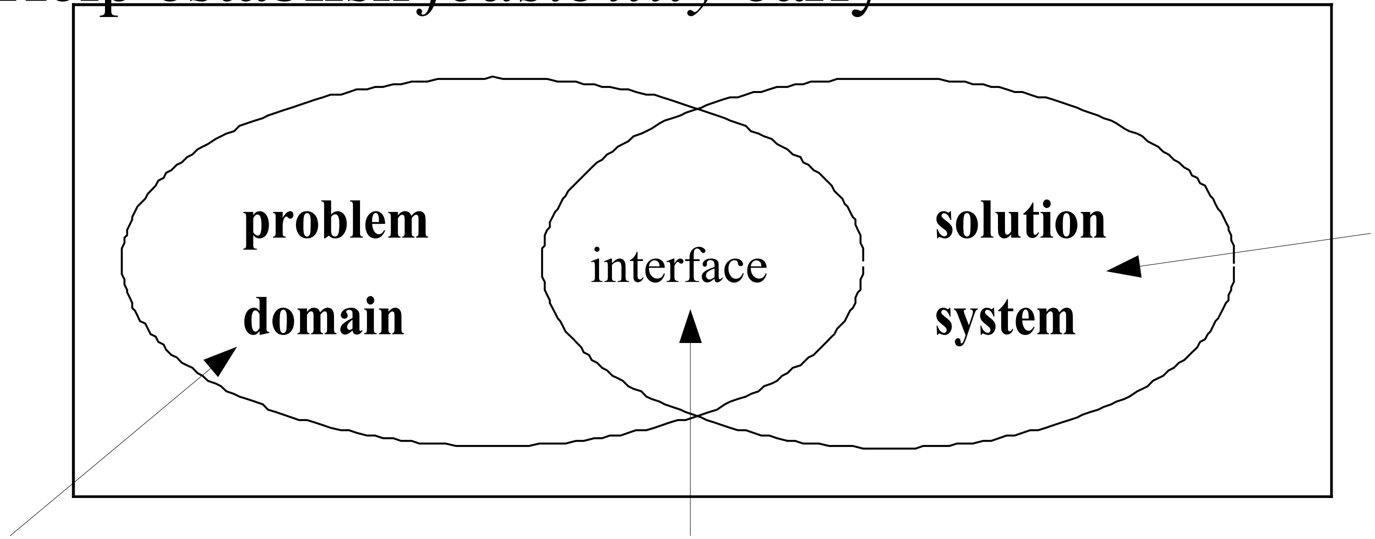


Consider the following conversation

- Gerhard, a senior manager at Contoso Pharmaceuticals, was meeting with Cynthia the new manager of Contoso's information systems (IS) development group. "*We need to build a chemical-tracking information system for Contoso. The system should let us keep track of all the chemical containers we already have in the stockroom and in individual laboratories. That way, maybe the chemists can get what they need for someone down the hall instead of buying a new container from a vendor. This should save us a lot of money. Also, the Health and Safety Department needs to generate some reports on chemical usage for the government. Can your group build this system in time for the first compliance audit five months from now?*"
- Are the above requirements ? Are they enough the build the system ?

Goal of elicitation

- Purpose is to get information about:
  - the domain model from which the requirements are written
  - the requirements from which system is developed
- Help establish *feasibility* early



- Hardware
- Software

- Domain properties
- Requirements

- Specification
  - description of Solution System needed to satisfy Problem Domain

Job of Requirements Analyst

- Understand problem from each stakeholder's point of view.
  - Review documentation
  - Observe current system
  - Questionaires and Interviews
  - Apprenticeship
- Extract the essense of the stakeholders' requirements
  - Interpreting stakeholders' descriptions of requirements
  - Building models
- Invent better ways to do the user's work
  - Ask why documented requirements are desired
  - Brainstorm to invent undreamed of requirements
- Negotiate a consistent set of requirements
- Record results in an SRS

Source of requirements

- Various sources for requirements
  - Clients
  - Pre-existing systems
    - not necessarily computer systems
  - Pre-existing documentation
  - Users (pre-existing and potential)
  - Competing systems
  - Domain experts
  - Documentation about interfacing systems
  - Standards and legislation
  - ...

Tasks performed as part of elicitation

- Plan for elicitation
  - why ?, who ?, when ?
- Examine project viability
  - is-it worth it ?
- Perform elicitation using elicitation techniques
  - interact to get the information
- Analyse results
  - understand information obtained
- Repeat as needed

Planning for elicitation

- Elicitation Plan should include:
  - Objectives
  - Strategies and processes
  - Products of elicitation efforts
  - Schedule and resource estimates
  - Risks

Planning for elicitation - Objectives

- Specify what the elicitation is for
- Examples
    - Validating marked data
    - Exploring use cases
    - Developing a detailed set of requirements for a system
    - etc

Planning for elicitation – Setting elicitation strategies and processes

- Approaches that will be used
- Generally a combination of approaches
- Possibly different approaches for different stakeholder types
- Examples:
    - Surveys (questionaires)
    - Workshops
    - Interviews
    - etc

Expected elicitation products

- Usually set of rough requirements
  - Written, audio, video notes
  - Documentation
- Depend on technique, e.g.
  - List of use cases
  - Detailed Software Requirements Specification (SRS)
  - Analysis of survey results
  - Performance attribute specification
  - ...
- Generally
  - Un-organized
  - Redundant
  - Incomplete
- Elicitation is incremental
  - Driven by information obtained
  - You always do a bit of elicitation – analysis – specification – verification at the same time

Requirements Specification Document

Specification Document

- A document that clearly and precisely describes, each of the essential requirements (functions, performance, design constraint, and quality attributes) of the software and the external interfaces. Each requirement being defined in such a way that its achievement is capable of being objectively verified by a prescribed method; for example inspection, demonstration, analysis, or test

Requirements Specification Document

- Basis for agreement between customers and contractors or suppliers
- Elaborated from elicitation notes
- Must
  - allow a rigorous assessment of requirements before design can begin
  - reduce later redesign
  - provide a realistic basis for estimating product costs, risks, and schedules
  - provide a basis for ensuring that a solution meets the original requirements

Requirements Specification Document

- Specifications are intended to a diverse audience
  - Customers and Users
    - for validation, contract, ...
  - Systems (Requirements) Analysts
  - Developers, Programmers
    - to implement the system
  - Testers
    - to check that the requirements have been met
  - Project Managers
    - to measure and control the project
- Different levels of detail and formality is needed for each audience
- Different templates for SRS (e.g. IEEE 830)

Planning for elicitation – Schedule and resource estimates

- Identify development and customer participants in various elicitation activities
- Estimate of effort for elicitation
- Scheduling

Planning for elicitation – Identifying elicitation risks

- Factors that could impede completion of elicitation activities
  - e.g. hostile stakeholders
- Severity of each risk
- Mitigation strategy for each risk

Examine Project Viability

- Does-it make good business sense ?
  - It's very difficult to cancel a project once started
- Based on:
  - product's purpose
  - business advantage,
  - costs vs. benefits,
  - Feasibility,
  - Scope,
  - required resources,
  - requirements constraints, and risks.

