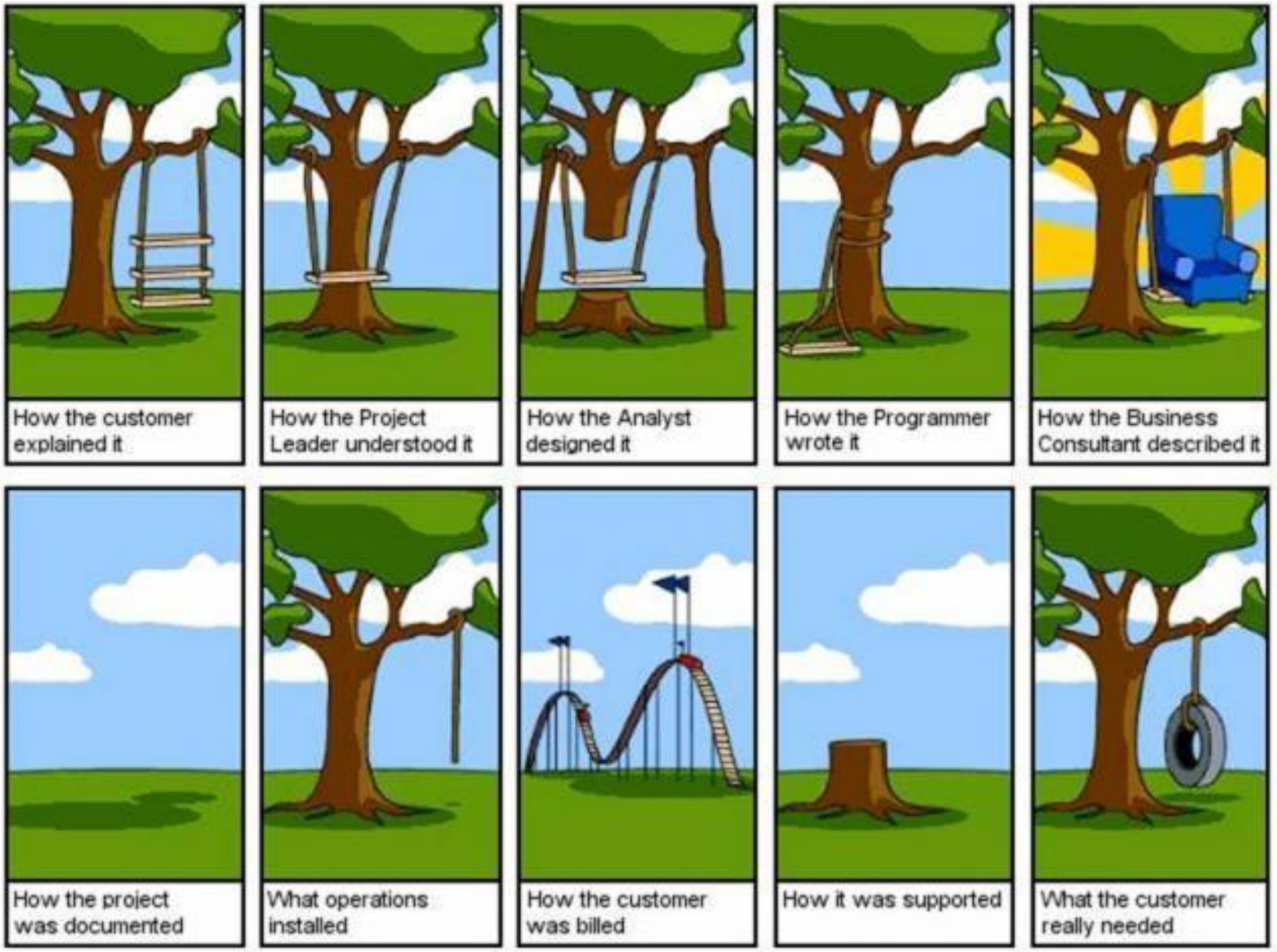
Requirements



Logistics

- One **deliverable** due every **Tuesday 11:59pm**
- **Progress report and agenda** due every **Wednesday 8pm**
- **Team meeting** every **Tuesday** 1:30pm -- 2:20pm
- **Project meeting** every **Thursday** 1:30pm -- 2:20pm

Suggested workflow:

- **Wednesday:** everyone has read the assignment
  - General assignment questions on Slack
  - Progress report and agenda: task assignment and project-specific questions
- **Thursday:** resolve project-specific questions in project meeting
- …
- **Tuesday:** final checks (all tasks done before the meeting)

Lecture outline

- What are requirements?
- How can we gather requirements?
- How can we document them? (Use cases)

Recap: Life-cycle stages

Virtually all SDLC models have the following stages:

- **Requirements** **⇐ Our focus today**
- Design
- Implementation
- Testing
- Release
- Maintenance

Traditional models:

- Waterfall, Prototyping, Spiral, etc.

Agile models:

- eXtreme Programming, Scrum, etc.

Software requirements

Requirements specify what to build

- tell “what” and not “how”
- tell the problem, not the solution
- reflect system design, not software design

"What vs. how" is relative

- One person's *what* is another person's *how*.
  - "One person's constant is another person's variable."
    Alan Perlis, "Epigrams on Programming" #1
    [first winner of the Turing Award, wrote the first compiler]

- Input file processing is the what, parsing is the how
- Parsing is the what, a stack is the how
- A stack is the what, an array or a linked list is the how
- A linked list is the what, a doubly linked list is the how
- A doubly linked list is the what, `Node*` is the how

Why requirements?

- Goals of requirements:
  - understand precisely what is required of the software
  - communicate this understanding precisely to all development parties
  - monitor and control production to ensure that system meets specification

- Requirements are useful to many people
  - customers: show what should be delivered (contractual base)
  - managers: scheduling and monitoring (progress indicator)
  - designers: provide a spec to design the system
  - developers: a range of acceptable implementations / output
  - QA / testers: a basis for testing, validation, verification

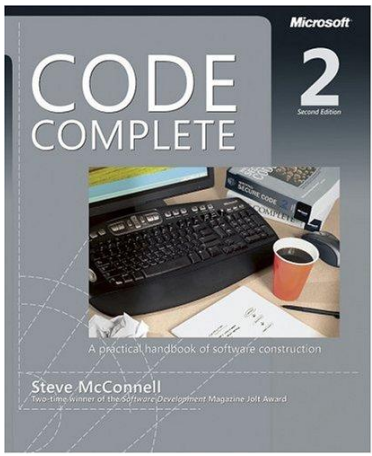
Value of requirements

The #1 reason that projects succeed is user involvement

- Standish Group survey of over 8000 projects

Easy access to end users is one of three critical success factors in rapid-development [agile] projects.

