Software Requirements

CSE 403 Software Engineering

Winter 2026

Today’s Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. What are techniques used to specify them?

Recapping where requirements fit in

Virtually all SDLC models have the following stages →

Requirements are at the top of the list as we start the journey of product development

<u>Common stages</u>

Requirements

Design

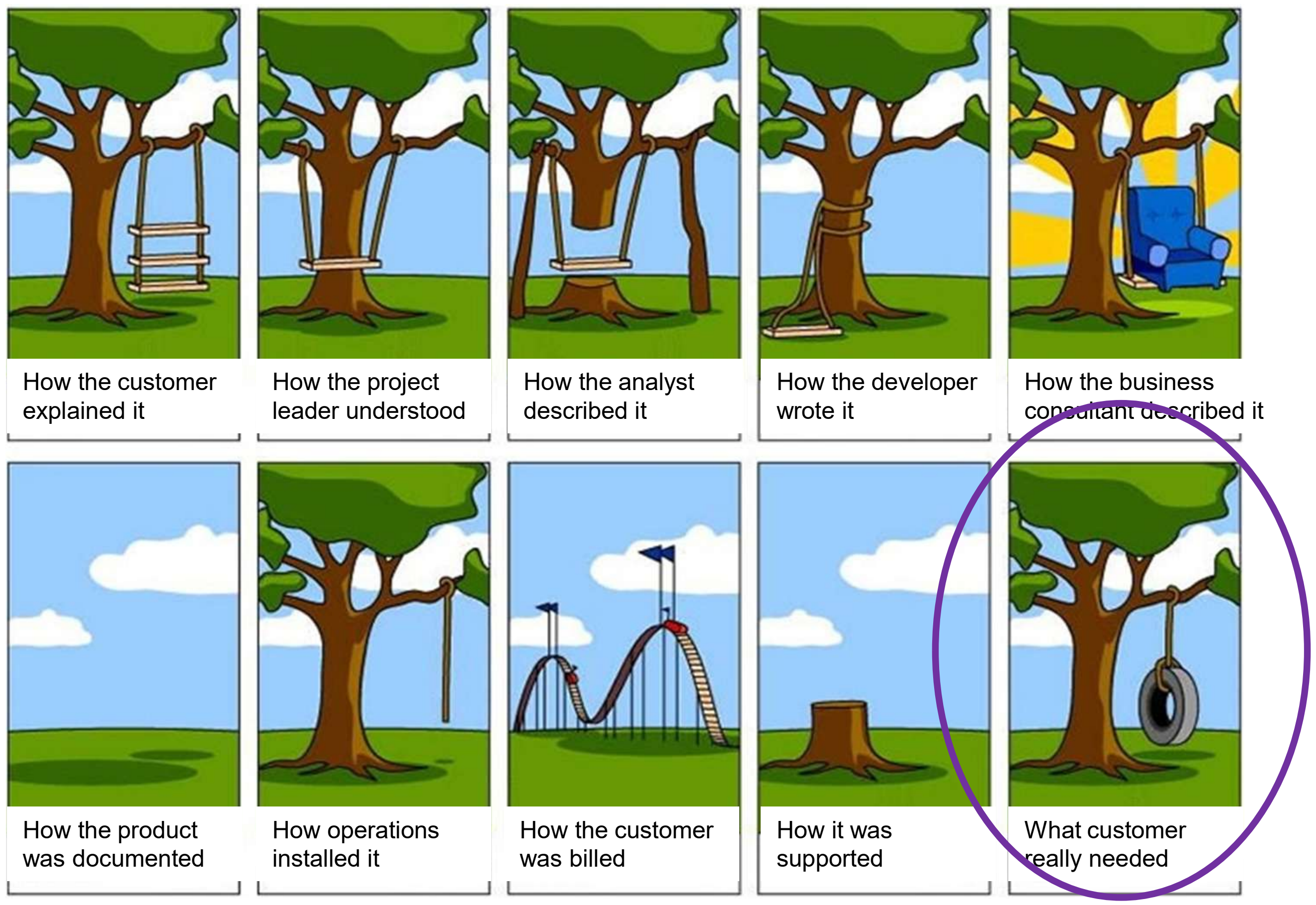
Implementation

Testing

Release

Maintenance

Sharing a visual of their importance


How the customer explained it
How the project leader understood
How the analyst described it
How the developer wrote it
How the business consultant described it
How the product was documented
How operations installed it
How the customer was billed
How it was supported
What customer really needed

What exactly are software requirements?

Requirements specify what to build

- describe **what, not how**
- describe customer needs, not how they'll be implemented
- reflect product design (product goals), not software design

Product requirements describe the product's functionality in terms understandable by customers and devs, with as close to zero ambiguity as possible

-Isaac Reynolds, Google GPM

Let’s work through an example

Are these good requirements for an audio player?

- 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

How about our swing example

What are good **and sufficient** requirements for the swing?

- Attaches to a single branch of a tree
- Seats one person 3-5ft tall
- Swings when pushed
- Appeals to environmental advocates
-
-

Timeout: how do **requirements** differ from **specifications**?