Examine Project Viability

- Purpose
  - *What does the product do*?
  - Highest-level customer requirement.
  - Business need.
  - All other requirements must contribute in some way to the purpose.

Examine Project Viability

- Business Advantage
  - *Why build the product?*
  - The purpose of the product should be not only to solve the problem, but also to provide a business advantage.
  - How will the product help the work?

Examine Project Viability

- Cost vs. Benefits
  - *How much will the product help our work?*
  - *How much will it cost to develop and operate the product?*

Examine Project Viability

- Feasibility
  - Technical feasibility
    - Need for measureable requirements
    - *Does the organization have the the skills needed to build and operate the product ?*
  - Economic feasibility
    - *Does the organization have the resources to construct the product ?*

Examine Project Viability

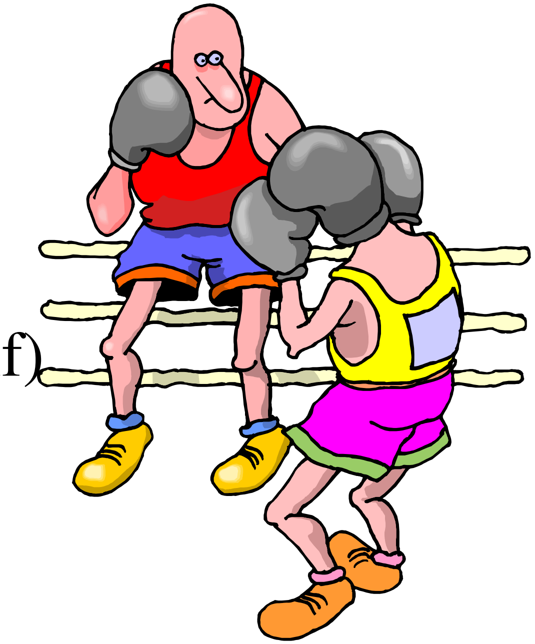
- Scope : product's purpose and the system's boundaries
    - *How much of the work will be done by the system-to-be-developed ?*
    - *How much of the work will be done by adjacent systems ?*
- Information needed for cost and time estimates.

Examine Project Viability

- Required Resources
  - *What are the required resources, i.e., money, time, and personnel ?*
  - *How do they compare with available money, time, and personnel ?*
- If the latter are smaller than the former, we should not even start the project.

Examine Project Viability

- Requirements Constraints: *Are there constraints that will restrict the system's requirements or how these requirements are elicited ?*
  - solution constraints:
    - mandated designs,
    - mandated adjacent systems, and
    - mandated COTS (commercial off-the-shelf)
    - components,
  - time constraints,
  - budget constraints.



Examine Project Viability

- Project Risks
  - *Are there any high-probability or high-impact risks that would make the project infeasible ?*
  - Include:
    - absense of clear purpose,
    - little or no agreement on requirements
    - unmeasureable requirements,
    - rapidly changing requirements,
    - etc.

Elicitation Problems

- According to Dean Leffingwell and Don Widrig:
  - The “**Yes, But**” syndrome stems from human nature and the users inability to experience the software as they might a physical device.
  - The “**Undiscovered Ruins”**, the more you find, the more you realize still remain
  - “**User and Developer**” syndrome reflects the profound differences between the two, making communication difficult.
  - “**The sins of your predecessors**” syndrome where marketing (user) and developers do not trust each other based on previous interactions, so marketing wants everything and developers commit to nothing.

Elicitation Problems
The “Yes, But” Syndrome

- First time users see the system the first reaction is either, “wow this is so cool” or “*Yes, but*, hmmmmm, now that I see it, what about *this*…? Wouldn’t it be nice …?
- Users reaction is simply human nature
- Need to employ techniques that get the “yes, buts” out early.
- Anticipate that there will be “yes, buts” and add time and resources to plan for feedback.
- Tends to be User Interface centric, these tend to be the touch points of the system by the users.

Elicitation Problems

The “Undiscovered Ruins” Syndrome

- Teams struggle with determining when they are done with requirements elicitation.
  - Is done when all the requirements are elicited or have they found at least enough?
  - Like asking an archeologist “how many undiscovered ruins are there?”

- First scope the requirements elicitation effort by defining the problem or problems that are to be solved with the system.
- Employ techniques that help find some of those ruins and have the stakeholders buy-into the requirements.

Elicitation Problems

The “User and the Developer” Syndrome

Characteristic	Response
• Users do not know what they want, or they know what they want but cannot articulate it.	• Recognize and appreciate the user as domain experts; try different techniques.
• Users think they know what they want until developers give them what they said they wanted.	• Provide alternative elicitation techniques earlier; storyboard, role playing, prototypes, and so on.
• Analysts think they understand user problems better than users do.	• Put the analyst in the users place. Try role playing for an hour or a day.
• Everybody believes everybody else is politically motivated.	• Yes, its part of human nature, so lets get on with the program.

Elicitation Problems

The “Living with the Sins of your Predecessors” syndrome

- Like it or not your users (marketing) and developers remember what happened in the past.
  - Quality programs that promised things would be different.
  - The last project where requirements were vague and/or were delivered short of expectations.
- The team “unilaterally” cut important features out of the last release.
- Need to build trust, slowly. Do not over commit to features, schedule, or budget.
- Build success by delivering highest priority features early in the process.

Elicitation Problems

Social and organizational factors

"No system is an island unto itself "

- All software systems exist and are used within a particular context (*technical AND social*)
- Social and organizational factors often dominate the system requirements
- Determining what these are can be difficult and time-consuming
  - developers are (usually) outsiders
  - people don't always tell the truth
  - awareness of one's own "culture" can be hard

Elicitation Problems

Social and organizational factors

These factors tend to cut across all aspects of the system:

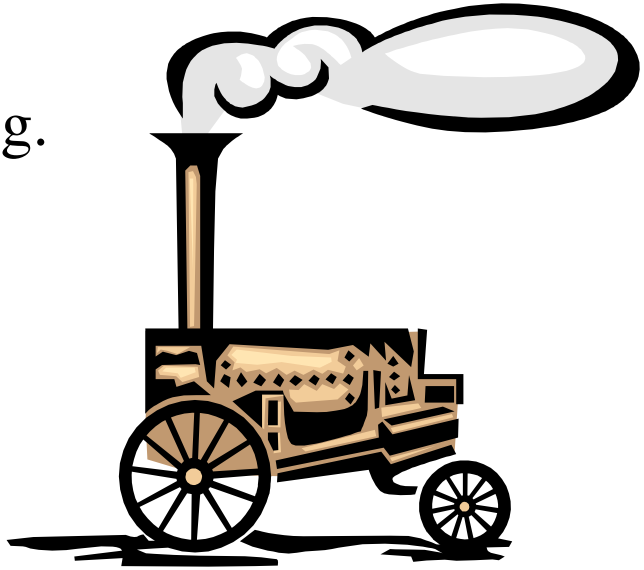
- *e.g., a system that allows senior managers to access information directly without going through middle managers*
  - interface must be simple enough for senior managers to be able to use
  - middle managers may feel threatened or encroached upon, be resistant to new system
  - lower-level users may concentrate on activities that impress senior managers, which is not necessarily what they ought to be doing
  - users may not like "random spot checks"; may devise ways of hiding what they're doing

Elicitation Techniques

- Various Techniques
  - Background reading
  - Brainstorming
  - Discourse analysis
  - Ethnography
  - Interviewing
  - Joint Application Design (JAD)
  - Questionnaires
  - Task observation
  - Use cases and Scenarios
  - Prototyping

Elicitation techniques
Analysis of Existing System

- When building a “new & improved” version of an existing system
- Important to know
  - What is used, what isn't, what's missing.
  - What works well, what doesn't.
  - How the system is used, how it was intended to be used, what new ways we want it to be used.



