CptS 322- Software Engineering Principles I

Requirements

Instructor: Sakire Arslan Ay

Fall 2023

Washington State University

World Class. Face to Face.

What is a Requirement ?

- It is a statement describing
  1. an aspect of what the proposed system must do,
  2. a constraint on the system’s development.

  – In either case it must contribute in some way towards adequately solving the customer’s problem;
  – the set of requirements as a whole represents a negotiated agreement among the stakeholders.

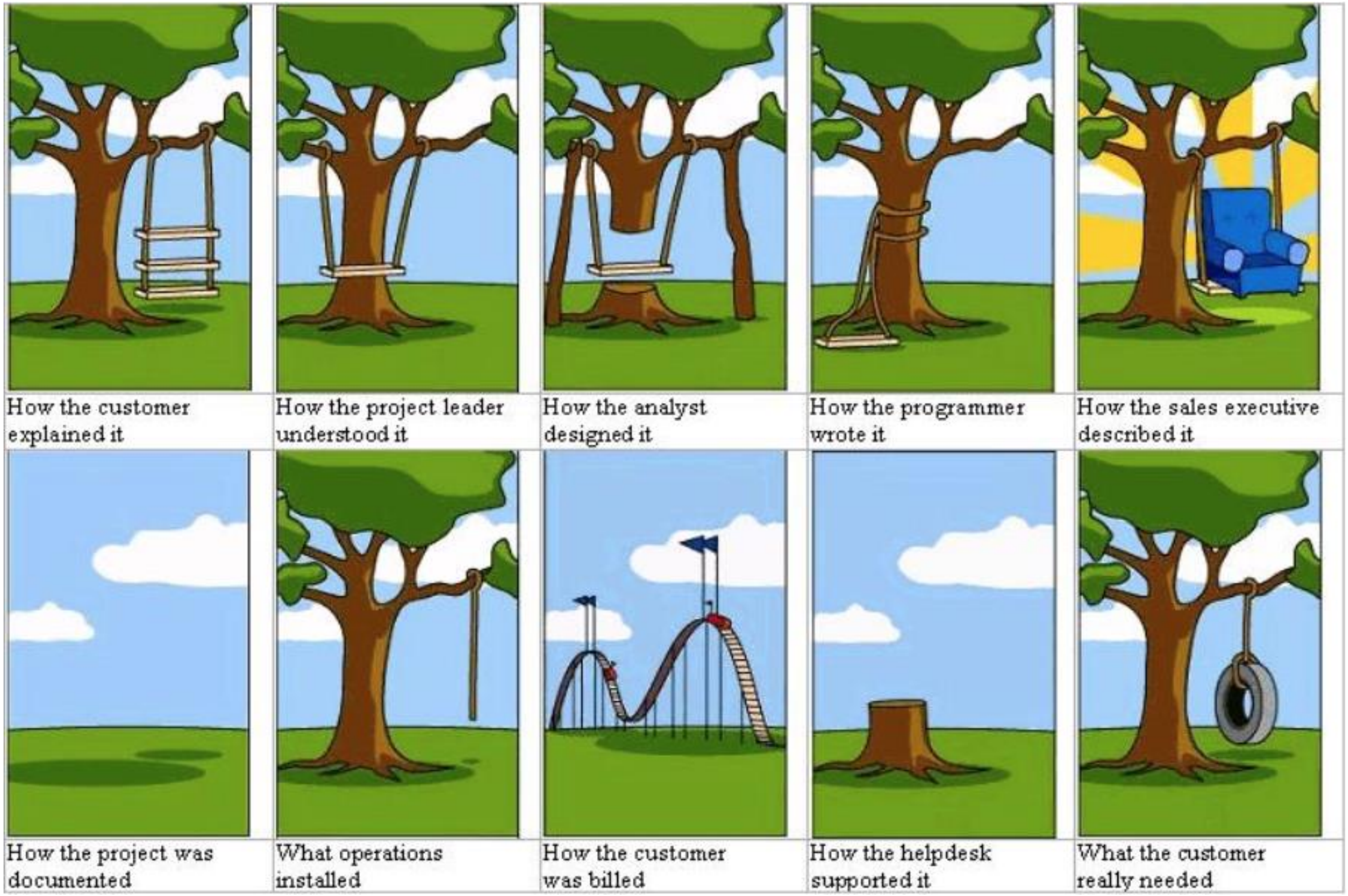
- Requirements Engineering is the process of establishing requirements.
- A collection of requirements is a *Software Requirements Specification* document. In Scrum this is the *Product Backlog*.

Software Requirements

- Requirements specify what to build:
  - tell “what” and not “how”
  - tell the problem, not the solution
  - reflect system design, not software design

Why Requirements?

- They help to:
  - Understand what is required of the software
  - Communicate this understanding precisely to all development parties
  - Control production to ensure that system meets specs (including changes)


How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Roles of Requirements

- Customers
  - show what should be delivered; contractual base
- Managers
  - a scheduling / progress indicator
- Designers
  - provide a spec to design
- Programmers
  - list a features to implement / output
- QA / testers
  - a basis for testing, validation, verification

Classifying Requirements
(the classic way)

- Functional vs. Non-functional Requirements

- Functional requirements
  - Describe what the system should do
  - Map inputs to outputs
  - Examples:
    - “All prerequisite requirements should be met in order to enroll in a class.”
    - “When a student enrolls in a course, it should show-up in their schedule.“

Functional Requirements

– What *inputs* the system should accept

– What *outputs* the system should produce

– What data the system should *store*

– What *computations* the system should perform

– The *timing and synchronization* of the above

Classifying Requirements
(the classic way)