Think of as **customer requirements** and **technical specifications**

Requirements specify what to build from a **customer** perspective

Specifications specify what to build from a **developer** perspective

Requirements are hard but important

They help us:

- **Understand** precisely what is required of the software
  - Ensure the product delights the customer
- **Communicate** this understanding precisely to all involved parties
  - Common language for stakeholders
- **Monitor and control** production to confirm that system meets its specification
  - Know when the goal is reached and that it stays operable

In practice, they're used by many during SDLC

- **Customers**: what should be delivered (contractual base)
- **Project managers**: scheduling and monitoring (progress indicator)
- **Designers**: basis for a spec to design the system
- **Developers**: a range of acceptable implementations
- **QA / Testers (DevTest)**: a basis for testing, verification, and validation

Today’s Outline

1. What are requirements and what is their value?
2. **How can we gather requirements?**
3. What are techniques used to specify them?

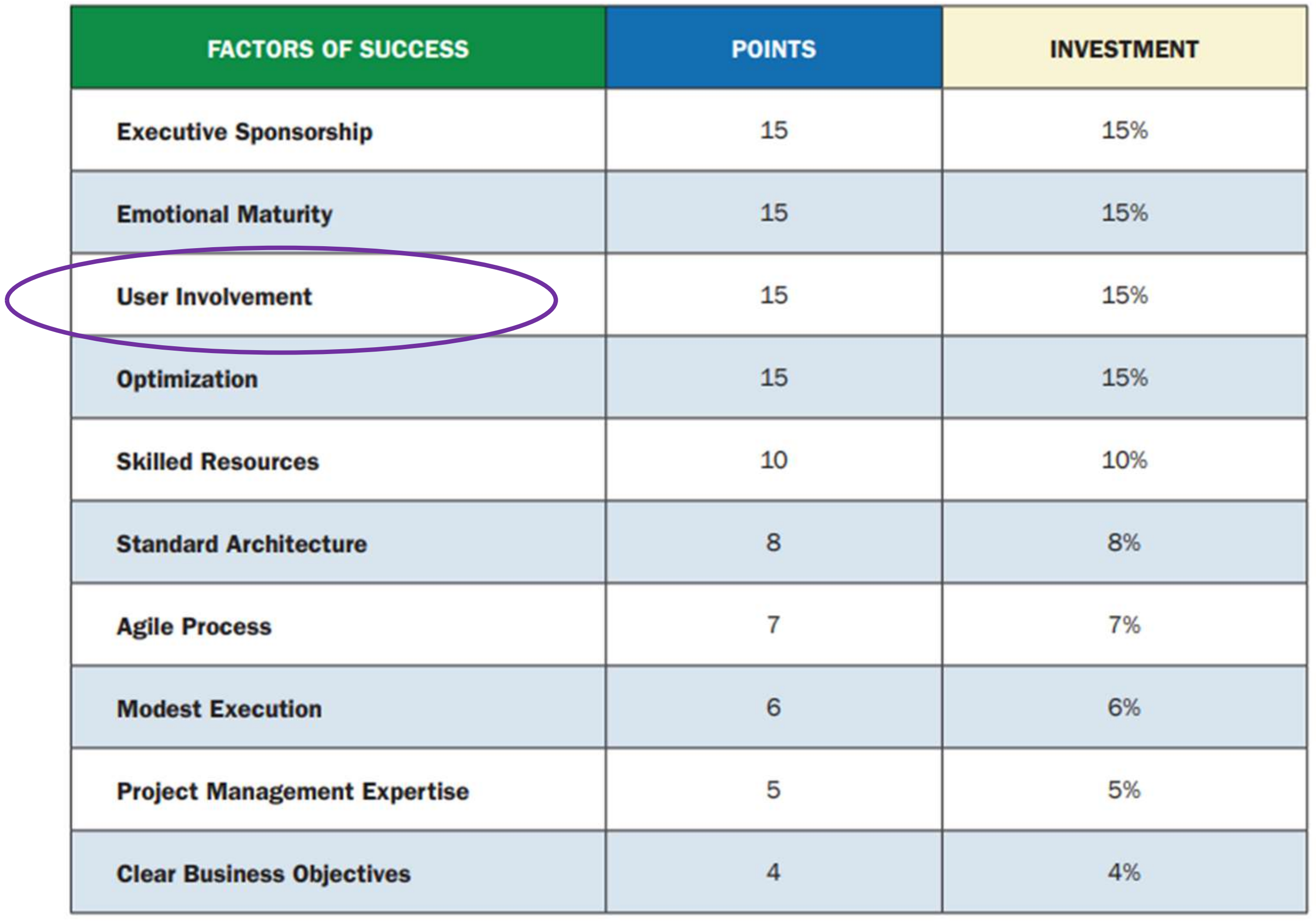
Let’s start with some data

From a Standish report on software project success

Customer involvement is 3rd highest factor of project success!

https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

CHAOS FACTORS OF SUCCESS



FACTORS OF SUCCESS	POINTS	INVESTMENT
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

The 2015 Factors of Success. This chart reflects our opinion of the importance of each attribute and our recommendation of the amount of effort and investment that should be considered to improve project success.