Elicitation techniques
Existing System

- Why analyze existing system?
  - Users may become disillusioned with new system if it is too different or doesn't do what they want. 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 don't currently work well).
  - To find out which "legacy" features can/can't be left out.

Elicitation techniques
Analysis of existing systems (1)

1. Review available documentation -- user docs, development docs, requirements docs, internal memos, change histories, etc.

   - *Of course, often these are out of date, poorly written, wrong, etc., but it's a good starting point.*

2. Observation -- Go out into the field, and observe the "IT specialists in the mist".

   - *Ideally, you are silent observer, at least initially. Otherwise, you risk getting a non-standard view of what is usually done. Later on, you can start asking questions OR interview people OR use questionnaires.*

Elicitation techniques

Analysis of existing systems (2)

3. Questionnaires -- Based on what you know now, circulate some questionnaires.
   - Useful when:
     - answers to questions need to be compared or corroborated or
     - large number of users and you want answers to specific questions.
       - "*How often do you use feature XXX?*"
       - "*What are the three features you would most like to see?*"
4. Interviews – after getting a better idea of what is to be done and have some good questions that require detailed answers.
   - You won't be wasting other people's time, or your own. This is very labour intensive!

Elicitation techniques
Interviews

- 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

Elicitation techniques
Interviews

- Process

1. Planning and Preparation
2. Interview session
3. Consolitation of information
4. Follow-up

Interviews – Planning and Preparation

- Important to plan and prepare interviews
  - Set goals and objectives for the interview
  - Prepare questions
  - Acquire background knowledge of the subject matter
  - Organize the environment for conducting an effective interview

Interviews

- Interview groups of people together to get *synergy*
    - Users cannot think of everything they need when asked individually, but will recall more requirements when they hear others' needs.
    - This interaction is called synergy, the effect by which group responses outperform the sum of the individuals' responses.

Common interviewing mistakes

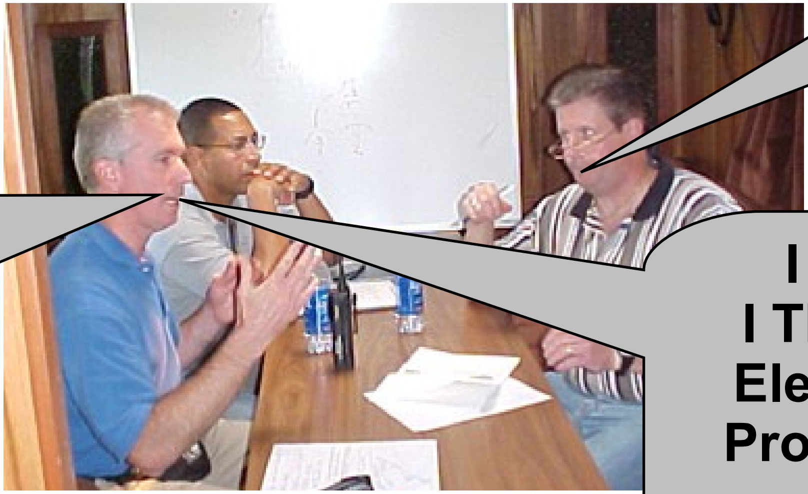
1. Not interviewing all of the right people.
   - Different points of view of stakeholders
2. Asking direct questions too early.
   - e.g., Designing a transportation system:
     - *How many horsepower do you need ?* (direct)
     - *How many people ? How far ? How fast ?* (indirect)
   - e.g., Camera 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

3. Interviewing one-at-a-time instead of in small groups.
   - More people might help get juices flowing as in brainstorming.
   - Reduces spotlight on individuals (may produce more interesting answers).
4. Assuming that stated needs are exactly correct.
   - Often users don't know exactly what they want and order "what he's eating".
   - Need to narrow what is asked for down to what is needed.

Common interviewing mistakes

5. Trying to Convince Stakeholders the *You* Are Smart – Wrong Place to Do That!
   - Instead Take Every Opportunity to Show You Think the Stakeholder is Smart
   - Contrast These Two Cases



My Elevators Are Too Slow!

I See. Tell Me Why You Feel They Are Too Slow.

I Don’t Think So. I Think You Have an Elevator Throughput Problem, not a Speed Problem

Alan M. Davis – Just enough requirements management

Elicitation Techniques - Brainstorming

- When to use ?
  - Few or too many ideas
  - Early on in a project particularilyy when:
    - Terrain is uncertain
    - There little expertise for the type of applications
    - Novel system requiring lot of innovation

Elicitation Techniques – Brainstorming Participants

- Any stakeholder, e.g.
  - developers,
  - domain experts,
  - end-users,
  - clients,
  - ...
- “Ideas-people” - special team of people in a company usually attend brainstorming sessions

Elicitation Techniques
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. Invite back most creative to multiple sessions.
- Encourage creativity
  - Choose good, provocative project name.
  - Choose good, provocative problem statement.
  - Get a room w/o distractions, but with good acoustics, whiteboards, coloured pens, provide coffee/donuts/pizza/beer
  - Provide appropriate props/mock-ups (e.g., ComfyCrate)

Elicitation Techniques Brainstorming - Roles

- 1. Scribe
  - Write down all ideas (may also contribute)
  - May ask clarifying questions during first phase, but not critical questions.
- 2. Moderator/leader -- Two schools of thought:
  - (a) Traffic cop -- enforces "rules of order", but doesn't throw his/her weight around otherwise.
  - (b) Agent provocateur - Traffic cop and 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.

Part I -- The Storm

Brainstorming Process

- Goal is to generate as many ideas as possible.
- Quantity, not quality, is goal at this stage.
- Look to combine or vary ideas already suggested.
- No criticism or debate is permitted. Don't want to inhibit participants.
- Participants understand nothing they say will be held against them later on.
- Scribe write down all ideas where all can see
  - e.g., whiteboard, paper taped to wall
  - ideas do not leave the room
- Wild is good. Feel free to be gloriously wrong.

Brainstorming Process

Part 2 -- The Calm

- Go over the list. Explain ideas more carefully.
- Categorize into "maybe" and "no" by pre-agreed consensus method.
  - informal consensus, 50%+1 vs. "clear majority", who has vetoes.
- Be careful about time.
  - Meetings (esp. if creative or technical in nature) tend to lose focus after 90 to 120 minutes. Take breaks or reconvene later.
- Review, consolidate, combine, clarify, expand.
- Rank the list by priority somehow; choose a winner.