- Steve McConnell



Companies recognize this

The customer is always right

(Marshall Field's department store, 1852)

Customer obsession rather than competitor focus

(One of Amazon's four principles)

(1) Understand and serve the customer better than anyone else,
(2) forget about everything else, and
(3) make sure every little thing you do serves (1), always and everywhere

(Summary of Apple's original three principles, Steve Jobs)

Classifying requirements

- The classic way to classify requirements:
  - **functional**: map inputs to outputs
    - "The user can search either all databases or a subset."
    - "Every order gets an ID the user can save to account storage."
  - **nonfunctional**: other user-visible properties
    - ilities: dependability, reusability, portability, scalability, performance, safety, security
    - "Our deliverable documents shall conform to the XYZ process."
    - "The system shall not disclose any personal user information."
  - **additional constraints**
    - e.g., programming language, frameworks, testing infrastructure
- Another way to classify them (S. Faulk)
  - **Behavioral (user-visible)**: about the artifact (often measurable)
    - features, performance, security
  - **Development quality attributes**: about the process (can be subjective)
    - flexibility, maintainability, reusability

General classes of requirements

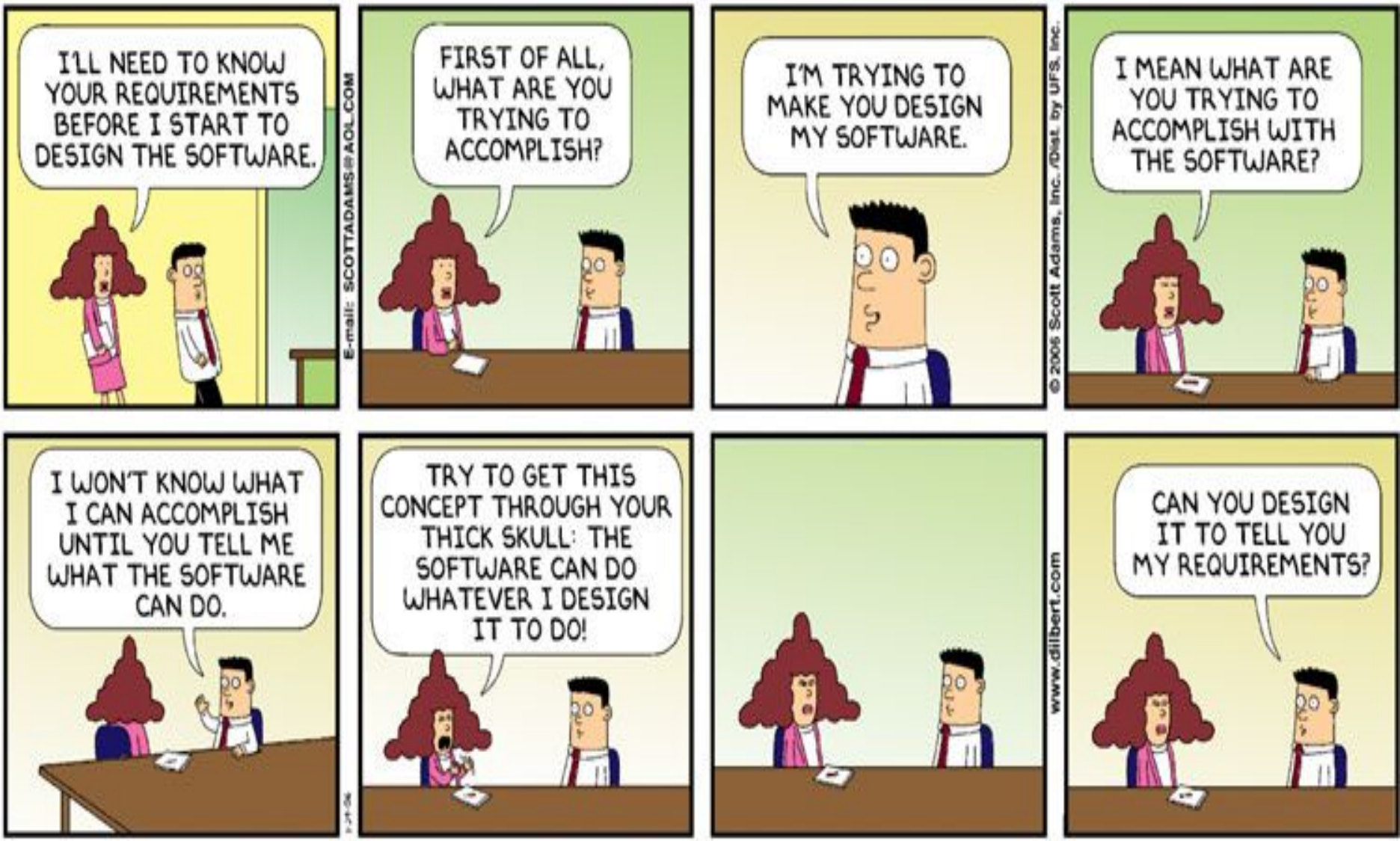
Example requirements types:

- Feature set
- GUI
- Performance
- Reliability
- Extensibility (support plug-ins)
- Environment (HW, OS, browsers)
- Schedule

Gather requirements from customers

Benefits of working with customers:

- Good relations improve development speed
- Improves perceived development speed
- They don’t always know what they want
- They do know what they want, and it changes over time



How to engage with customers

- Interviews & hallway conversations
- Observations, shadowing
- Use cases
- Feature list
- Mockups
- Prototyping

Keep your customer (user) at the center of the discussion
Listen, observe, and **ask clarifying questions**

How to elicit requirements

- Do:
  - Talk to the users, or work with them, to learn how they work.
  - Ask questions throughout the process — "dig" for requirements.
  - Think about *why* users do something in your app, not just what.
  - Allow (and expect) requirements to change later.
- Don't:
  - Be too specific or detailed.
  - Describe complex business logic or rules of the system.
  - 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.

Feature creep/bloat

Feature creep:

- Gradual accumulation of features over time.
- Beyond what was originally committed and/or actually needed.

Why does feature creep happen? Because features are fun!

- Developers like to code them.
- Sales teams like to brag about them.
- Users (think they) want them.