Successful businesses always start with the customer's goal

Marshall Field's

The customer is always right

Google

Focus on the user and all else will follow

amazon

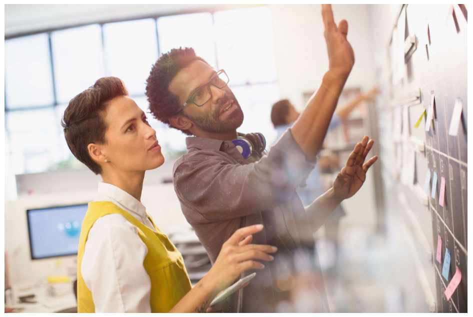
Customer obsession rather than competitor focus

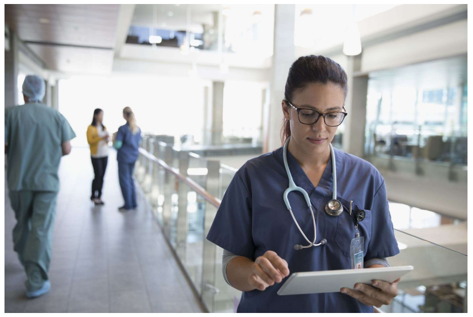
Understand and serve the customer better than anyone else

So, how do we engage with customers

Ideas?

- Be a user yourself (but be careful not to bias)
- Talk with users informally (hallway chats, mixers)
- Talk with users formally (interviews, surveys, diary studies, field studies)
- Build low-fidelity prototypes (mocks, UX prototypes, eng prototypes)
- Launch and get feedback early (“launch and iterate”)





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

Do’s and don’ts in requirements gathering

Do:

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

Do’s and don’ts in requirements gathering

Do:

- Talk to the **customers** -- to learn how they work
- Ask questions throughout the process -- "**dig**" for requirements
- Think about **why** users do something in your service, 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* (caveats apply)
- Add unnecessary features not wanted by the customers

The whole process is more formally known as requirements engineering

The science of eliciting, analyzing, documenting, and maintaining requirements

As you collect your class project requirements (02 Requirements), consider three categories:

- **Functional requirements**
  - e.g., input-output behavior, customer visible features
- **Non-functional requirements**
  - e.g., security, privacy, scalability
- **Additional constraints**
  - e.g., programming language, frameworks, testing infrastructure

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

Consider the example of a simple “camera” app.

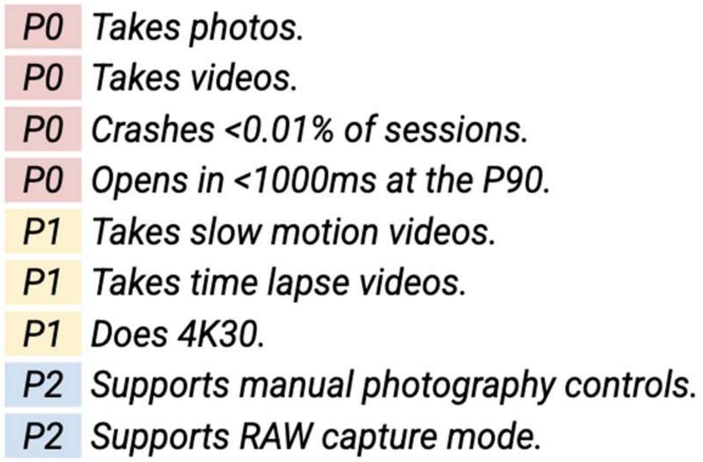
- Takes photos.
- Takes videos.
- Crashes <0.01% of sessions.
- Opens in <1000ms 90% of the time.
- Takes slow motion videos.
- Takes time lapse videos.
- Does 4K30 resolution.
- Supports manual photography controls.
- Supports RAW capture mode.

It’s essential to prioritize

If everything is a “Priority 0” (P0), then nothing is!

- P0 means we’d be embarrassed not to have this
- P1 is what makes the feature better than the competition
- P2 is nice to have

Consider the example of a simple “camera” app.



P0	*Takes photos.*
P0	*Takes videos.*
P0	*Crashes <0.01% of sessions.*
P0	*Opens in <1000ms at the P90.*
P1	*Takes slow motion videos.*
P1	*Takes time lapse videos.*
P1	*Does 4K30.*
P2	*Supports manual photography controls.*
P2	*Supports RAW capture mode.*

Today’s Outline

1. What are requirements and what is their value?
2. How can we gather requirements?
3. **What are techniques used to specify them?**
    - Use cases
    - Personas and user scenarios
    - Storyboarding
    - Paper prototyping
    - Prototyping
    - Feature list

Start with a requirements template

Cockburn's requirement 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)

https://alistaircockburn.com/

Be it the Cockburn requirements template or another – central to all – in one form or another – are **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**

Terminology

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

Use a **use case** to capture a requirement

As a [user], I want to [action] so that [result]

Use cases capture functional requirements

Camera app

1. As a parent, I want to take sharp photos of my kids in medium-low light so I can have memories of early holiday mornings.
2. As a creative, I want to adjust the look-and-feel of my photos so that they match how I remember the moment.
3. As a YouTube Shorts creator, I want to caption my videos so that people without sound don't skip my videos.
4. As a restaurateur, I want to take fresh, juicy-looking photos of food so that customers want to eat at my restaurant.