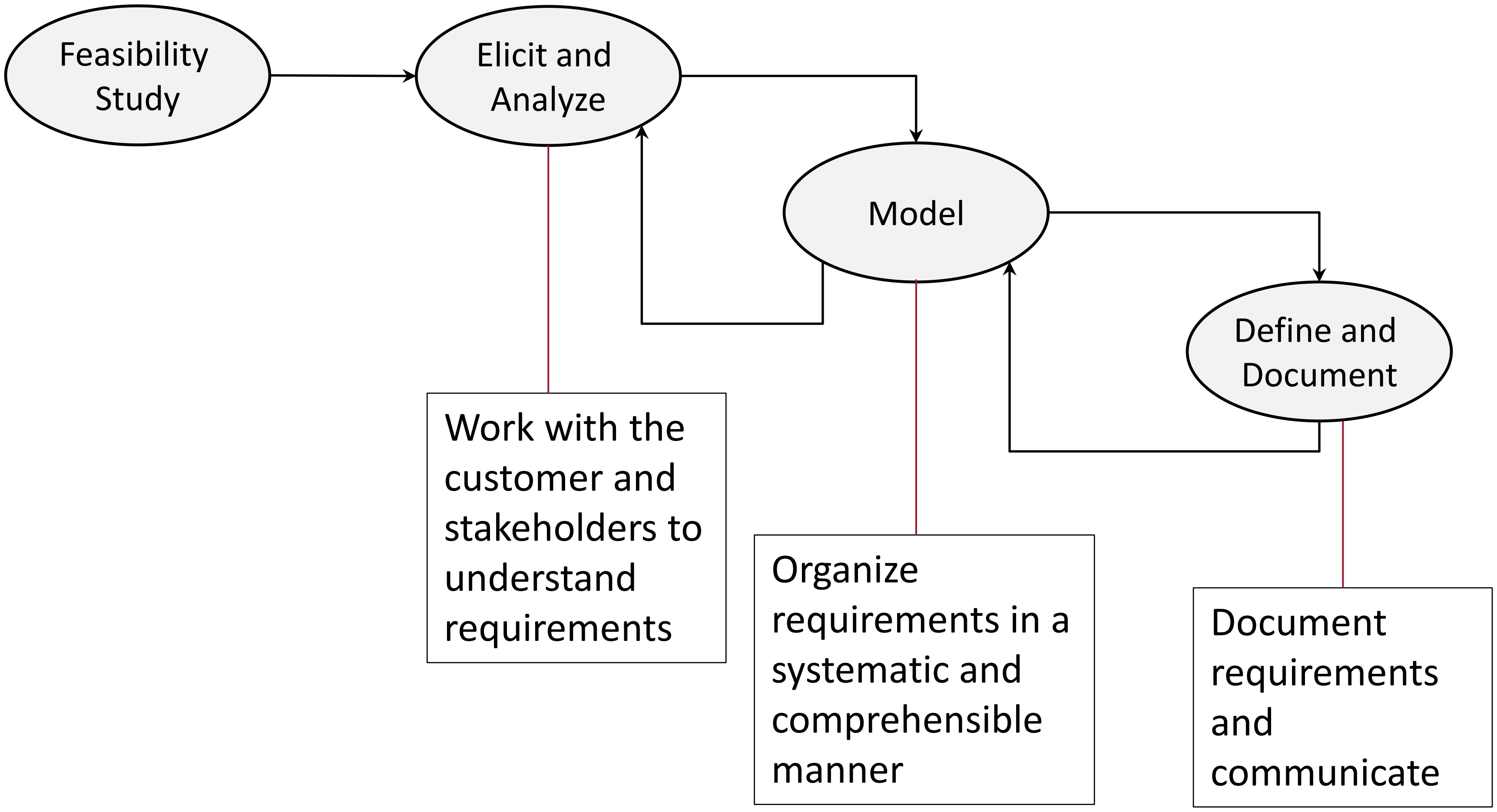
- Nonfunctional requirements : other constraints
  - Software quality requirements
    - Constraints on the design to meet specified levels of quality
    - Dependability, reusability, portability, scalability, performance, safety, security
    - Example: “The system shall not disclose any personal user information.”
  - Platform requirements
    - Constraints on the environment and technology of the system
    - Example: “The system should be compatible with all major browsers: Chrome, Firefox, IE/ Edge, Safari, etc.
  - Process requirements
    - Constraints on the project plan and development methods
    - Example: “Our deliverable documents shall conform to the XYZ process.”

Nonfunctional Requirements

- Constraints on
  - Response time
  - Throughput
  - Scalability
  - Reliability
  - Robustness
  - Availability
  - Security
  - Recovery from failure
  - Allowances for maintainability and enhancement
  - Allowances for reusability

- All must be verifiable

The Requirements Process


Feasibility Study
Elicit and Analyze
Model
Define and Document
Work with the customer and stakeholders to understand requirements
Organize requirements in a systematic and comprehensible manner
Document requirements and communicate

Determining Stakeholders and Needs

- Must determine stakeholders
  - Anyone who benefits from the system developed
  - E.g., who’s customer and who’s user ?
- Try to understand what their needs are
- Reconcile different needs/points of view

How to elicit requirements from users/stakeholders?

- Talk to the users, or work with them, to learn how they work.
- Ask questions throughout the process to "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

- Client interviews are the heart of the requirements analysis.
  - Allow plenty of time.
  - Prepare before you meet with the client.
  - Keep full notes.
  - If you do not understand, delve further, again and again.
  - Repeat what you hear

How <u>not to</u> gather requirements?

- Describe complex business logic or rules of the system.
- Be too specific or detailed.
- Describe the exact user interface used to implement a feature.
- Try to think of everything ahead of time. (You will fail.)
- Add unnecessary features not wanted by the customers.

Requirements should be Realistic and Verifiable

- Requirements must be realistic, i.e., it must be possible to meet them.
  - Wrong: “*The system must be capable of x (if no known computer system can do x at a reasonable cost*)”.

- Requirements must be verifiable, i.e., since the requirements are the basis for acceptance testing, it must be possible to test whether a requirement has been met.
  - Wrong: “*The system must be easy to use.*”
  - Right: “*After one day's training an operator should be able to input 50 orders per hour.*”

Good or bad requirements?

1. The system will enforce 6.5% sales tax on Washington purchases.
2. The system shall display the elapsed time for the car to make one tour around the track within 5 seconds, in hh:mm:ss format.
3. The product will never crash. It will also be secure against hacks.
4. The server backend will be written using PHP or Ruby on Rails OR Node.js.
5. The system will support a large number of connections at once, and each user will not experience slowness or lag.
6. The user can choose a document type from the drop-down list.

Reviewing Requirements

- **Each individual requirement should**
  - Have **benefits that outweigh the costs** of development
  - Be **important** for the solution of the current problem
  - Be expressed in a **clear and consistent** manner
  - Be **unambiguous**
  - Be **logically consistent**
  - Lead to a system of **sufficient quality**
  - Be **realistic** with available resources
  - Be **verifiable**
  - Be uniquely **identifiable**
  - **Not over-constrain the design** of the system

How do we Specify Requirements?

- User stories and use cases
- Prototype
- Feature list
- Paper UI prototype
- Formal specification

User Stories and Use Cases

- User Stories are short, simple descriptions of a feature told from the perspective of the person who wants the feature. They typically follow a simple template:
  - Example: Search and replace in a word processor

  "As a user, when I realize that I mis-capitalized a word everywhere in the document, I want to search for all occurrences of it and replace them with the corrected word."