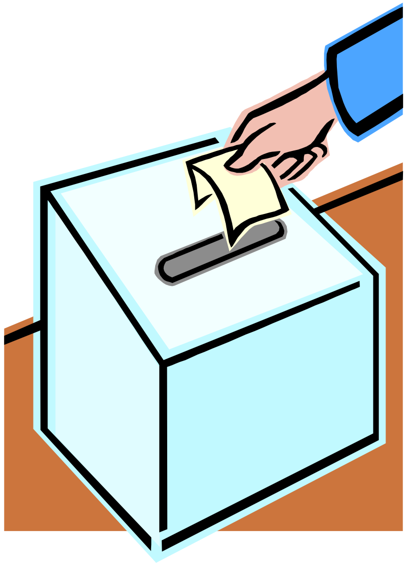
Elicitation Techniques Brainstorming - Blending ideas

- Apply acceptance criteria (which tend to be ignored in first step) to ideas.
- Rank accepted ideas.
- Select top k for voting treatment.

Elicitation Techniques

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 thereof 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 thereof with smaller j.

Elicitation Techniques
JAD – Joint Application Design

- Developed at IBM in the 1970s;
  - lots of success stories.
- "Structured brainstorming", IBM-style:
  - full of structure, defined roles, forms to be filled out, ...
- Two major "steps", three phases each, and six (human) roles to be played

JAD – Joint Application Design

Four main tenets of JAD:

1. Effective use of group dynamics
   - *facilitated and directed group sessions to get common understanding and universal buy-in*
2. Use of visual aids
   - *to enhance understanding, e.g., props, prepared diagrams*
3. Defined process
   - *i.e., not a random hodgepodge*
4. Standardized forms for documenting results
   - e.g., *use of LCD approach*

JAD – Joint Application Design

- 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,
  - designing a solution,
  - defining requirements
- JAD session may last few days

JAD – Joint Application Design

Main steps:

1. Pre-session Planning
   - Evaluate project
   - Select JAD participants
   - Create preliminary agenda
   - Determine deliverables for the working session
   - Enable participants to prepare for the session

JAD – Joint Application Design

2. Pre-work

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

JAD – Joint Application Design

3. Working Session

- Session leader set-up stage (welcome participants, present task to be discussed, establish rules, what is on/off topic,...)
- Generate common understanding
- Achieve consensus on decisions
- Generate ownership of results
- Create the deliverables (using standard JAD forms)
- Identify open issues and questions

JAD – Joint Application Design

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

JAD – Joint Application Design

Roles

1. Session leader

   - *organizer; facilitator; JAD expert; good with people skills; enthusiastic; sets tone of meeting.*

2. Analyst

   - *scribe ; produces official JAD documents; experienced developer who understands the "big picture"; good philosopher/writer/organizer.*

3. Executive sponsor

   - *manager who has ultimate responsibility for product being built; provides "strategic insights" into company's high-level goals/practices; later on, makes "executive decisions" as required.*

JAD – Joint Application Design

Roles

4. 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.*

5. Information system representative

   - *technical expert on ISs; helps users think big, know what's easy/hard/cheap/expensive; mostly there to provide information rather than make decisions.*

6. Specialist

   - *technical expert on particular narrow topic, e.g., security, application domain (travel agencies), law, UI issues.*

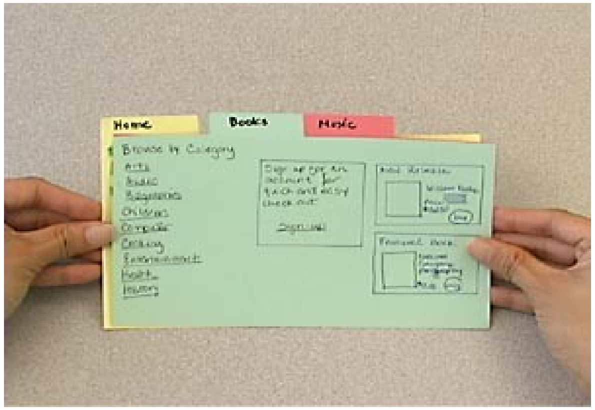
Elicitation Techniques
Prototyping

- 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 see it* " attitude.
  - Effective in addressing the "Yes, But" and the "Undiscovered Ruins" syndromes.
- Prototyping is effective to resolve uncertainties early in the development process.
  - focus prototype development on these uncertain parts.

Elicitation Techniques

Prototyping

- Different types of prototypes
  - Horizontal Prototype (focus on one layer – e.g. user interface)
  - Vertical Prototype (a slice of the real system)
  - Throwaway Prototype (not to be used in production system)
  - Evolutionary Prototype (do evolve as the production system)
- Different ways to realize a prototype
  - Using paper and pencil
    - prototype on index card
    - Storyboard
  - Using high-level languages (e.g. Visual Basic, Delphi, Prolog)
  - Using scripting languages (e.g. Perl, Python)
  - Using animation tools (e.g. Flash/Shockwave)
  - ...



Elicitation Techniques
Prototyping

- Risks of prototyping
  - 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

Elicitation Techniques
Use Cases

- 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
  - *People*
  - *Other software systems*
  - *Hardware devices*
- Used to describe users tasks in relation with a system

Elicitation Techniques
Use Cases - Scenarios

- Specific instance of a use case
  - Path through a use case
- A use case includes:
  - 1 primary scenario (*normal course of events*)
  - 0 or more secondary scenarios (*alternative/exceptional course of events*)
    - Variations of primary scenario

Elicitation Techniques
Use Cases Documentation

- Different *templates* for use cases
- Elements
  - Identifier: unique label for use case used to reference elsewhere
  - Name: succinctly state user task
    - Suggested form “verb object” (e.g. Order a product)
  - Authors: people who *discovered* use case

Elicitation Techniques
Use Cases Documentation

- **Preconditions**: what need to be satisfied before use case can begin
- **Postconditions**: state of system after successful completion of use case
  - Minimal guarantee: state of system at use completion regardless of outcome
- **Primary actor**: initiate the use case
- **Participants**: other actors involved in use case
- **Goal**: short description of expected outcome from actors'point of view
- **Related requirements**: identifiers of functional and non-functional requirements linked to the use case
- **Related use cases**: identifiers of use cases related to this use case
  - Specify relationship: e.g.
    - supposes use case UC2 has been successfully completed

Elicitation Techniques
Use Cases Documentation

- Related requirements: identifiers of functional and non-functional requirements linked to the use case
- Related use cases: identifiers of use cases related to this use case
  - Specify relationship: e.g.
    - suppose use case UC2 has been successfully completed
    - alternative to use case UC3
    - ...

Elicitation Techniques
Use Cases Documentation

- Description of events (scenarios)
  - Different use cases description formats
    - Narrative form: paragraph description may include only primary scenario or multiple scenarios.
    - Single column form: use case as a linear sequence may include only primary scenario, or primary scenarios with secondary scenarios
    - Multiple column form: different column for actors and system.