Use a more **formal use case** to describe the requirement (the goal) as a journey

A sequence of actions taken by the "system" and the "actor"

Requirements Milestone requires you to provide formal use cases

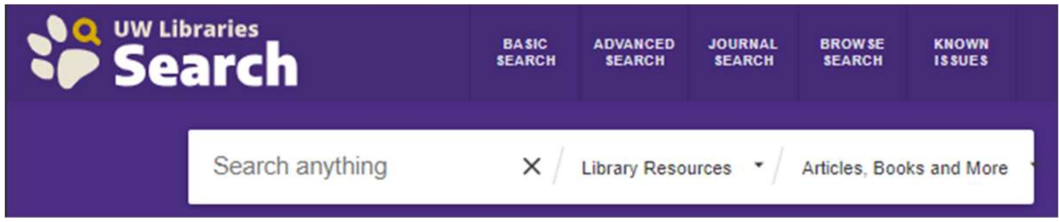
Actor: As a parent ("actor"),

Goal: I want to take pictures of my young kids where they're all smiling.

Steps:

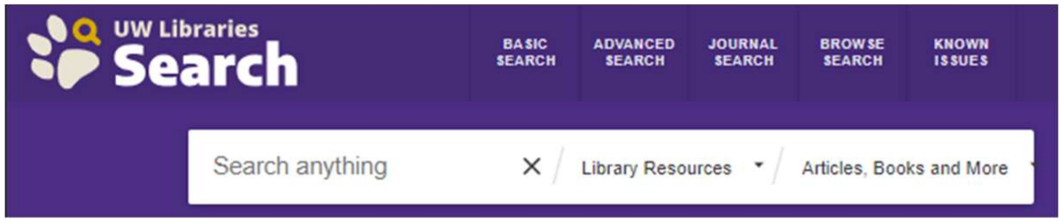
1. User arranges the family for a photo
2. User presses the shutter 3 times in 5 seconds
3. System saves 3 full-quality images
4. User taps "gallery"
5. System opens the gallery and shows a button to "select a better moment"
6. User taps the button
7. System creates and shows the "Best Take"
8. User taps "Save as copy"
9. System shows user the saved copy in the gallery

Library app example



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

Library app example



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

Flow describes interactions between actor and system

What is a use case

Use cases capture the **functional requirements** of a system

- A use case is an **example behavior** of the system
- Written from an **actor's point of view**
- **5-10 clearly written steps (flow)** lead to a “main success scenario”
- Also used to describe “variation” and “exception” scenarios

Can this requirement be described with a use case?
- The library system must have 99.9% uptime
- Checkouts may not be for longer than 30 days

What is a use case

Use cases capture the **functional requirements** of a system

- A use case is an **example behavior** of the system
- Written from an **actor's point of view**
- **5-10 clearly written steps (flow)** lead to a “main success scenario”
- Also used to describe “variation” and “exception” scenarios

Try it with a use case for your product

Capture your thoughts and we'll discuss!

Goal	
Actor	
Main (success) flow	1. 2. 3. 4. …

Use cases are hugely valuable

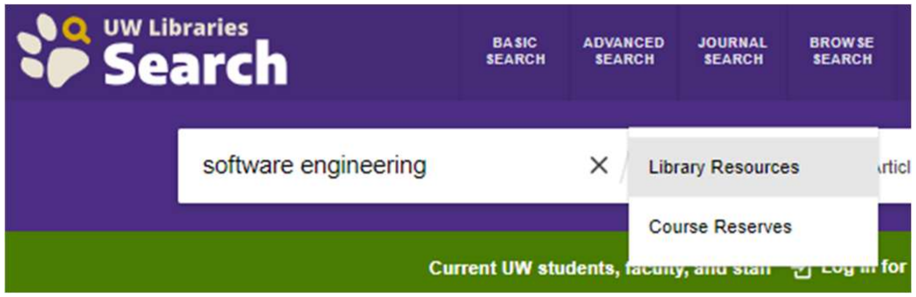
- Capture a level of functionality (**list of goals**)
- Establish an understanding between the customer and the developers of the requirements (**success scenarios**)
- Alert developers of **variations** (extensions) and **exceptions** (errors) cases to test

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:

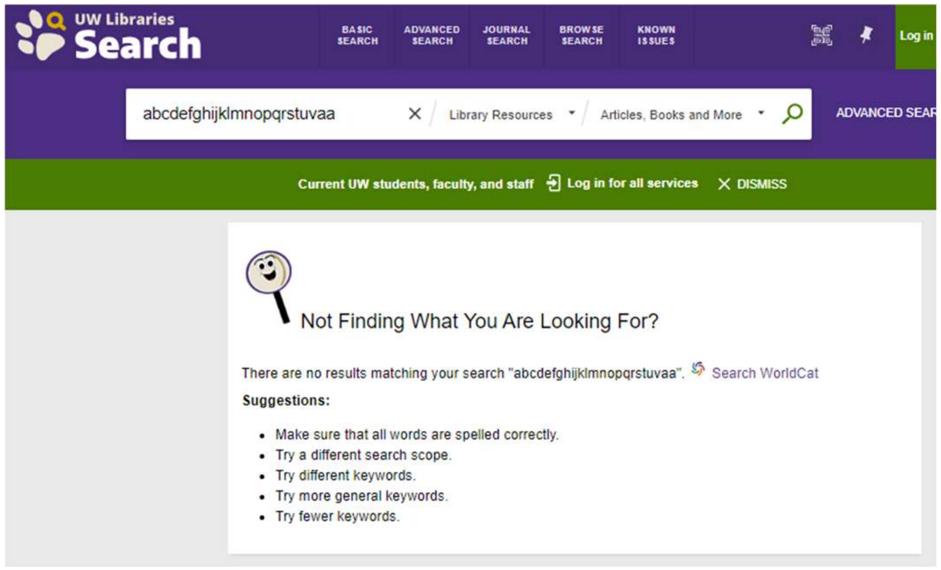
- These paths describe extensions on the main theme
- **Another way to meet the goal**
- Library search - Patron enters an author or subject or category



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



Not Finding What You Are Looking For?