- A Use Case is an expansion of a User Story -- sequence of actions that a user performs in order to complete a task

Actor actions	System responses
1. Choose the search/replace option	2. Display the search/replace dialog
3. Enter "search and replace keywords"	4. Highlight all occurrences of the search keyword on the document
…	…

User Stories

- Write on index cards (electronic versions)
  - meaningful title
  - short (customer-centered) description
- Focus on “what” not the “how”
- Uses client language
  - Client must be able to test if a story is completed (customer acceptance test)

Example: Word Processor

- Customer: “*I need a word processor software where I can create a new document, write text to the document, save the document, open a previously created document, search a particular term in the text, and replace the occurrences of a term with another.*”

- Analyze the customer’s statement and create some user stories

User Stories

Title: Create Document

Description: As a user, I can create a new document

Title: Open Document

Description: As a user, I can open an existing document

Title: Save Document

Description: As a user, I can save the current document

Title: Search and Replace

Description: As a user, I can search for the occurrences of a term or keyword and replace them with a different keyword.

User Stories

Title: Create Document

Description: As a user, I can create a new document

Title: Open Document

Description: As a user, I can open an existing document

If the document has not been saved yet, it should ask me to prompt the document name.

Title: Save Document

Description: As a user, I can save the current document

itle: Search and Replace

Description: As a user, I can search for the occurrences of a term or keyword and replace them with a different keyword.

Can I save the document with a different name?

User Stories

Title: Create Document

Description: As a user, I can create a new document

Title: Open Document

Description: As a user, I can open an existing document

Can the search be case sensitive?

Title: Save Document

Description: As a user, I can save the current document. If the document has not been saved yet, it should ask for the filename. I can also save a (existing) document with a different name.

Title: Search and Replace

Description: As a user, I can search for the occurrences of a term or keyword and replace them with a different keyword.

User Stories

Title: Create Document

Description: As a user, I can create a new document

Title: Open Document

Description: As a user, I can open an existing document

Title: Save Document

Description: As a user, I can save the current document. If the document has not been saved yet, it should ask for the filename. I can also save a (existing) document with a different name.

Title: Search and Replace

Description: As a user, I can search for the occurrences of a term or keyword and replace them with a different keyword. I can choose to make the search case-sensitive.

User Story?

Title: Use dropdown menu

Description: The user interface should have a drop-down menu in order to choose a previously used keyword.

(Generally) Not a user story (an implementation detail)

Title: Should be portable

Description: The word processor should be platform independent

In Scrum We Create User Stories

- *As a <type of user>, I want <some goal>.*
  - *“As a user, I want to open files by giving their name or by browsing.”*
- Include in the Product Backlog all user stories /use cases that you might possibly implement in your system.
- Modify and add to these user stories throughout the process.

The benefits of basing software development on user stories

- They can
  - Help to define the scope of the system
  - Be used to plan the development process
  - Be used to both develop and validate the requirements
  - Form the basis for the definition of test cases
  - Be used to structure user documentation

Use-Cases: Describing how the user will use the system

- A *use case* is a typical sequence of actions that a user 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 user
    - How users interact with this system when trying to achieve their objectives
    - It is one of the key activities in requirements analysis
  - 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.
  - Cover the *full sequence of steps* from the beginning of a task until the end.
  - Be written so as to be as *independent* as possible from any particular user interface design.
  - Only include actions in which the actor interacts with the system.
    - Not actions a user does manually nor internal actions of the system

User Stories and Use Cases

- Two possibilities:

1. Create User Stories and Use Cases before Sprint 1 for every item in the Product Backlog
2. Create a Use Case for a User Story in the sprint just before implementing it

Use Case Example

Name	**Search and Replace**
Users	All users
Rationale	While editing a document, many users find that there is text somewhere in the file being edited that needs to be replaced, but searching for it manually by looking through the entire document is time-consuming and ineffective. The search-and-replace function allows the user to find it automatically and replace it with specified text. Sometimes this term is repeated in many places and needs to be replaced. At other times, only the first occurrence should be replaced. The user may also wish to simply find the location of that text without replacing it.
Triggers	The user select the “Search/Replace” option
Preconditions	A document is loaded and being edited.

Use Case Example (cont.)

Actions	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Alternative paths	

Use Case Example (cont.)

Actions	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Alternative paths	1. In Step 3, the user indicates that only the first occurrence is to be replaced. In this case, the software finds the first occurrence of the search term in the document being edited and replaces it with the replacement text. The postcondition state is identical, except only the first occurrence is replaced, and the replacement text is highlighted. The user may indicate to replace the next occurrence, and this may repeated until all are replaced. 2. In Step 3, the user indicates that the software is only to search and not replace, and does not specify replacement text. In this case, the software highlights the first occurrence of the search term and the use case ends. The user may indicate to search for the next occurrence, and this may repeated until all occurences are retrieved.

Use Case Example (cont.)

Alternative paths (cont.)	3. In Step 2, the user may choose the case-sensitive option. 4. The user may decide to abort the search-and-replace operation at any time during Steps 1, 2, or 3. In this case, the software returns to the precondition state.
Postconditions	All occurrences of the search term have been replaced with the replacement text.
Acceptance Tests	Make sure that all occurrences of the search keyword are replaced with the replacement text.

Customer Acceptance Tests

- Client must describe how the user stories will be tested
  - With concrete data examples,
  - Associated with (one or more) user stories and use cases

Use Case Example - Smile

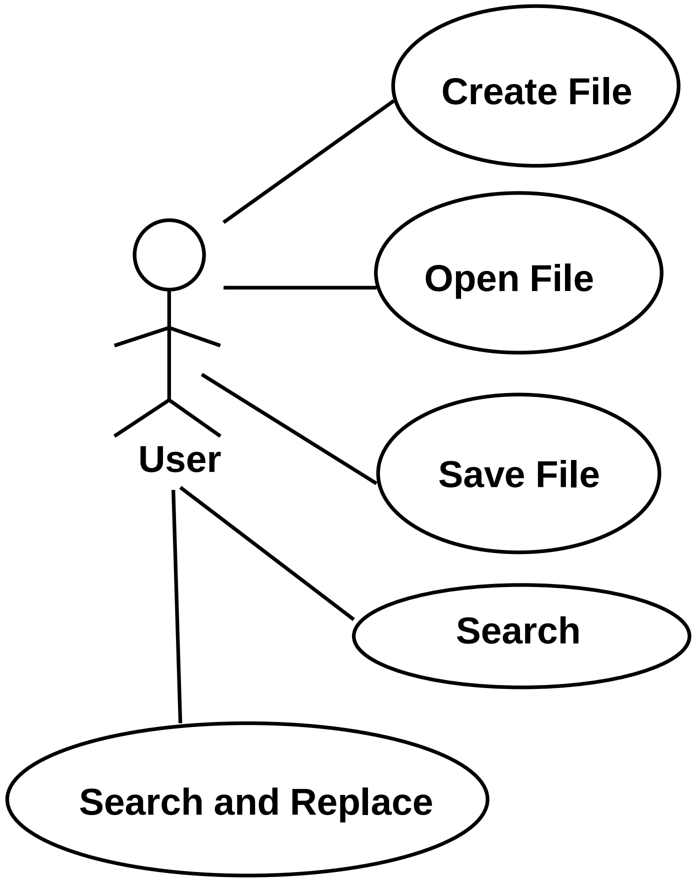
Name	
Users	
Rationale	
Triggers	
Preconditions	
Actions	
Alternative paths	
Postconditions	
Acceptance Tests	