Use Cases Description

- Narrative form

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 Cases Description

- Single column

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 Cases Description

Multi columns

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 Cases Documentation

- Essential use cases (Constantine & Lockwood):
  - abstract, technology free, implementation independents
  - 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

Elicitation Techniques
UML Use Case diagram

- Specify use cases and relations between use cases
  - Inclusion: «*include*»
  - Extension: «*extend*»
  - Generalization – inheritance
    - applies to *actors* and *use cases*
- Used for
  - structuring use cases specification
  - giving context

UML Use Case diagram

«include»

- Include behavior of one use case in another
- Avoids repetition of the same behavior
- 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

UML Use Case diagram

«extend»

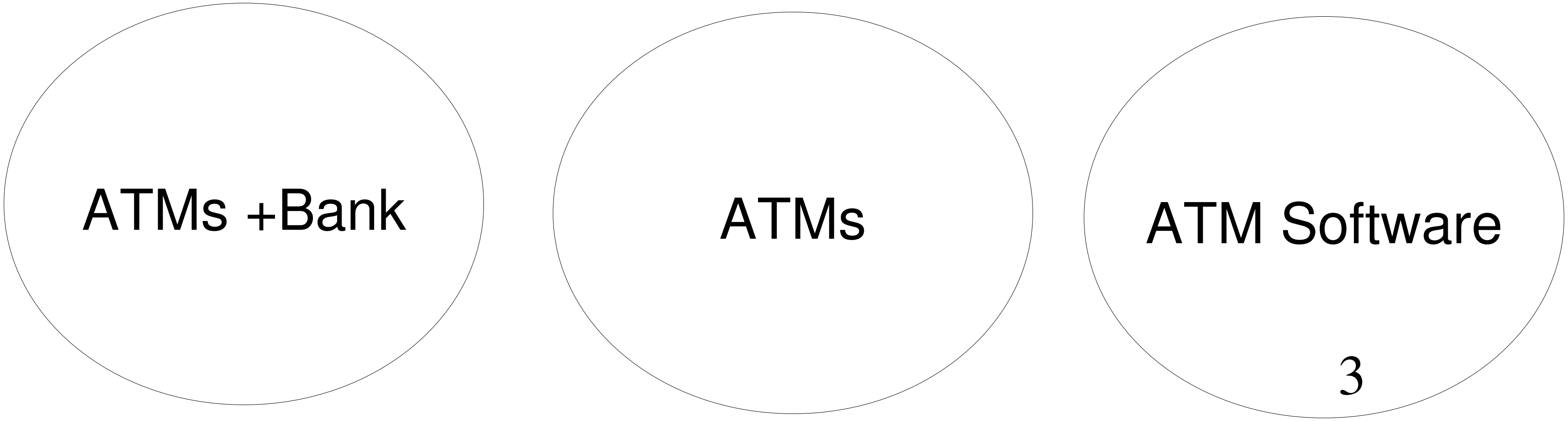
- Insert new behavior in an existing use case
- Base use case has hooks where it can be extended
- Unlike «include», base case should flow independently of extension
- Use sparingly: there is disagreement over the semantics.

Use Cases Development

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

Use Cases Development

Example for an ATM system



ATMs +Bank

ATMs

ATM Software

1

- Customer

2

- Customer
- Bank

3

- Card reader
- Cash dispenser
- Printer
- Bank

Use Cases Development

2. Identify actors

   - *Who interact 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 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 actors generalization

Use Cases Development

Chose actors names carefully

- Actors give value, get value from system or both
- Should reflect *roles* rather than actual people
- 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 Cases Development

Example ATM with scope 2

- *Who or what uses the system?*
  - Bank Customer
- *Who installs the system?*
  - Installation technician
- *Who or what starts and shuts the system?*
  - Administrator
- *Who maintains the system?*
  - Administrator
- *Who or what gets and provides information to the system?*
  - Bank
- *Does anything happen at a fixed time?*
  - Weekly reports

Use Cases Development

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: *enter form 104-B, complete approval window, get approval from immediate supervisor in Accounting Department*
- Develop a brief description in narrative form

Use Cases Development

Example actor Customer

- Overall Goal
  - *Access account 24h/7 for regular banking operations (withdrawal, deposit, statement, ...) in a timely an secure way.*
  - To be refined in sub-goals
- From goals we can identify tasks
  - Withdraw Cash
  - Make deposit
  - Print statement
  - ....

Use Cases Development

Description of use case *Withdraw Cash*

The Customer identifies himself/herself to the system, then select cash withdrawal operation. The system ask for the amount to withdraw. The Customer specifies the amount. The system provides the requested amount. The Customer takes the amount and leave.

Use Cases Development

4. Identify *precondition*
   - Example Withdraw Cash
     - *System is on operation, Cash is available*
5. Definition of primary scenario
   - Happy day story where everything goes fine

Use Cases Development

6. Definition of secondary scenarios

- 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)

Use Cases Development

7. Structure use case diagram

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

Use Cases Development

- Approach is iterative
- System scope and boundaries may change as more information is known about actors, their goals and use cases.