There are no results matching your search "abcdefghijklmnopqrstuvaa". Search WorldCat

Suggestions:

- Make sure that all words are spelled correctly.
- Try a different search scope.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

We can capture this in our template

Goal	Reserve a book in the library app
Actor	Library patron
Main (success) flow	1. User selects the search screen 2. System presents a search box (with filters) **3. User types in the book title** 4. System presents the books that match and branch locations 5. User selects location and reserves 6. System confirms and represents home page
Variation (alternate) flow	**(In step 3)** **3.1 User types in an author ...** **3.2 User types in a subject ...**

Here’s another example – ATM machine

Goal	Withdraw money
Actor	Bank patron
System	ATM
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 returns to step 1]**

Precondition	**Authenticated in**
Trigger	**Select withdraw**

These other flows are hugely valuable

Do

- Think about how every step of the use case could be enhanced or 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 scope of 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 ☺

Back to basics – 4 steps for writing a use case

1. Identify actors and their goals

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

4 steps for writing a use case

1. Identify actors and goals
2. **For each goal, identify what each actor needs the system to do**

   Main success scenario is the preferred "happy path"
   - Easiest to read and understand

   Capture each actor's intent and responsibility, from trigger to goal
   - State what information passes between actor(s) and system
   - Number each step (line)

4 steps for writing a use case

1. Identify actors and goals
2. For each goal, identify what each actor needs the system to do
3. **List the variations to the main (success) flow**
    - These are alternate branches from the main path
    - What are some options/enhancements that the user might want/expect
    - Label with step number (success scenario line)
        - Example variation to step 5:
            - 5.1 <variation>; 5.1 <steps>; 5.1<continue at step 6>

4 steps for writing a use case

1. Identify actors and goals
2. For each goal, identify what each actor need our system to do
3. List the variations to the main flow
4. **List the exception (error) flows**
    - Many steps can fail
    - Describe failure-handling
    - Label with step number (success scenario line)
        - 5.1<failure condition>; 5.1 <actions>; 5.1<continue at failure step 7>