Use Case Example - Smile

Actions	
Alternative paths	
Postconditions	
Acceptance Tests	

Use Case Diagrams

- Actors and use cases communicate when information is exchanged between them

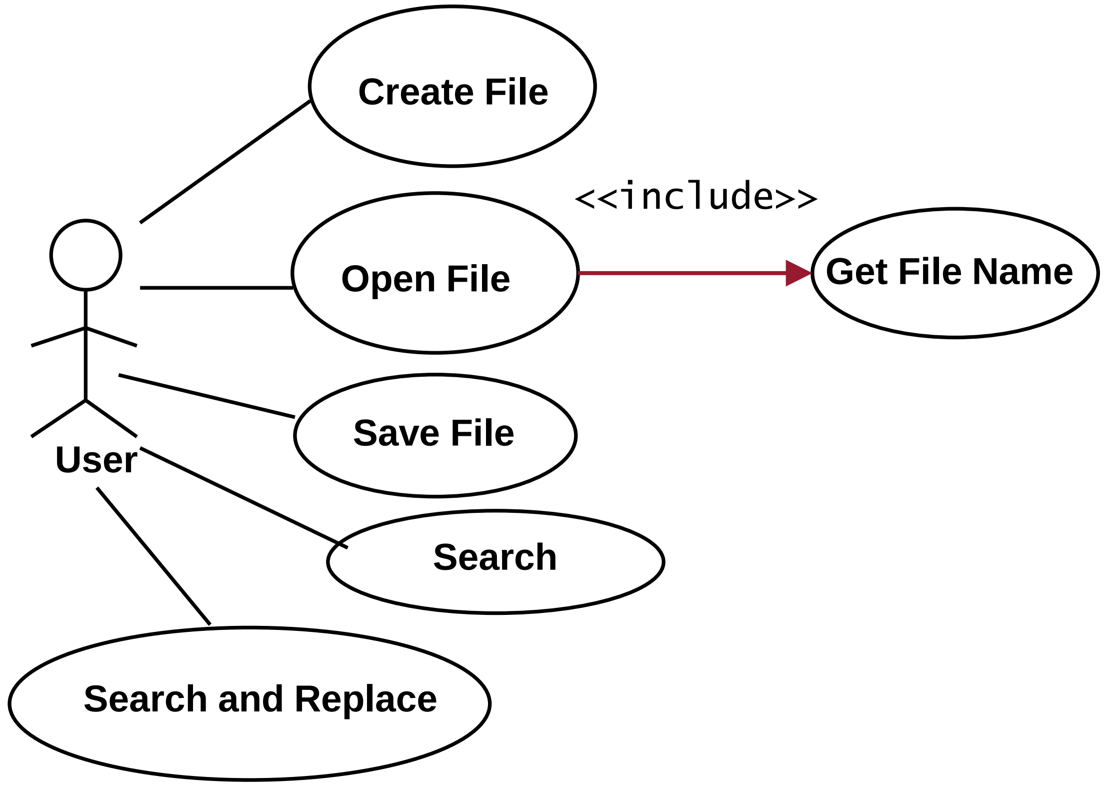


Use Case Associations

- Dependencies between use cases are represented with use case associations
- Associations are used to reduce complexity
- Types of use case associations
  - Include
    - Decompose a long use case into shorter ones
    - Reuse of existing functionality, avoid redundancy
  - Extend
    - Separate alternate flows of events
    - Handle exceptional cases
  - Generalization
    - Refine abstract use cases

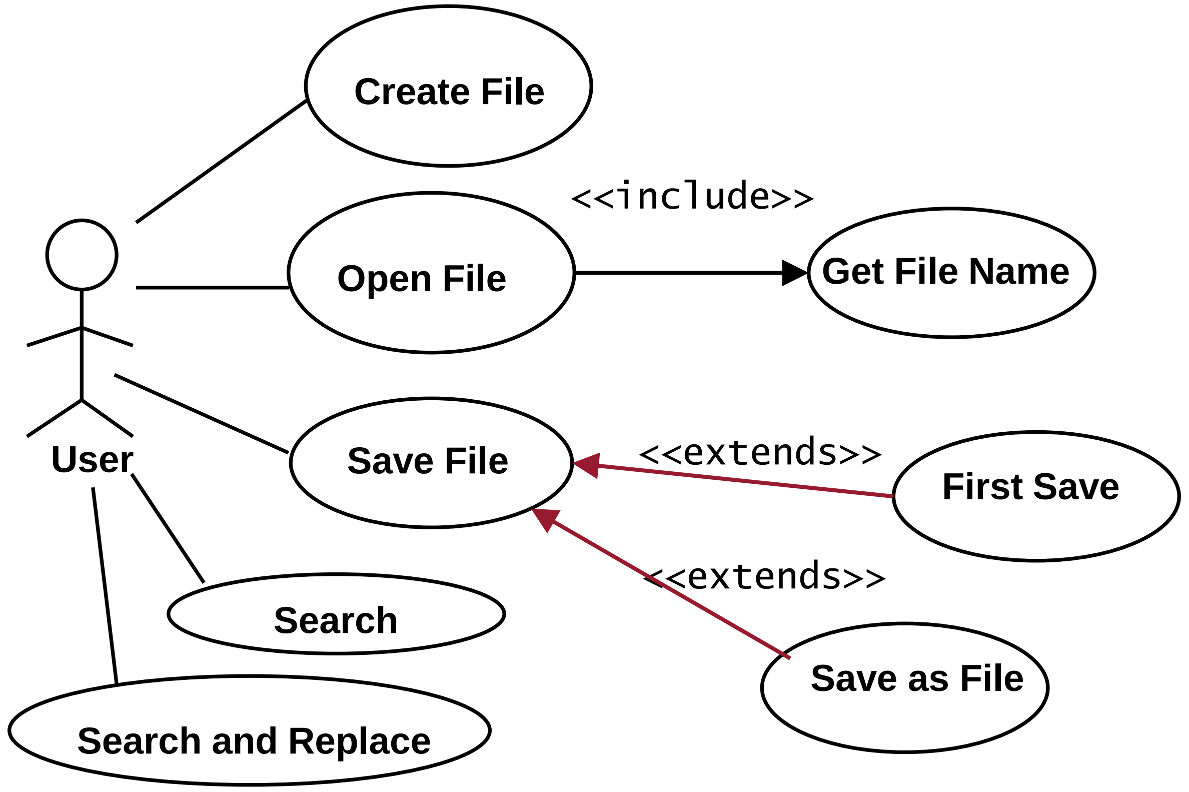
<<include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into shorter use cases