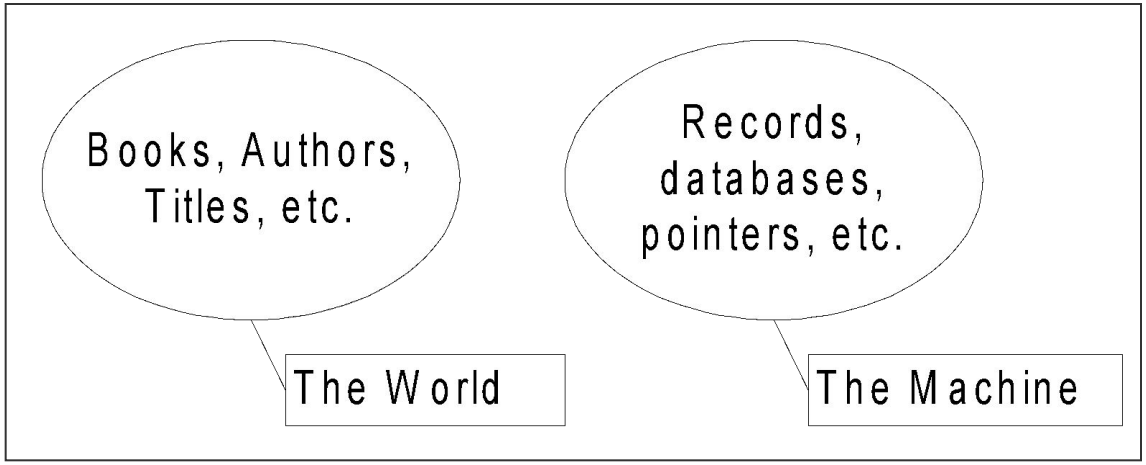
Why is it bad?

- Puts product delivery at risk
- Too many options, more bugs, more delays, less testing, …
- “Boiled frog” analogy.

Can you think of any products that have had feature creep?

The machine and the world

- The requirements are in the application domain
- The program defines the machine that has an effect in the application domain
- Example: a database system dealing with books



- Some things in the world are not represented by a given machine
  - Book sequels or trilogies
  - Pseudonyms
  - Anonymous books
- Some things in the machine do not represent anything in the world
  - Null pointers
  - Deleting a record
  - Back pointers

Good or bad requirements? (and why?)

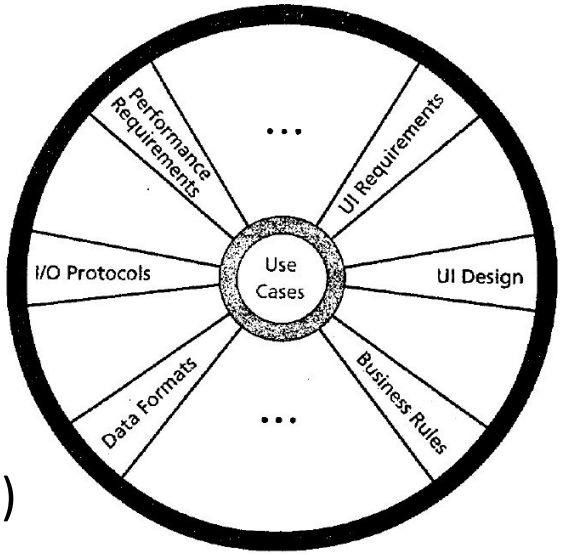
- The system will enforce 6.5% sales tax on Washington purchases.
- The system shall display the elapsed time for the car to make one circuit around the track within 5 seconds, in hh:mm:ss format.
- The product will never crash. It will also be secure against hacks.
- The server backend will be written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

How do we specify requirements?

- Use cases
- Feature list
- Paper UI prototype
- Prototype

Cockburn's requirements template

1. Purpose and scope
2. Terms (glossary)
3. **Use cases (the central artifact of requirements)**
4. Technology used
5. Other
   a. Development process: participants, values (fast-good-cheap), visibility, competition, dependencies
   b. Business rules (constraints)
   c. Performance demands
   d. Security, documentation
   e. Usability
   f. Portability
   g. Unresolved (deferred)
6. Human factors (legal, political, organizational, training)



A template leads to uniformity (good for you and the customer)

Challenges and common mistakes

Challenges

- Unclear scope and unclear requirements.
- Changing/evolving requirements.
- Finding the right balance (depends on customer):
  - Comprehensible vs. detailed.
  - Graphics vs. tables and explicit and precise wording.
  - Short and timely vs. complete and late.

Common Mistakes

- Implementation details instead of requirements.
- Projection of own models/ideas.
- Feature creep/bloat.

How to specify requirements

- Use cases
- Personas, user scenarios
- Storyboarding
- Paper prototyping
- Prototyping
- UML
- …

Use cases

What is a use case?

A **use case** is a written description of a **user's interaction** with the **software system** to accomplish a **goal**

Let's start with some terminology

- **Actor**: user interacting with the system (may be another system)
- **System:** the software product
- **Goal**: desired outcome of the primary actor
- **Flow:** interactive steps to achieve the goals

Use cases

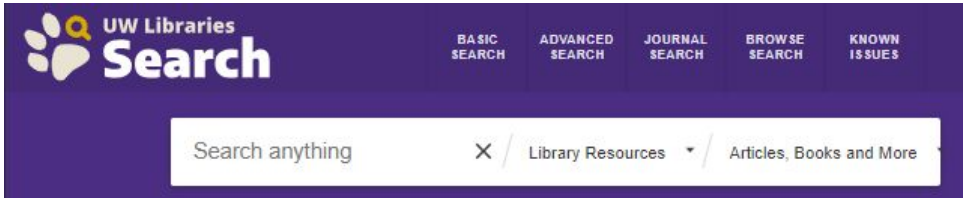
- A use case is an example behavior of the system

- Example:
  - Jane has a meeting at 10AM
  - Jim tries to schedule another meeting for her at 10AM
  - He is notified about the conflict

- A use case is a written description of a user's interaction with the software system to accomplish a goal

- A use case characterizes one way of using a system
- It represents a dialogue, or flow of events, between a user and the system, from the user's point of view
- It captures *functional* (input-output) requirements
- Similar to Extreme Programming "stories" and CRC (class responsibility collaborator) cards

An example



Goal	Reserve a book in the library app
Actor	Library patron
Main (success) flow	1. Patron selects the search screen 2. System presents a search box (with filters) 3. Patron types in the book title 4. System presents the books that match and branch locations 5. Patron selects location and reserves 6. System confirms and re-presents home page

Qualities of a good use case

- starts with a request from an actor to the system
- ends with the production of all the answers to the request
- defines the interactions (between system and actors) related to the function
- takes into account the actor's point of view, not the system's
- focuses on interaction, not internal system activities
- doesn't describe the GUI in detail
- has 3-9 steps in the main success scenario
- is easy to read
- summary fits on a page

Benefits of use cases

- Establish an understanding between the customer and the system developers of the requirements (success scenarios)
- Alert developers of problematic situations (extension scenarios)
- Capture a level of functionality to plan around (list of goals)

Actors: the agents in a use case

Actor: something that interacts with your system and appears in a use case
Examples:

- a human
- external hardware (like a timer)
- another system

Primary actor: actor who initiates the action

Goal: desired outcome of the primary actor

Do use cases capture these?