Try it with a use case for your product

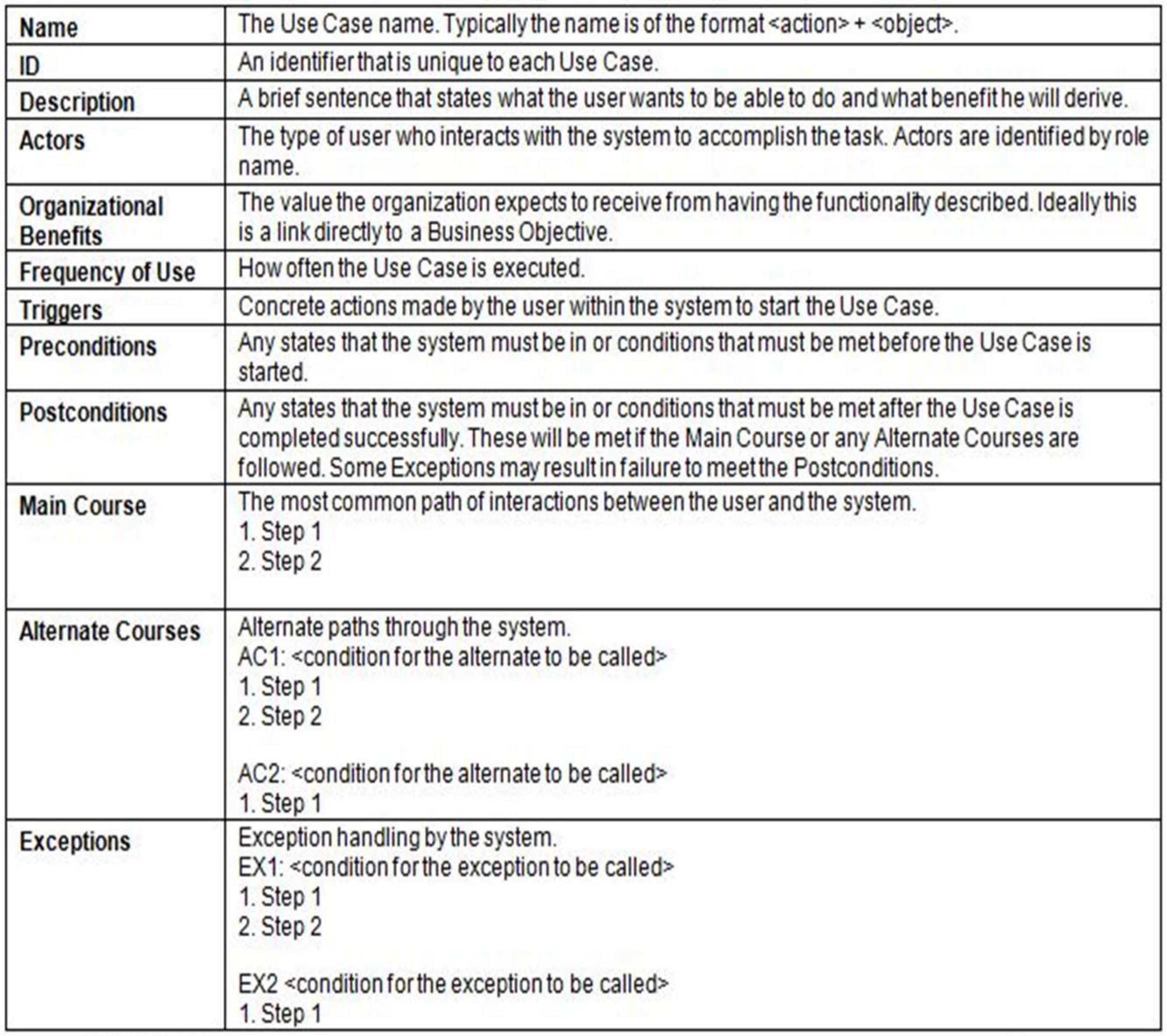
- Capture your thoughts –

Goal	
Actor	
Trigger	
Main (success) Flow	
Variation (alternate) Flow	
Exception (error) Flow	

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
  - Main success scenario, and variations and exceptions

See reference materials posted on the Class Calendar



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

What are techniques used to specify requirements?

- Use cases
- **Personas and user scenarios**
- Storyboarding
- Paper prototyping
- Prototyping
- UML
- Feature list

Personas

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

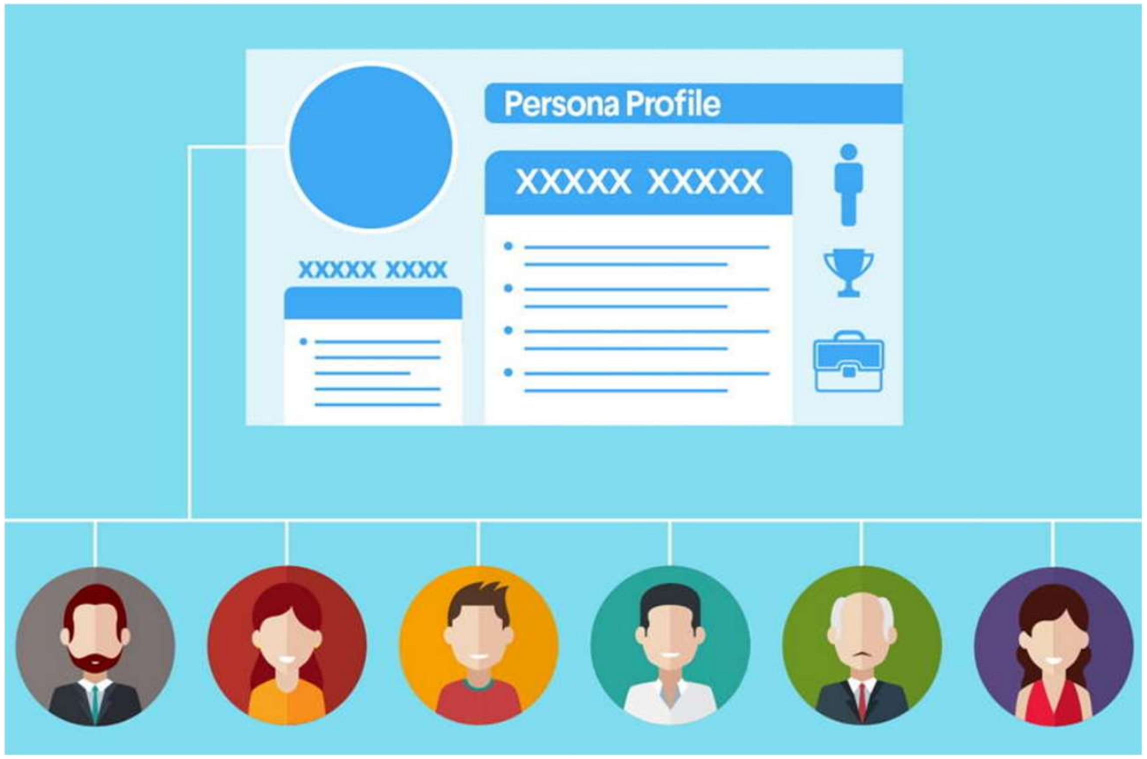
Each persona may have a different perspective of what they need