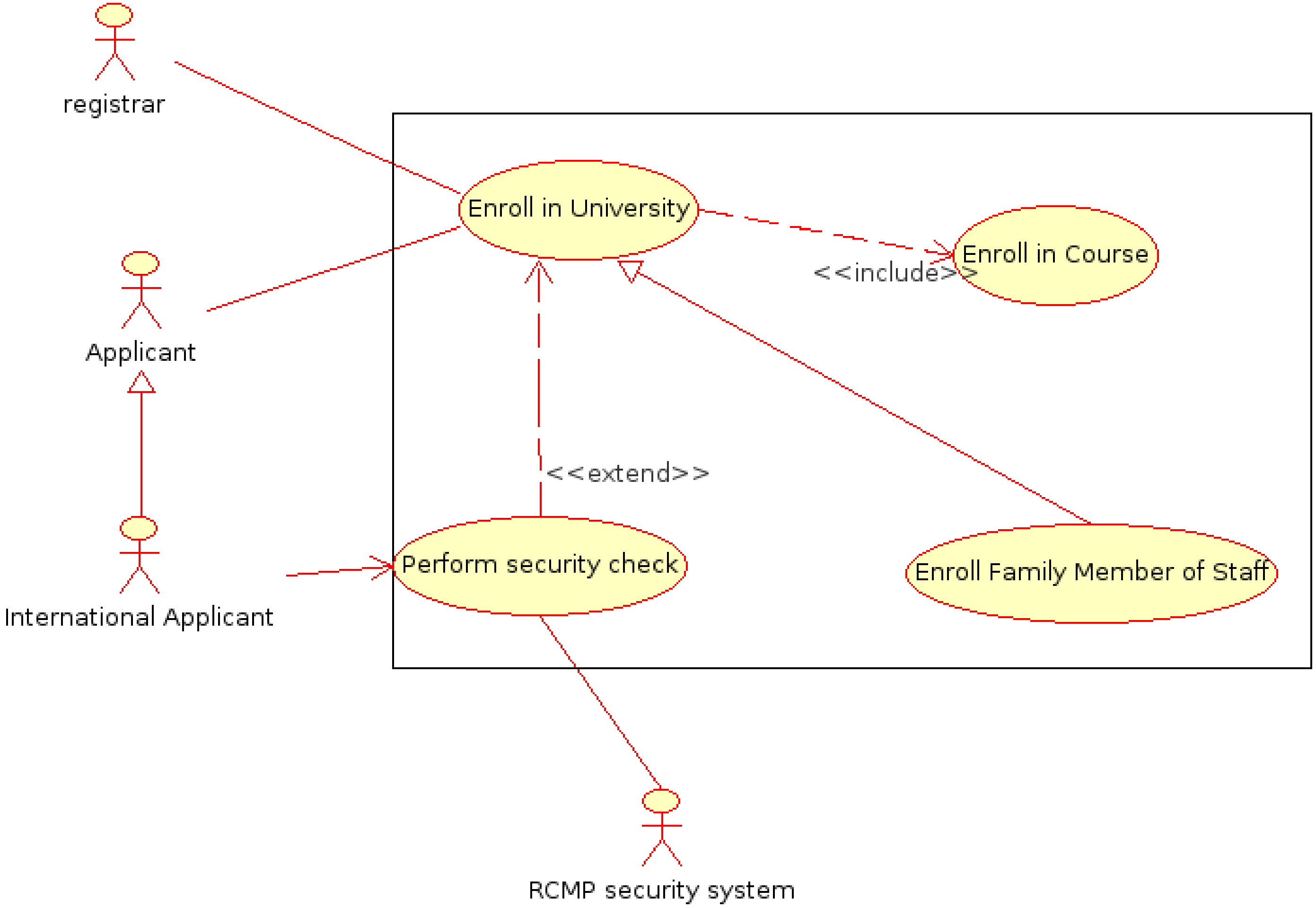
Use Cases Development

- 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 don't trace to any use case, consider whether you really need them.

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



Use Case Diagram

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.

Use Case Diagram

Course of Action:

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 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 enroll the student in courses via the use case UC 17 Enroll in Course.
12. The system prepares a bill for the applicant enrollment fees.

4.a The forms have not been adequately filled

.....

Use Case Diagram

Name: ***Perform Security Check***

Identifier: UC 34

At Extension Point XPCheck

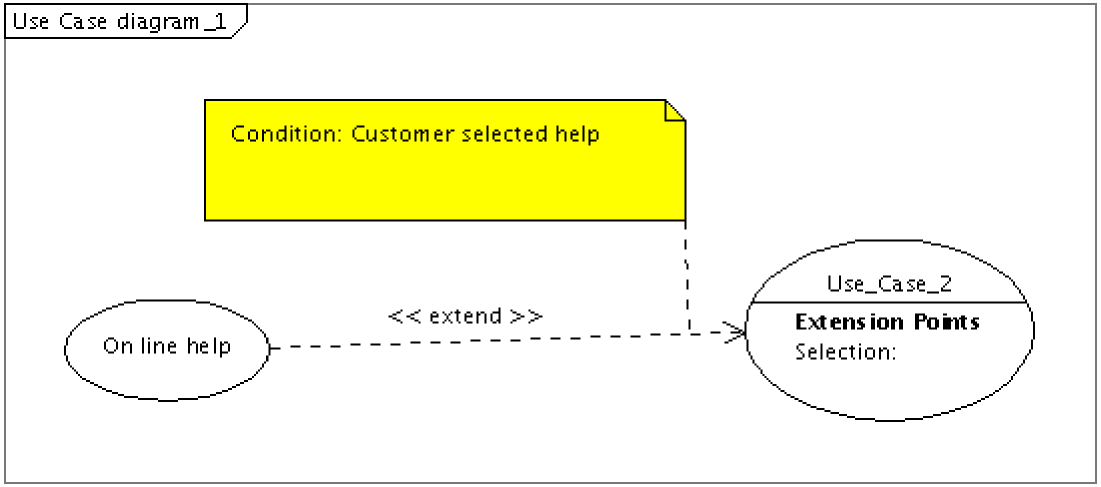
1. The registrar asks for security check results about applicant.
2. The system asks RCMP Security System for applicant security check results.
3. The RCMP Security System responds that applicant has been cleared.

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

...

Use Case Diagram

- Possibility to specify
  - extension points and
  - extension condition



Use Case Diagram

- Use Case generalization
  - Similar to Class generalization.
    - need to satisfy “is-a” relation
  - Inheriting use case can replace steps of inherited use case.
  - Semantics unclear: use with caution.

Name: ***Enroll Family Member of Staff***

Identifier: UC 20

Inherits From: UC 19

Use Case Diagram

Course of Action:

1. An applicant family member of staff 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.
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 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 enroll the student in courses via the use case UC 17 Enroll in Course.
12. The system prepares a bill for the applicant enrollment fees based on staff family members rate.

4.a The forms have not been adequately filled

.....

Use Cases Development - Rules of thumb

- 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 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

Use Cases Development -Rules of thumb

- Be careful not to under specify behavior 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 behavior for all possible inputs, both valid and invalid input

Use Cases Development - Rules of thumb

- 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: don't try to structure the use cases as nested functions with «includes»
- Avoid deep hierarchies with «includes»

Use Cases Development - 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*"

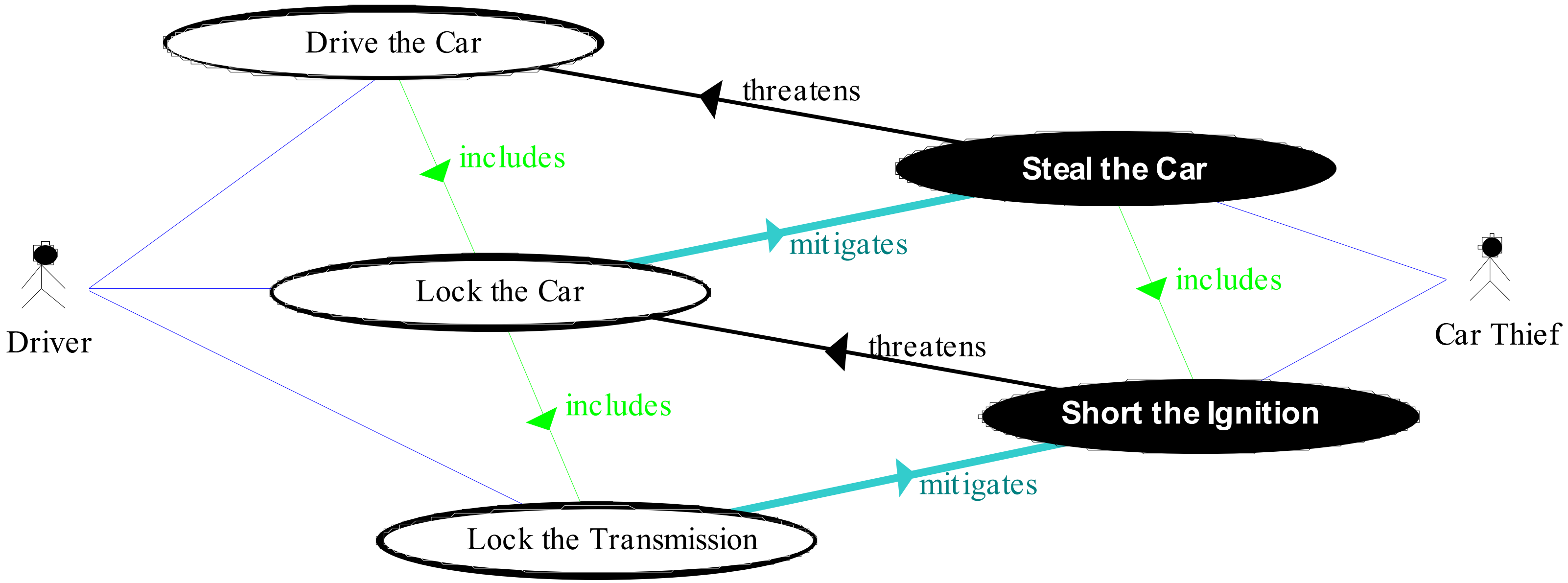
Use Cases Development - Advantages of use cases