Which of these requirements should be represented directly in a use case?

1. Order cost = order item costs * 1.065 tax
2. Promotions may not run longer than 6 months
3. Customers only become Preferred after 1 year
4. A customer has one and only one sales contact
5. Response time is less than 2 seconds
6. Uptime requirement is 99.8%
7. Number of simultaneous users will be 200 max

Styles of use cases

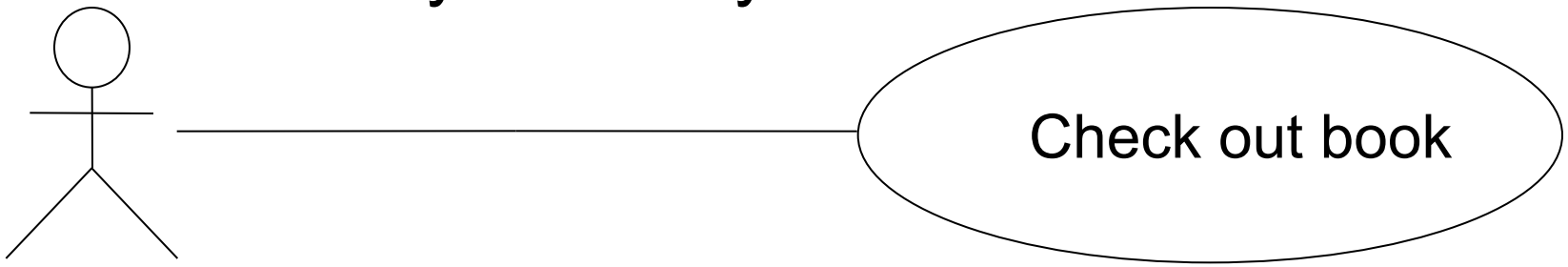
1. Use case diagram
   - often in UML, the Unified Modeling Language
2. Informal use case
3. Formal use case
   (≠ formal specification)

Let's examine each of these in detail...

1. Use case summary diagrams

The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses, with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on