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

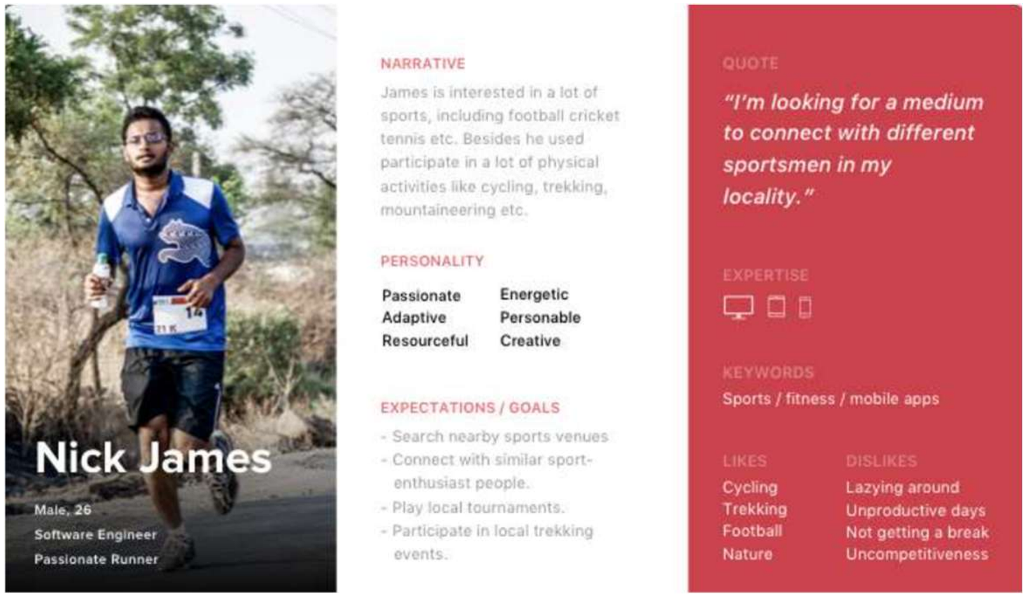


Mockplus: User Persona Templates for Free Download

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

Lots of great examples on the web



Nick James

Male, 26

Software Engineer

Passionate Runner

NARRATIVE

James is interested in a lot of sports, including football cricket tennis etc. Besides he used participate in a lot of physical activities like cycling, trekking, mountaineering etc.

PERSONALITY

Passionate	Energetic
Adaptive	Personable
Resourceful	Creative

EXPECTATIONS / GOALS

- Search nearby sports venues
- Connect with similar sport-enthusiast people.
- Play local tournaments.
- Participate in local trekking events.

QUOTE

"I'm looking for a medium to connect with different sportsmen in my locality."

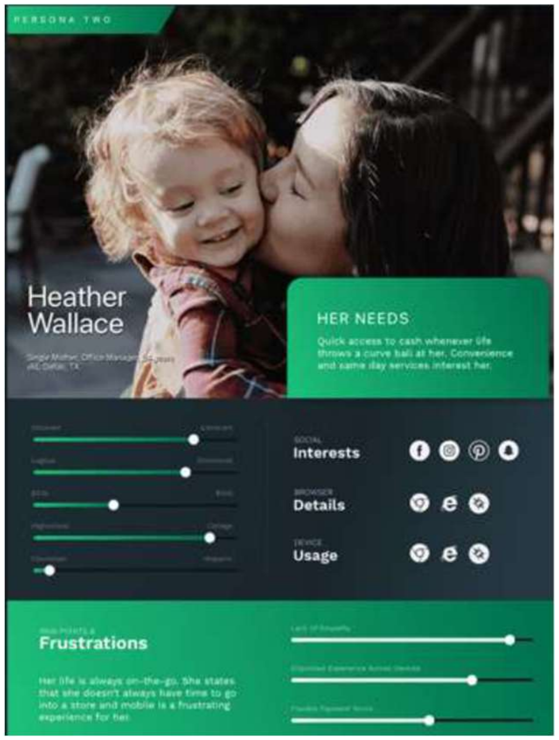
EXPERTISE

KEYWORDS

Sports / fitness / mobile apps

LIKES	DISLIKES
Cycling	Lazying around
Trekking	Unproductive days
Football	Not getting a break
Nature	Uncompetitiveness

[Mockplus: User Persona Templates for Free Download]()



PERSONA TWO

Heather Wallace

HER NEEDS

Quick access to cash whenever life throws a curve ball at her. Convenience and same day services interest her.

Interests

Details

Usage

Frustrations

Her life is always on-the-go. She states that she doesn't always have time to go into a store and mobile is a frustrating experience for her.

User scenarios

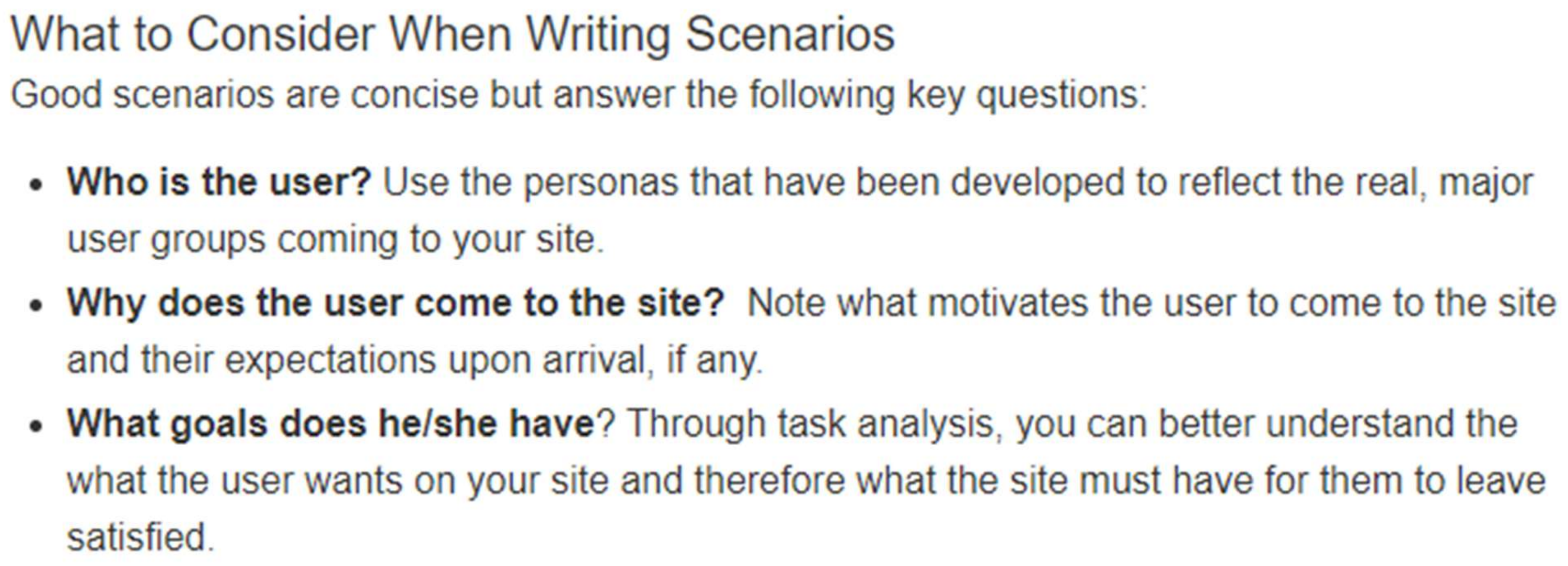
For each persona you can define the requirements from that person's perspective through a user scenario