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

Misuse Cases

- Introduced by Sindre and Opdahl (2000)
- Capture use cases that a system must be protected against
  - Use cases to avoid, e.g., security breach
- Misuse case goal is
  - desired Not to occur by the organisation, or
  - desired by a hostile agent (not necessarily human)

Misuse Cases

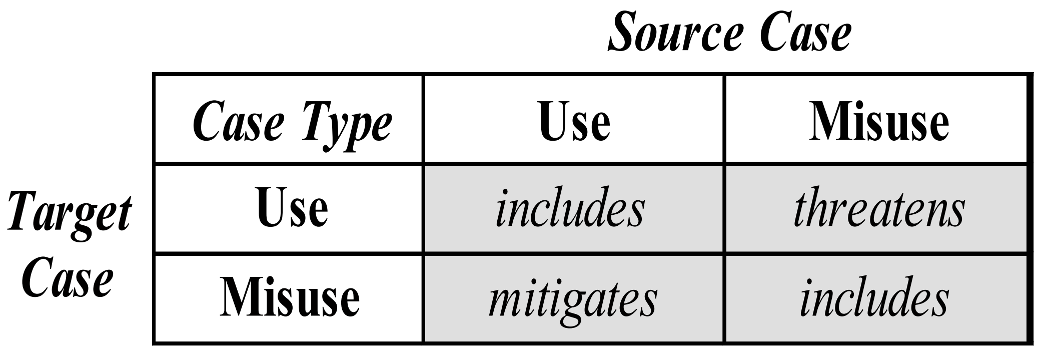


Use Cases for 'Car Security'

By Ian Alexander

- Actor is a Hostile Agent
- Bubble is drawn in inverted colours
- Goal is a ***Threat*** to System
  - Obvious Security Applications
  - Can be ***Mitigated*** by other use cases

Misuse Cases



		Source Case	
	Case Type	**Use**	**Misuse**
Target Case	**Use**	*includes*	*threatens*
	Misuse	*mitigates*	*includes*

Rule governing creation of relationships between Use and Misuse Cases

By Ian Alexander

- Other relations possible, e.g.
  - A misuse case can include a use case when legit actions can be used as part of the misuse

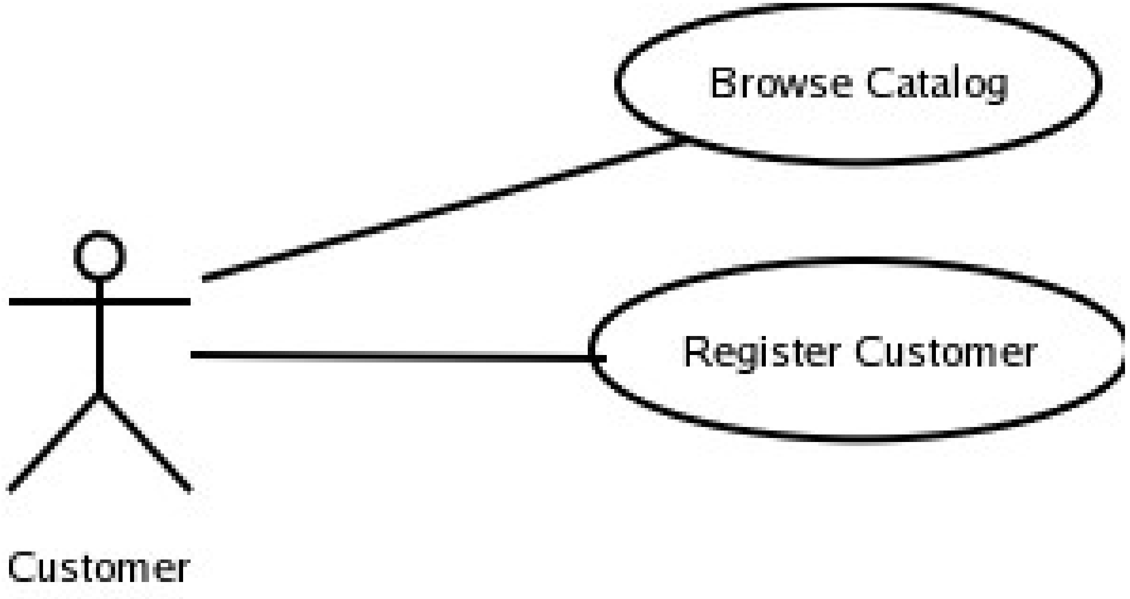
Misuse case - elicitation

- Start from a normal use case diagram then

1. Find mis-actors (hostile role)
2. Find misuse cases
   - Add relationships (threaten)
   - If needed elaborate with *scenarios*
3. Find new Use Cases or Exception scenarios to neutralise (mitigate) Misuse Cases.