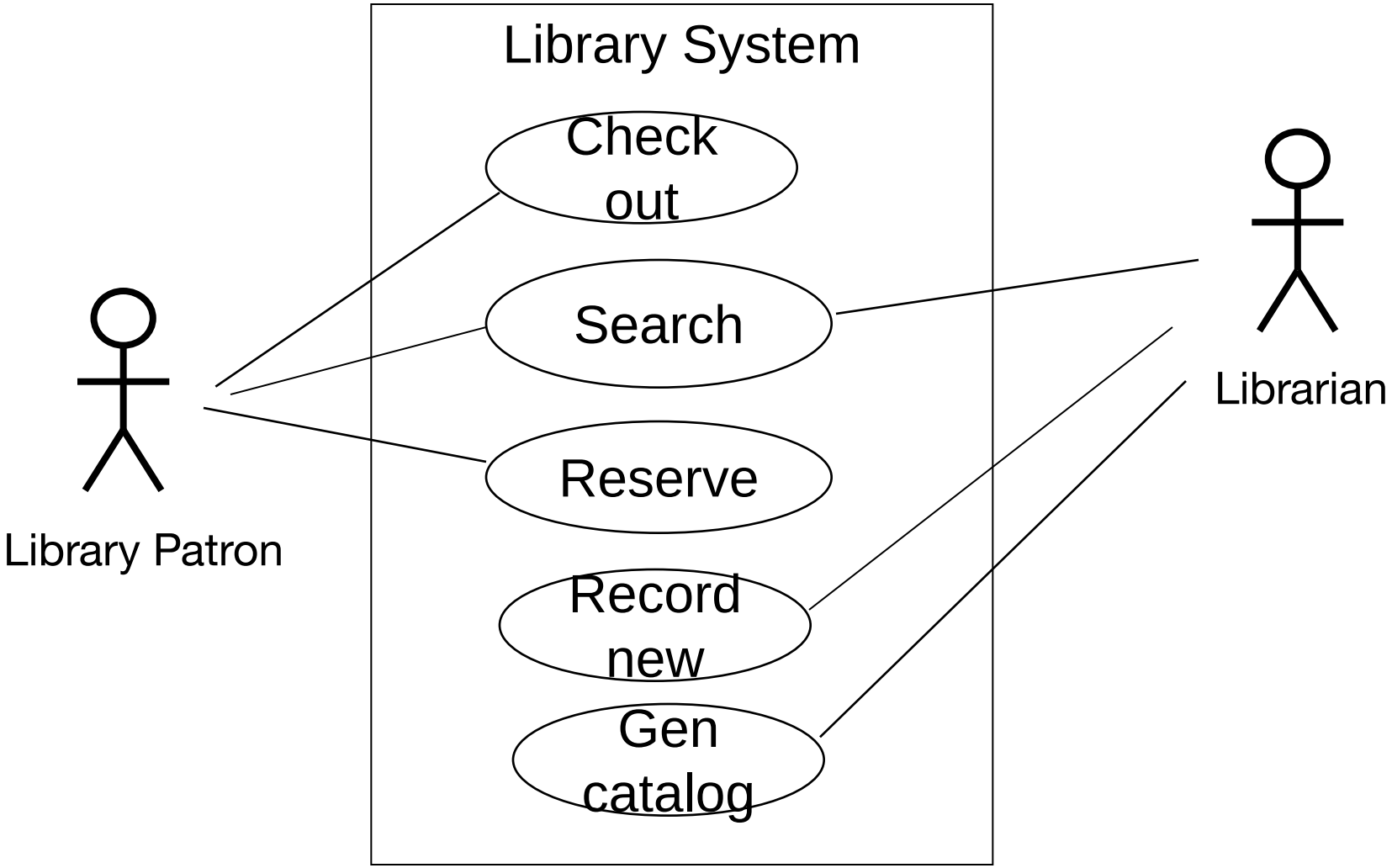


Use case summary diagrams

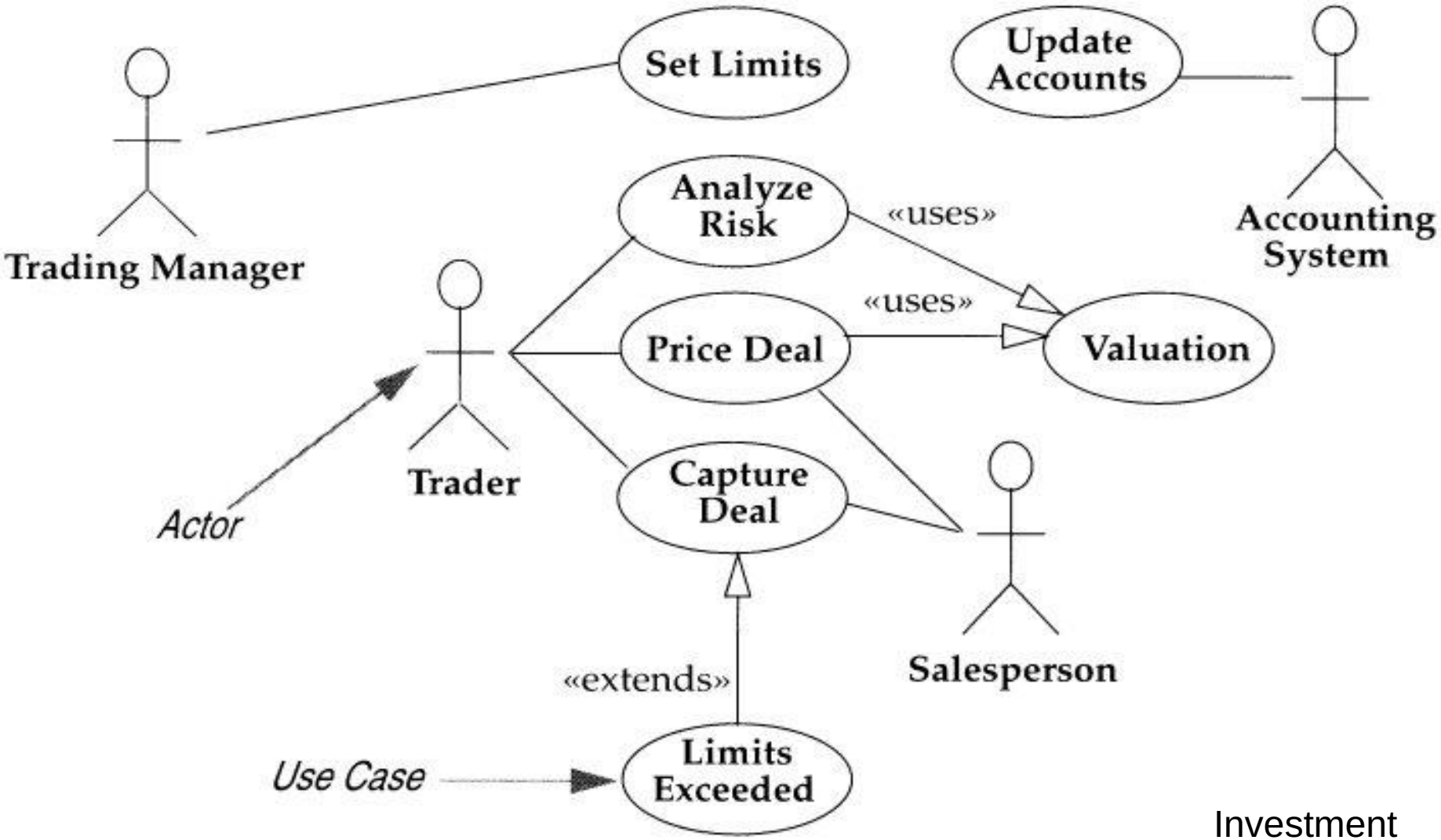
It can be useful to create a list or table of primary actors and their "goals" (use cases they start). The diagram will then capture this material.

Actor	Goal
Library Patron	Search for a book
	Check out a book
	Return a book
Librarian	Search for a book
	Check availability
	Request a book from another library

Use case summary diagram 1



Use case summary diagram 2



Investment System

2. Informal use case

Informal use case is written as a paragraph describing the scenario/interaction

- Example:
  - Patron Loses a Book
    The library patron reports to the librarian that she has lost a book.
    The librarian prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee.
    The system will be updated to reflect lost book, and patron's record is updated as well.
    The head librarian may authorize purchase of a replacement book.

Structured natural language

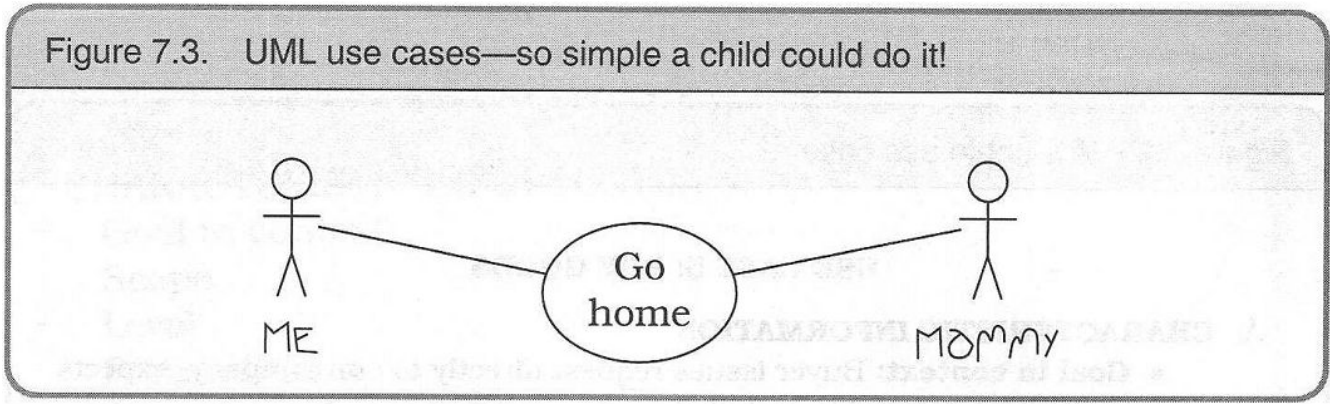
- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - » I.A.ii.3.q
- Although not ideal, it is almost always better than unstructured natural language
  - Unless the structure is used as an excuse to avoid content
- You will probably use something in this general style

3. Formal use case

Goal	**Patron wishes to reserve a book using the online catalog**
Primary actor	**Patron**
Scope	**Library system**
Level	**User**
Precondition	**Patron is at the login screen**
Success end condition	**Book is reserved**
Failure end condition	**Book is not reserved**
Trigger	**Patron logs into system**

Main Success Scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices to patron 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject

What notation is good?



- There are standard templates for requirements documents, diagrams, etc. with specific rules. Is this a good thing? Should we use these standards or make up our own?
  - Good: standards are helpful as a template or starting point; Others are more likely to understand
  - But don't be a slave to formal rules or use a model/scheme that doesn't fit your project's needs.

4 steps for creating a use case

1. **Identify actors and goals**

- Actors: What users and (sub)systems interact with our system?
- Goals: What does each actor need our system to do?