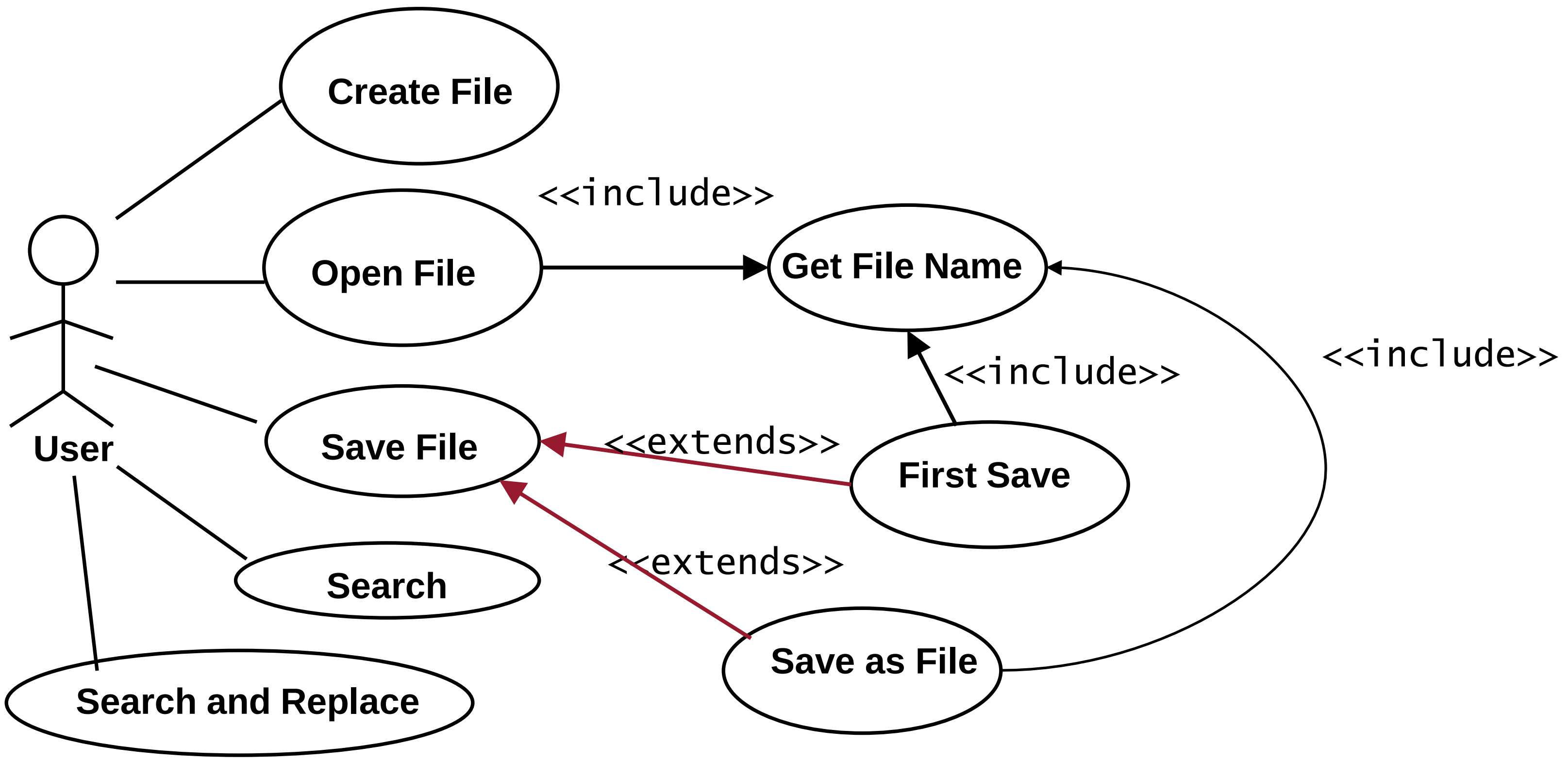
<<extend>> Handle Exceptional Cases

- Problem:
  - An exception B could occur during the detailed use case A.
- Solution:
  - Depict the exception handling use case B as an extension of the use case A