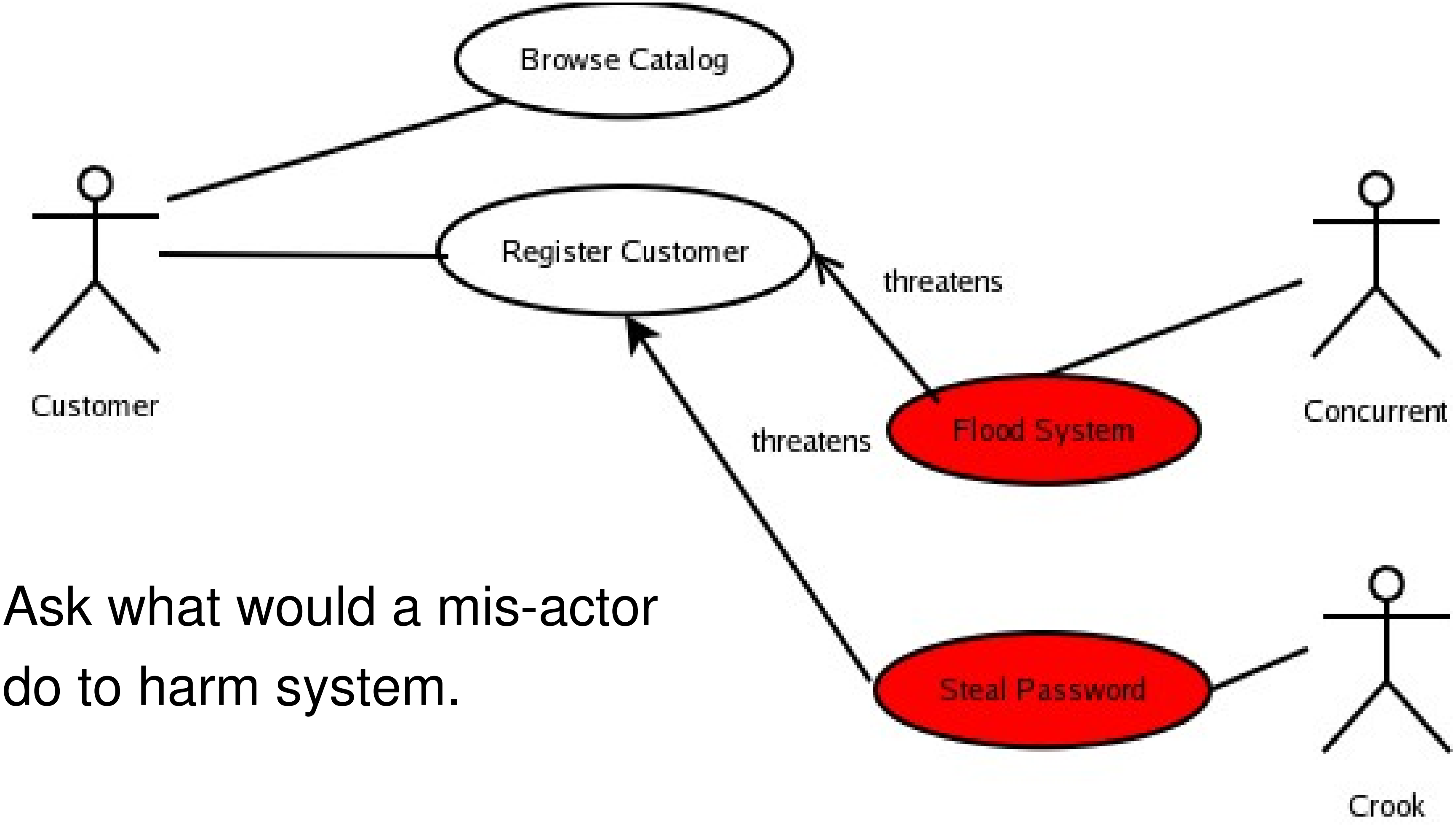
Misuse case – finding mis-actors

- Who might want either:
  - To harm the system, its stakeholders, or their resources intentionally, or
  - To achieve goals that are incompatible with the system's goals.



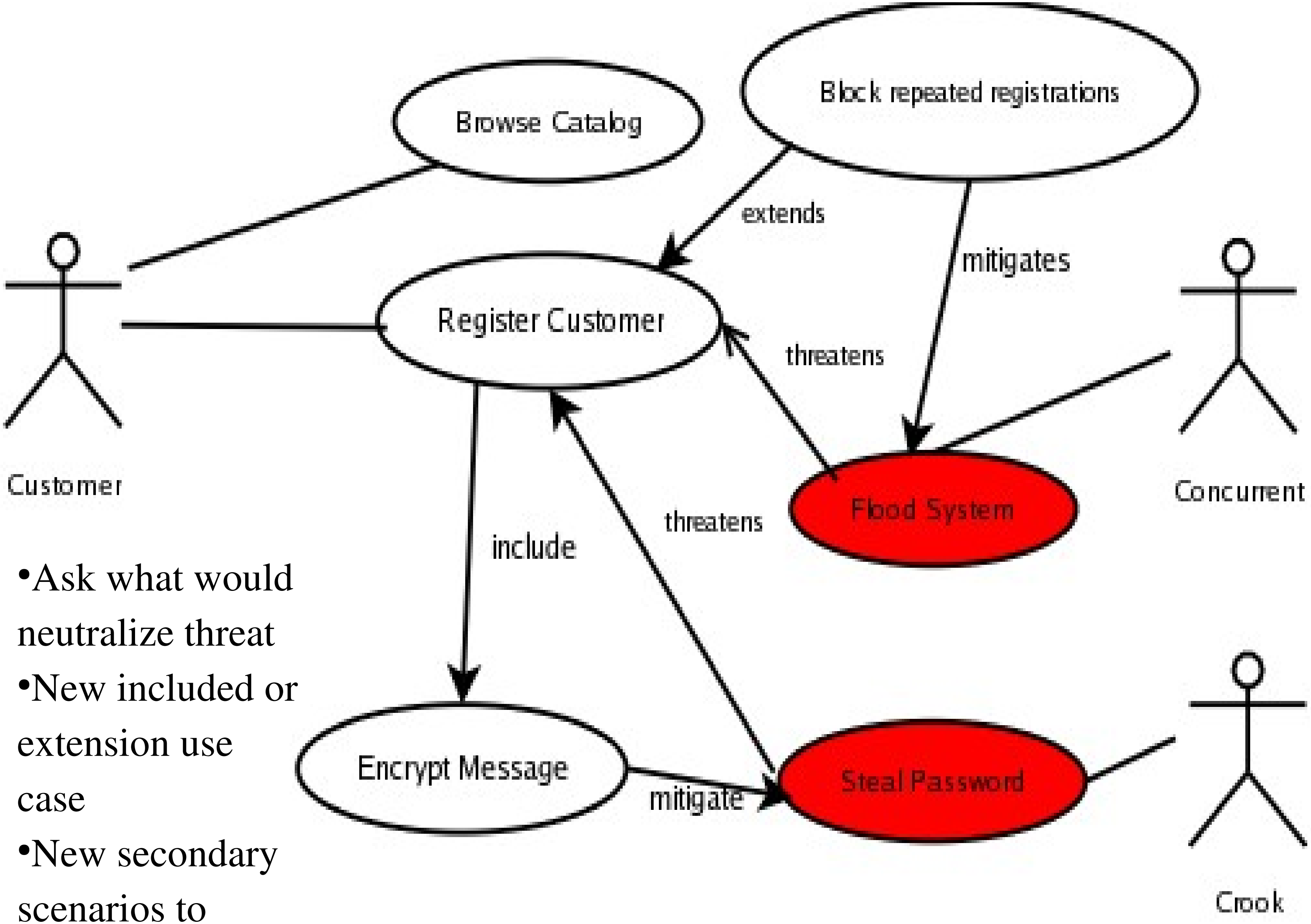
Misuse case – finding mis-use cases

- Express goals of mis-actors



Ask what would a mis-actor do to harm system.

Misuse case – answering threats



- Ask what would neutralize threat
- New included or extension use case
- New secondary scenarios to existing use cases

Misuse Cases

- Advantages – help early identification of
  - Threats and mitigations
  - Security requirements
  - Safety requirements
  - Exceptions that could cause system failure
  - Test cases

Misuse Cases

- Caution - mitigation tend to design solution when out of control
  - Goal should be find requirements
  - Some threats need solution in design
    - e.g. subsystems for alarms, cameras, etc
    - It's important to keep traceability between misuse cases and design solution in these cases