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**

- Main success scenario is the preferred "happy path”
  - Easiest to read and understand
  - Everything else is a complication on this
- Capture each actor's intent and responsibility, from trigger to goal
  - State what information passes between actors
  - Number each step (line)

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**
3. **List the failure extensions**

- Many steps can fail (e.g., denied credit card, out of stock)
  - Note each failure condition separately, after the main success scenario
- Describe failure-handling
  - recoverable: back to main scenario (low stock + reduce quantity)
  - non-recoverable: fails (out of stock)
  - each scenario goes from trigger to completion
- Label with step number (success scenario line) and letter
  - 5a <failure condition>; 5a.1 <fail with error message>
  - 5b <failure condition>; 5b.1 <action>; 5b.2 <continue at failure step 7>

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**
3. **List the failure extensions**
4. **List the variations**

- Steps can have alternative behaviors
  - Label alternatives with step number (success scenario line) and symbol
    - 5’ <Alternative 1 for step 5>
    - 5’’ <Alternative 2 for step 5>

Use case description

- How and when it begins and ends
- The interactions between the use case and its actors, including when the interaction occurs and what is exchanged
- How and when the use case will need data from or store data to the system
- How and when concepts of the problem domain are handled

Jacobson example: recycling

The course of events starts when the customer presses the “Start-Button” on the customer panel. The panel’s built-in sensors are thereby activated.

The customer can now return deposit items via the customer panel. The sensors inform the system that an object has been inserted, they also measure the deposit item and return the result to the system.

The system uses the measurement result to determine the type of deposit item: can, bottle or crate.

The day total for the received deposit item type is incremented as is the number of returned deposit items of the current type that this customer has returned...

Another example: Buy a product

http://ontolog.cim3.net/wiki/UseCasesSimpleTextExample.html

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- Alternative: **Authorization Failure**
  - At step 6, system fails to authorize credit purchase
  - Allow customer to re-enter credit card information and re-try
- Alternative: **Regular Customer**
  - 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  - 3b. Customer may accept or override these defaults
  - Return to primary scenario at step 6

What is a use case?

A **use case** is a **written description** of a **user's interaction with** the software **system to accomplish** a **goal**.

- It is an **example behavior** of the system
- Written from an **actor's point of view**, not the system's
- **3-9 clearly written steps** lead to a "main success scenario"

Benefits of use cases

- Establish an understanding between the customer and the developers of the requirements (**success scenarios**)
- Alert developers of special cases (alternatives) and error cases (exceptions) to test (**extension scenarios**)
- Capture a level of functionality (**list of goals**)

What is an extension?

A possible **branch** in a use case, e.g., **triggered by an error**; useful for identifying what **edge cases** need to be **handled/tested**

Do

- Think about how every step of the use case could fail
- Give a plausible response to each extension from the system
- Response should either jump to another step of the case, or end it

Don’t

- List things outside the use case ("User's power goes out")
- Make unreasonable assumptions ("DB will never fail")
- List a remedy that your system can't actually implement
- Go overboard

Qualities of a good use case

- **Focuses on interaction**
  - Starts with a request from an actor to the system
  - Ends with the production of all the answers to the request
- **Focuses on essential behaviors, from actor's point of view**
  - Does not describe internal system activities
  - Does not describe the GUI in detail
- **Concise, clear, and accessible to non-programmers**
  - Easy to read
  - Summary fits on a page
  - Main success scenario and extensions

Use cases vs. other requirements

Which of the following requirements should be directly represented as a use case?

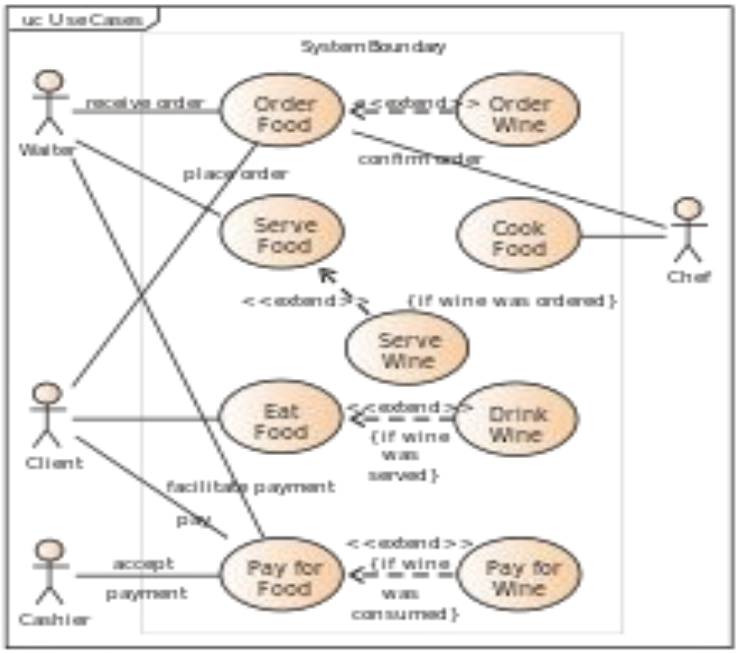
- Special deals may not run longer than 6 months.
- Customers only become preferred after 1 year.
- A customer has one and only one sales contact.
- Database response time is less than 2 seconds.
- Web site uptime requirement is 99.8%.
- Number of simultaneous users will be 200 max.

Styles of use cases

- Use case diagram (often in UML)
- Textual use case
  - Formal use case (≠ formal specification)
  - Informal use case

Use case diagram

“For reasons that remain a mystery to me, many people have focused on the stick figures and ellipses in use case writing since Jacobson's first book came out, and neglected to notice that use cases are fundamentally a text form.”
[Writing Effective Use Cases, Alistair Cockburn, 2000]



Formal use case

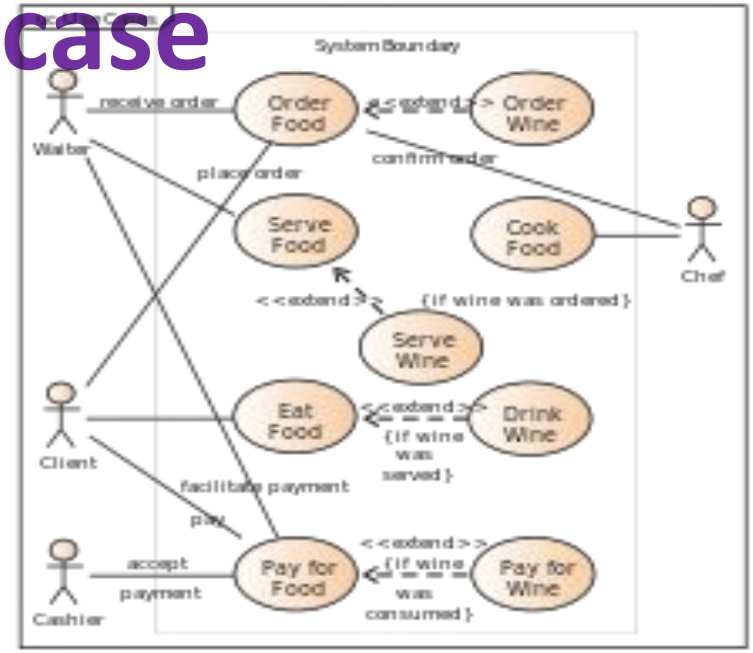
Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Formal use case: example

Goal	Patron wishes to reserve a book using the online catalog
Primary actor	Patron
Scope	Library system
Level	User
Precondition	Patron is at the login screen
Success end	Book is reserved
Failure end	Book is not reserved
Trigger	Patron logs into system

Main success scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject

Use case diagram vs. textual use case



Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Which one would you choose and why?