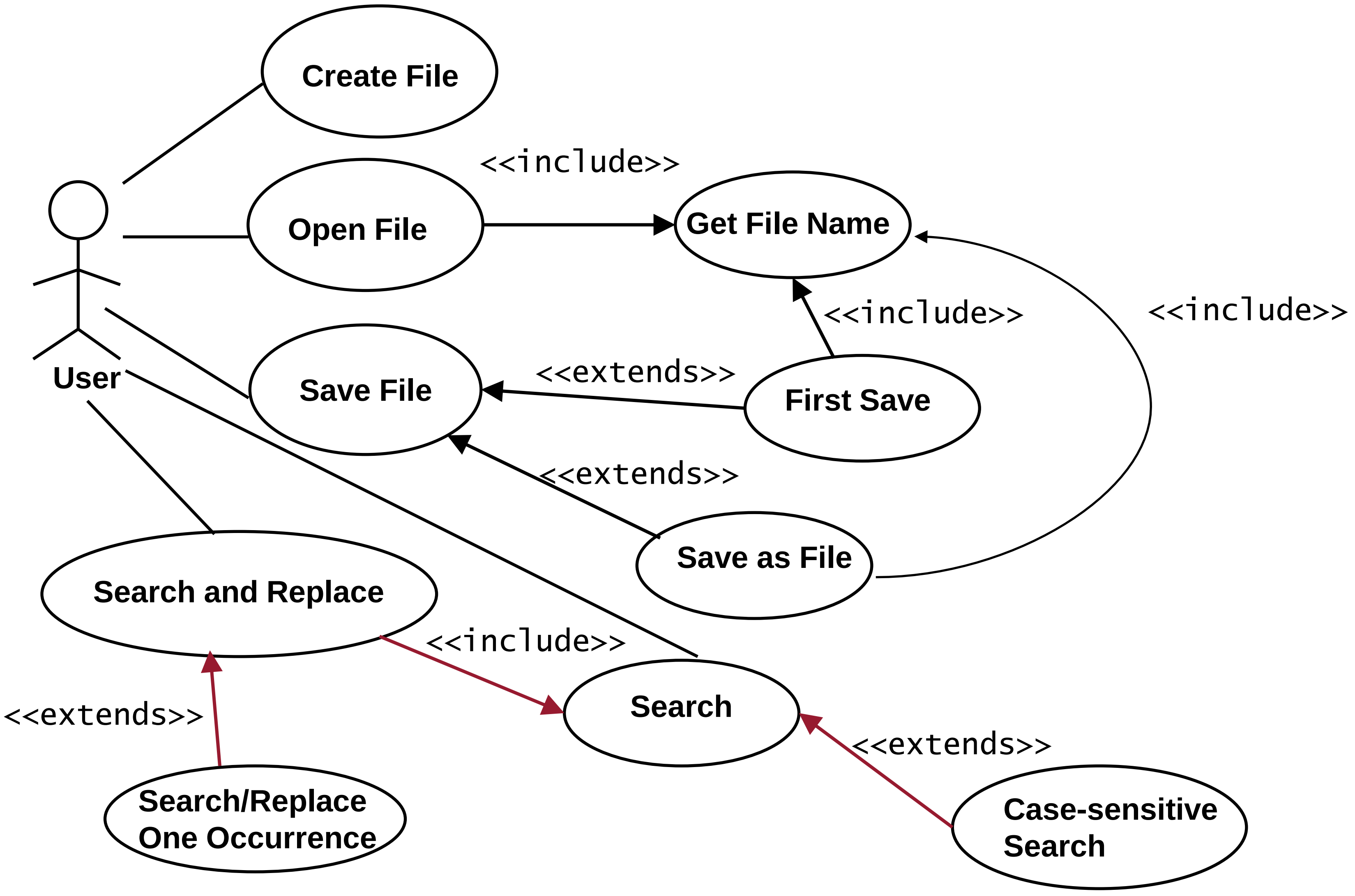


<<extend>> Handle Exceptional Cases

- Problem:
  - An exception B could occur during the detailed use case A.
- Solution:
  - Depict the exception handling use case B as an extension of the use case A



<<extend>> Handle Exceptional Cases



The Unified Modeling Language

UML is a standard language for modeling software systems

- Serves as a bridge between the requirements specification and the implementation.
- Provides a means to specify and document the design of a software system.
- Is process and programming language independent.
- Is particularly suited to object-oriented program development.

Models: Diagrams and Specification in UML

- In UML, a model consists of a diagram and a specification.
  - A diagram is the graphical representation of a set of elements, usually rendered as a connected graph of vertices (things) and arcs (relationships).
  - Each diagram is supported by technical documentation that specifies in more detail the model represented by the diagram.
  - A diagram without a specification is of little value.

Requirements Document

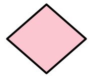
A. Problem
B. Background information
C. Requirements
    a. Functional Requirements
    b. Non-Functional requirements

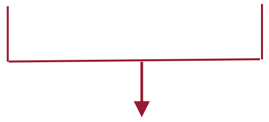
Activity Diagram

- Supplements the use-case by providing a graphical representation of the flow of interactions within a specific scenario.
  - Shows the flow of activities (sequential, branched, or concurrent)
  - The activity can be described as an operation in the system.

Specific activity/ operation

Flow through the system


Branching decision


Parallel activities

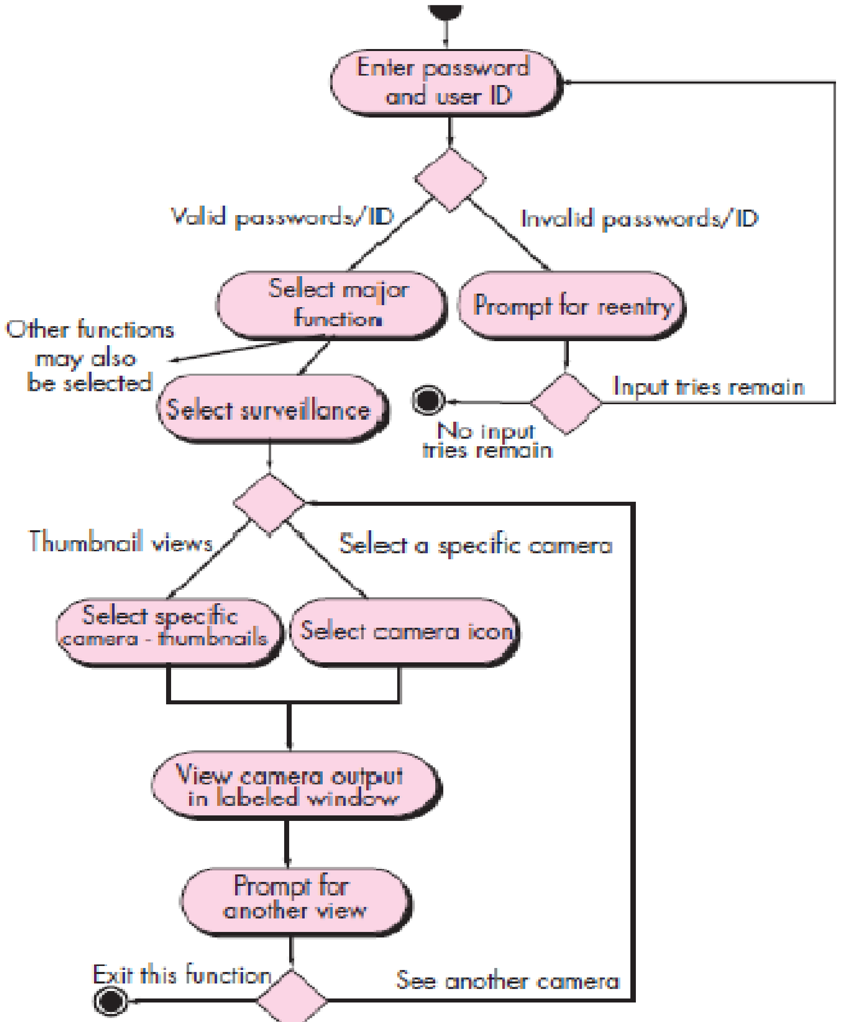
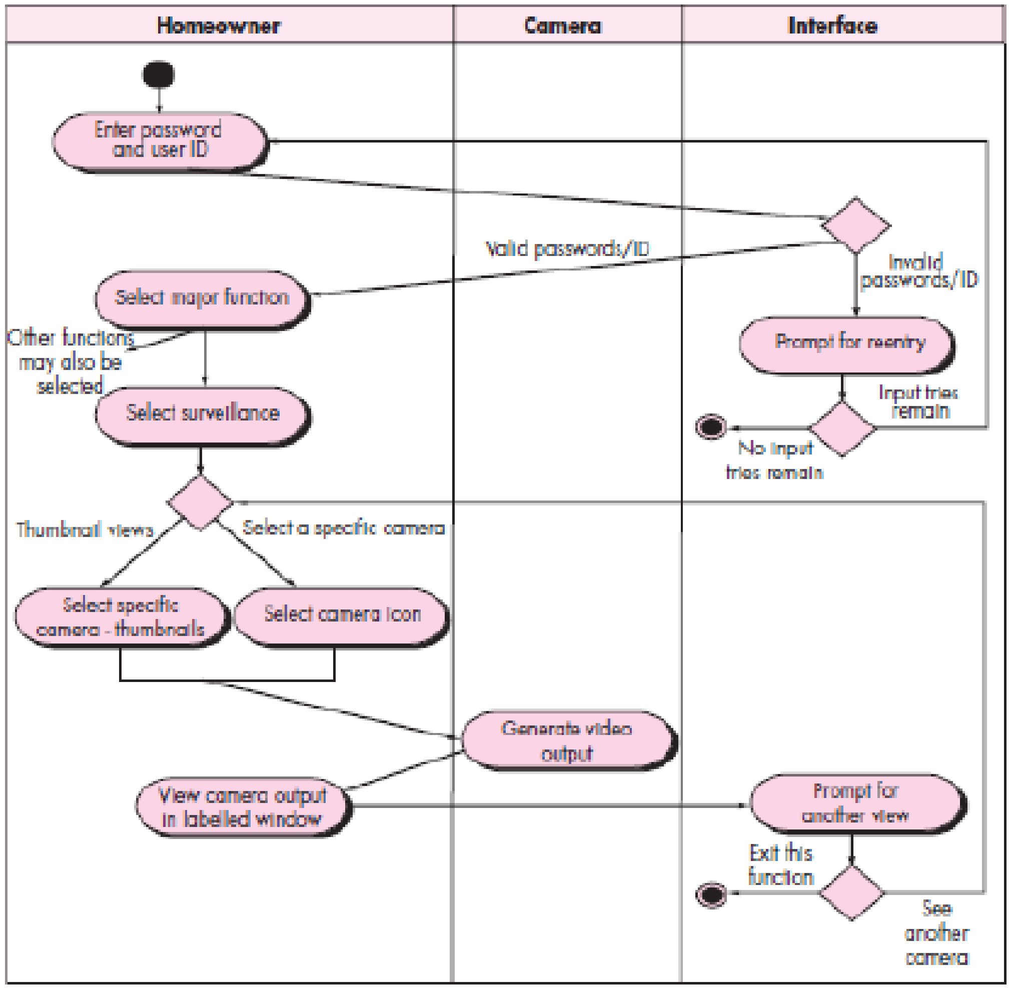