Informal use case: example

Patron loses a book

The **library patron** reports to the librarian that she has lost a book. The **librarian** prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The **system** will be updated to reflect lost book, and patron's record is updated as well. The **head librarian** may authorize purchase of a replacement book.

Informal use case with added structure

Use case 1: Patron loses a book

1.

   a.

      i.

Although not ideal, it is almost always better than unstructured text.

You will probably use something in this general style
or a template for formal use cases.

Use case wrap up (time permitting)

Which of the following requirements could be directly represented as a use case?

- Special deals may not run longer than 6 months
- Customers only become preferred after 1 year
- A customer has one and only one sales contact
- Database response time is less than 2 seconds
- Web site uptime requirement is 99.8%
- Number of simultaneous users will be 200 max

Requirements for a music player

Are these good requirements?

- Available on web and mobile
- Provide volume control
- Provide ability to flag favorites using a pulldown menu
- Enable variable playback speed
- Propose songs using ChatGPT recommendations
- Propose songs based on customer selected genres
- Written in JavaScript for extensibility and reliability

Pulling it all together

How much is enough?

You have to find a balance

- comprehensible vs. detailed
- graphics vs. explicit wording and tables
- short and timely vs. complete and late

Your balance may differ with each customer depending on your relationship and flexibility

Try it with a use case for your project

Goal

Actor

Main (success) flow

1.
2.
3.
4.
…

- Capture your thoughts –

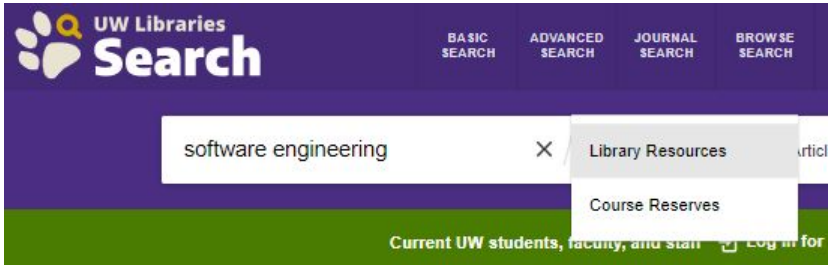
We'll rotate today through groups to discuss your use cases

Let's double click on these other flows

Variations and exceptions can be thought of as **branches** in a use case useful for identifying other situations that need to be handled

Variation (alternate) flows:

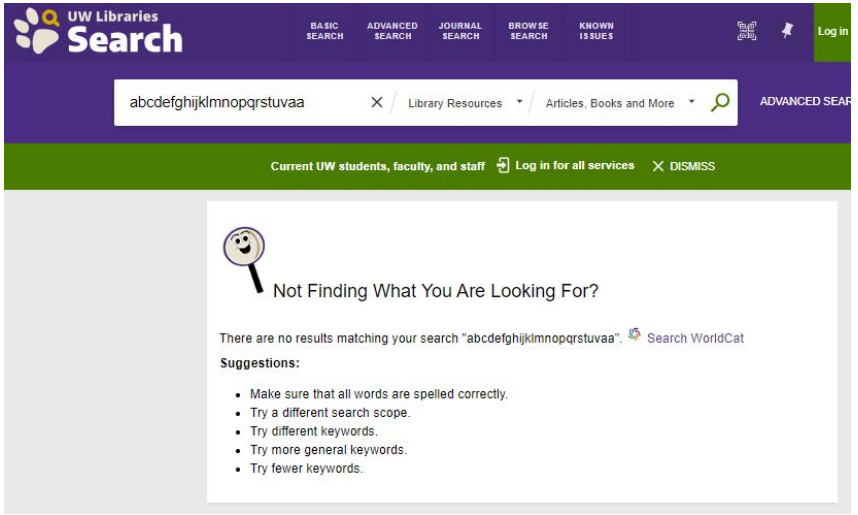
- These paths describe extensions on the main theme
- Another way to meet the goal
- Library search – Patron enters an author or subject or category



Let's double click on these other flows

Exception (error) flows:

- These paths describe failure conditions
- What happens when the goal is not achieved
- Library search – no book is found, system times out



We can capture this in our template

Goal	Reserve a book in the library app
Actor	Library patron
Main (success) flow	1. Patron selects the search screen 2. System presents a search box (with filters) **3. Patron types in the book title** 4. System presents the books that match and branch locations 5. Patron selects location and reserves 6. System confirms and represents home page
Variation (alternat	**(In step 3)** **3.1 Patron types in an author …**

Here's another example – ATM machine

Goal	Withdraw money
Actor	Bank patron
System	ATM

Precondition	Authenticated in
Trigger	Select withdraw

Main (success) flow	1. System displays account types 2. User chooses type 3. System asks for amount to withdraw 4. User enters amount 5. System debits user's account and dispenses money 6. User removes money 7. System prints and dispenses receipt
Exception flow	(In step 5) … 5.1.a System notifies that account funds are insufficient 5.1.b System displays current balance [and

Try it with a use case for your project

Goal

Actor

Main (success) Flow

Variation (alternate) Flow

Exception (error) Flow

- Capture your thoughts – We'll hear from another few teams

Summing up use cases

- Focus on interaction
  - Start with a request from an actor to the system
  - End with the production of all the answers to the request
- Focus on essential behaviors, from actor's point of view
  - Don't describe internal system activities
  - Don't describe the GUI in detail
- Be concise, clear, and accessible to non-programmers
  - Easy to read
  - Summary fits on a page

Some reference

Basic Use Case Template (Cockburn) https://canvas.uw.edu/courses/1680496/files/folder/UseCase%20Template?preview=110607742

and/or

Use Cases (Usability.gov) https://www.usability.gov/how-to-and-tools/methods/use-cases.html

Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Switching gears to another technique …

1. What are techniques used to specify requirements?
   - Use cases
   - Personas and user scenarios we are here
   - Storyboarding
   - Paper prototyping
   - Prototyping
   - UML
   - Feature list
   - …

What to Consider When Writing Scenarios

Good scenarios are concise but answer the following key questions:

- **Who is the user?** Use the personas that have been developed to reflect the real, major user groups coming to your site.
- **Why does the user come to the site?** Note what motivates the user to come to the site and their expectations upon arrival, if any.
- **What goals does he/she have**? Through task analysis, you can better understand the what the user wants on your site and therefore what the site must have for them to leave satisfied.