Fig 6.5: Activity diagram for Access Camera surveillance via Internet display cams

Swimlane Diagram

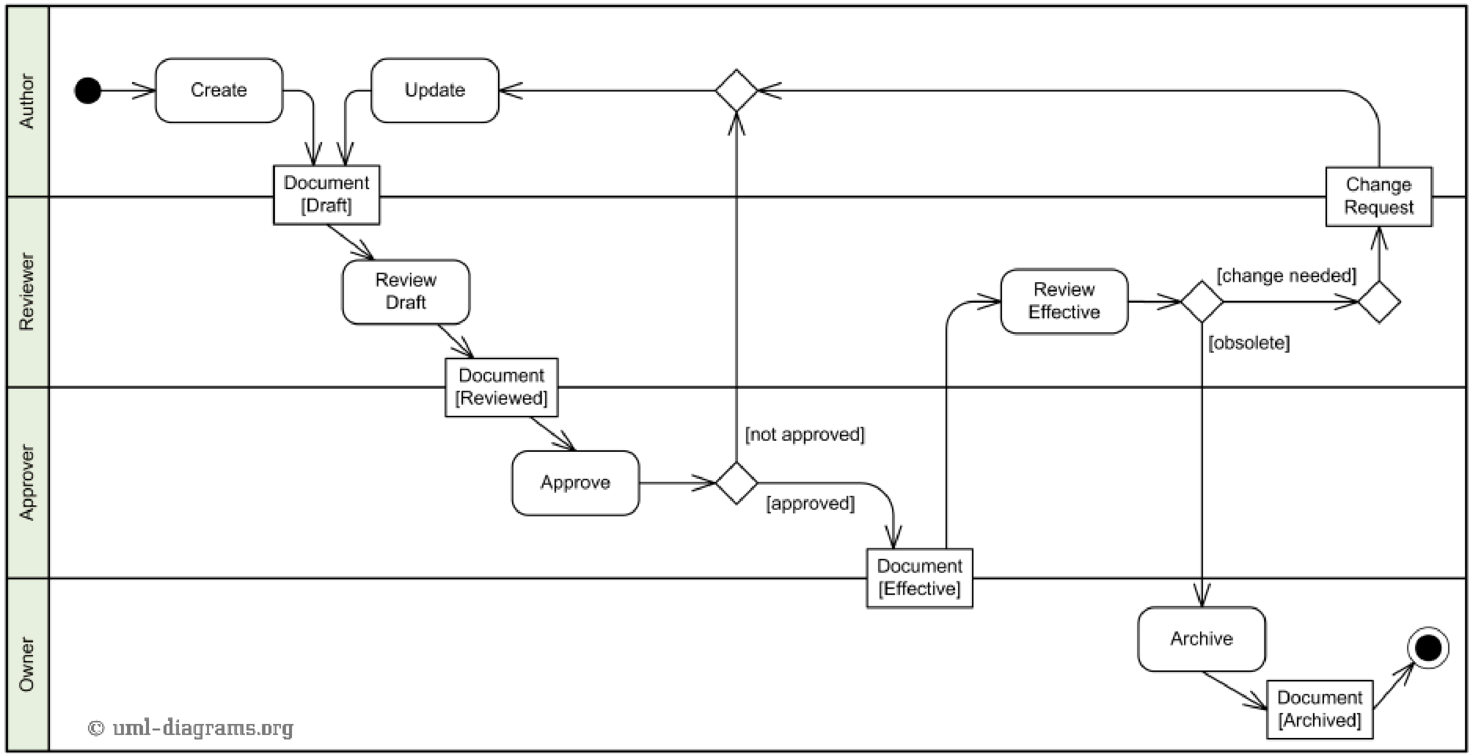
Fig 6.6: Swimlane diagram for Access camera surveillance via the Internet—display camera views function