Personas

A **persona** is a description of a person who is representative of a population using your system

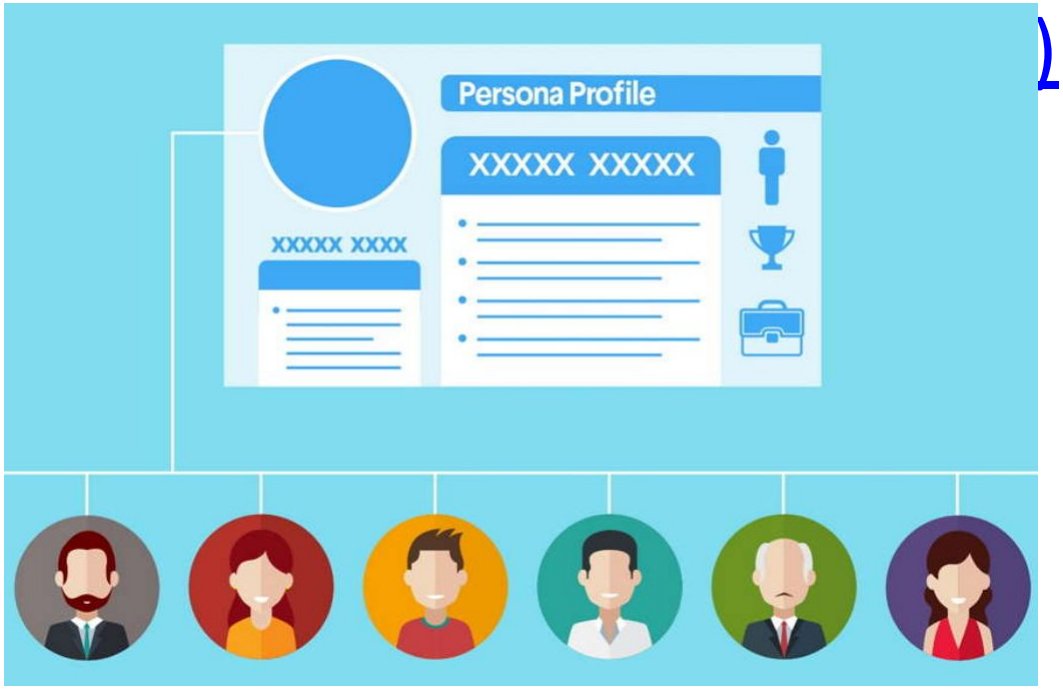
Each persona may have a different perspective of what they need

Example: Library catalog service (UW Libs)

- Persona: Admin
- Persona: Librarian
- Persona: Student
- Persona: Instructor

What might be an analogy to a persona in a use case?

Personas can be described with cards

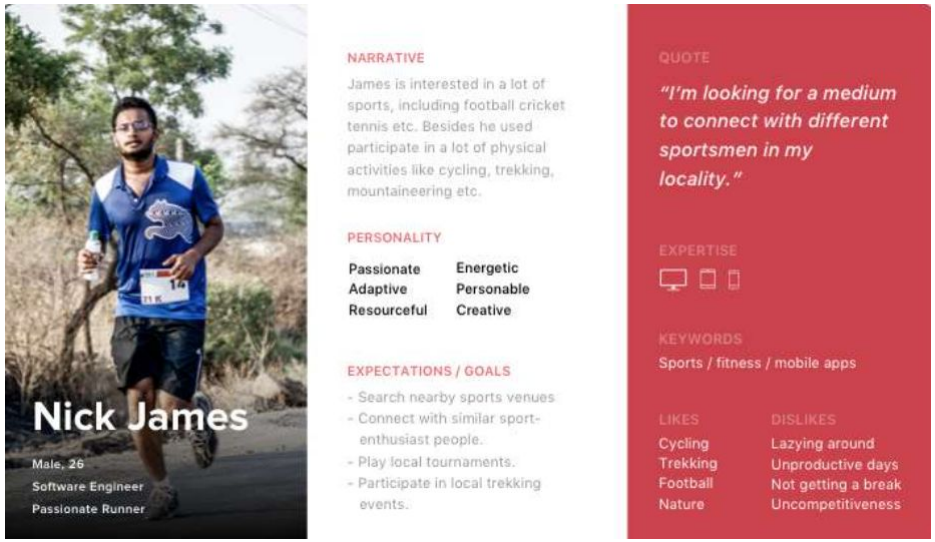


Cards typically include:

- Persona name and photo/image
- A quote that captures their goals and motivations
- Demographics (group they represent)
- Computer competence and usage
- Wants and needs
- Frustrations and pain points

Mockplus: User Persona Templates for Free Download

Lots of great examples on the web



Mockplus: User Persona Templates for Free Download

User scenarios

For each persona you can define the requirements from that person's perspective through a user scenario

Example: As an instructor, I am constantly looking for class resources that are relevant and up to date. Moreover, when I find a resource, I want to know it's available free-of-charge for the students and comes with online access.

Example: As a student, I want to be able to have the search provide smart results, so that I don't spend hours wading through irrelevant matches. I'd like to prioritize results that are timely, in-the-news, most-popular, and most-referenced across the

Writing user scenarios

From:
https://www.usability.gov/how-to-and-tools/methods/sce

What to Consider When Writing Scenarios

Good scenarios are concise but answer the following key questions:

- **Who is the user?** Use the personas that have been developed to reflect the real, major user groups coming to your site.
- **Why does the user come to the site?** Note what motivates the user to come to the site and their expectations upon arrival, if any.
- **What goals does he/she have**? Through task analysis, you can better understand the what the user wants on your site and therefore what the site must have for them to leave satisfied.

Doesn't this sound like use cases!

persona ~= actor

scenario ~= flow

Personas and scenarios are hugely valuable

- **They tap into a fundamental human skill—the ability to make predictions about how other people will react based on mental models of them**
- Enable us to capture inferences about the needs and desires of audience segments
- Draw attention to "pain points" and opportunity for new solutions
- Serve to communicate user characteristics and their individual types of requirements in a compact and easily understood way