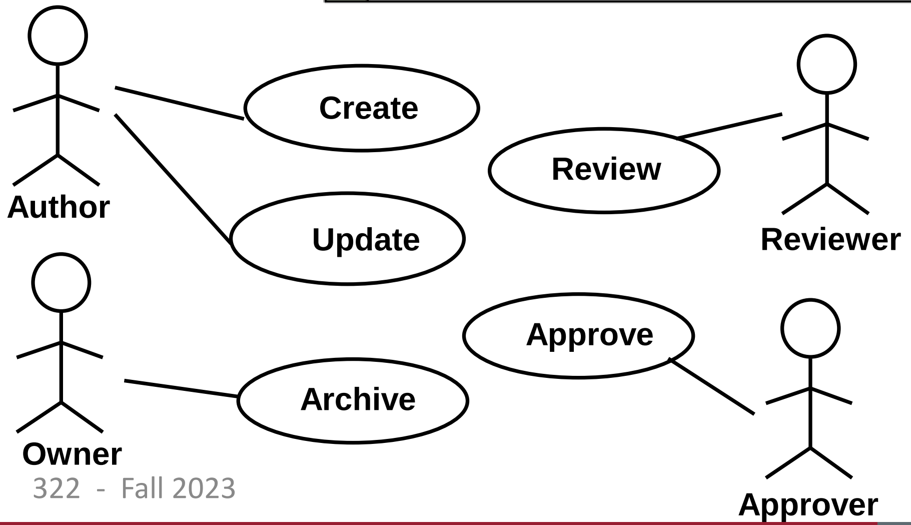


- Variation of the activity diagram.
- Allows to indicate which actor has the responsibility for the action described by an activity.

Swimlane diagram for a camera surveillance systems via Internet – display camera views function.

Swimlane Diagram



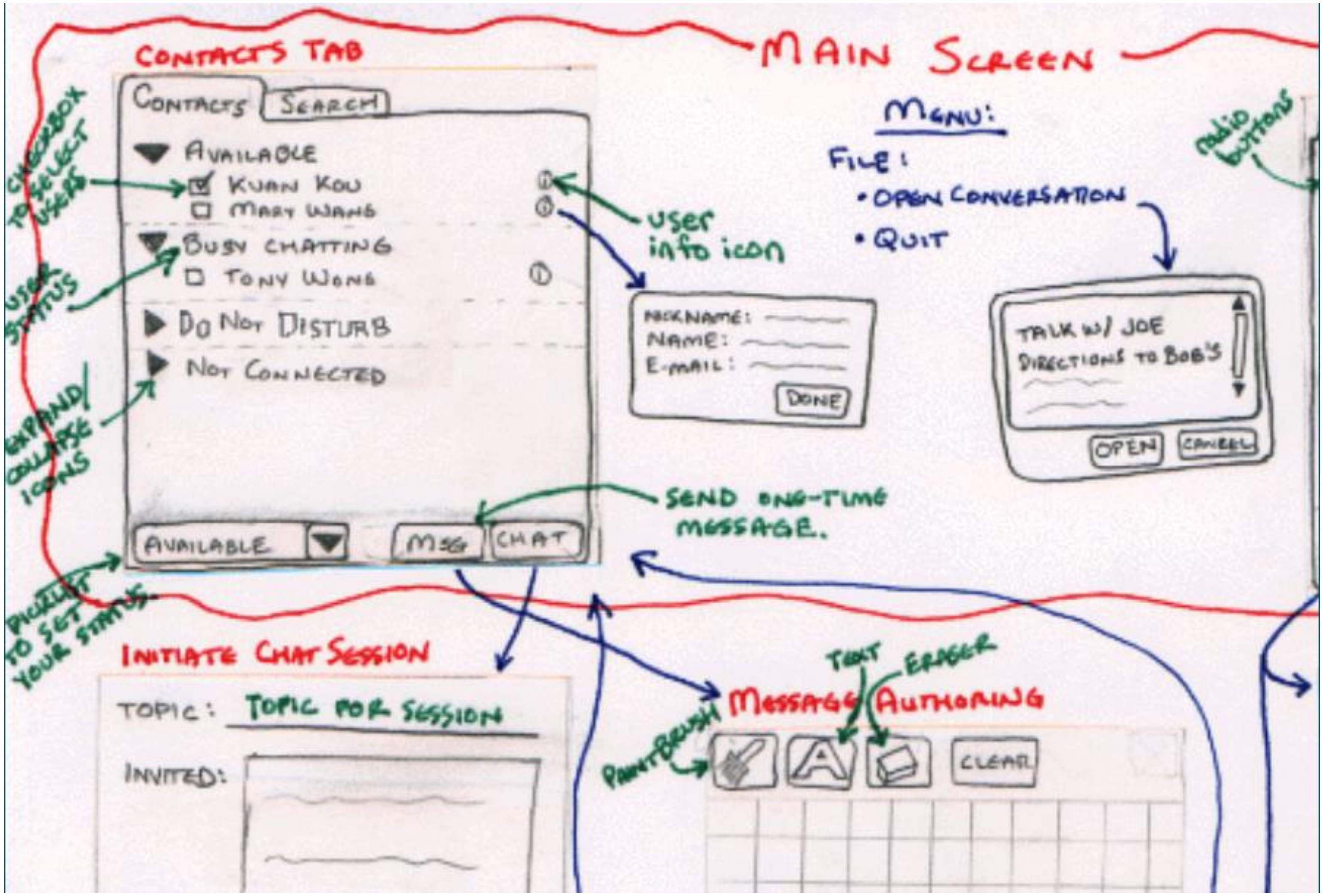


Managing Changing Requirements

- Requirements change because:
  - Business process changes
  - Technology changes
  - The problem becomes better understood

- Requirements analysis never stops
  - Continue to interact with the clients and users
  - The benefits of changes must outweigh the costs.
  - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
  - Larger-scale changes have to be carefully assessed

Paper UI Prototype



CONTACTS TAB
MAIN SCREEN
Contacts
Search
Available
Kuan Kou
Mary Wang
Busy Chatting
Tony Wong
Do Not Disturb
Not Connected
Available
MSG
CHAT
Checkbox to select users
user status
expand/collapse icons
picklist to set your status
user info icon
Nickname:
Name:
E-mail:
Done
Menu:
File:
• Open Conversation
• Quit
Talk w/ Joe
Directions to Bob's
Open
Cancel
radio buttons
Send one-time message.
Initiate Chat Session
Topic:
Topic for session
Invited:
Message Authoring
Text
Eraser
Paintbrush
Clear