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 industry. I'd also like each result to come with a summary for quick scanning.

Writing user scenarios

Doesn't this sound like use cases!
persona ~= actor
scenario ~= goal (w/ flow)


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.

Read: https://www.justinmind.com/blog/how-to-design-user-scenarios/

Your turn

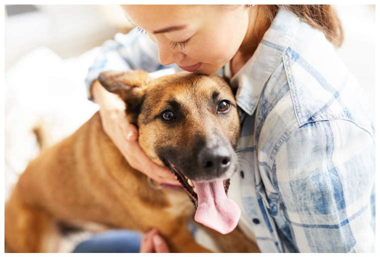
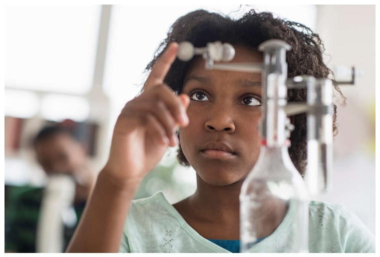
- Describe the main persona (user) of your product
- Provide a user scenario to represent a goal (a feature) that the system will light up

Persona:

Scenario:

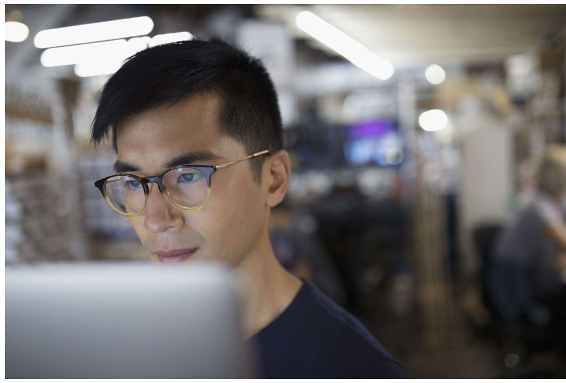
Personas and scenarios are hugely valuable

- **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

Closing thoughts: watch out for these as you requirements engineer

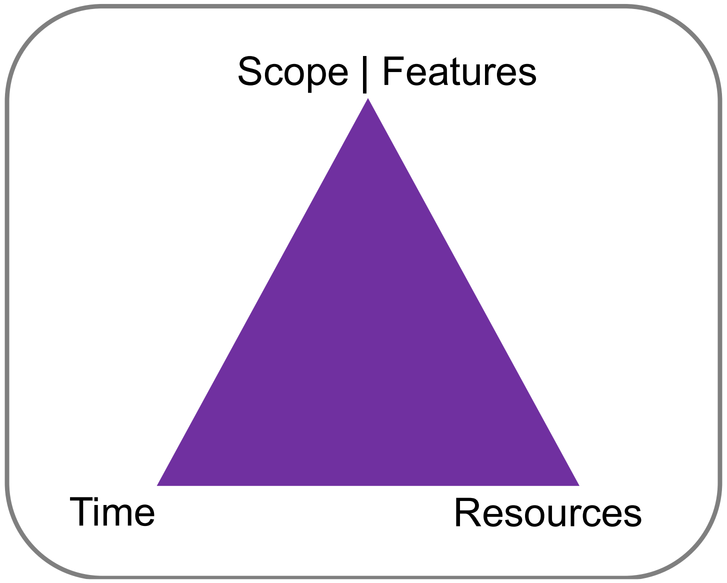
- Unclear scope leading to unclear requirements
- Finding the right balance (depends on customer, and the team):
  - Comprehensible vs. detailed
  - Graphics vs. tables and explicit and precise wording
  - Short and timely vs. complete and late
- Capturing implementation details instead of requirements
- Projecting your own models/ideas
- Feature creep



Feature creep?

For your project, consider **major** features (P0/P1s) and **stretch** features (P2s)

Feature creep is the gradual accumulation of features over time, beyond what was originally committed and/or actually needed



Project Management Triangle

Why does it happen? Because features are fun!

- Developers like to code them
- Sales teams like to pitch them
- Users (think they) want them

Why can it be bad?

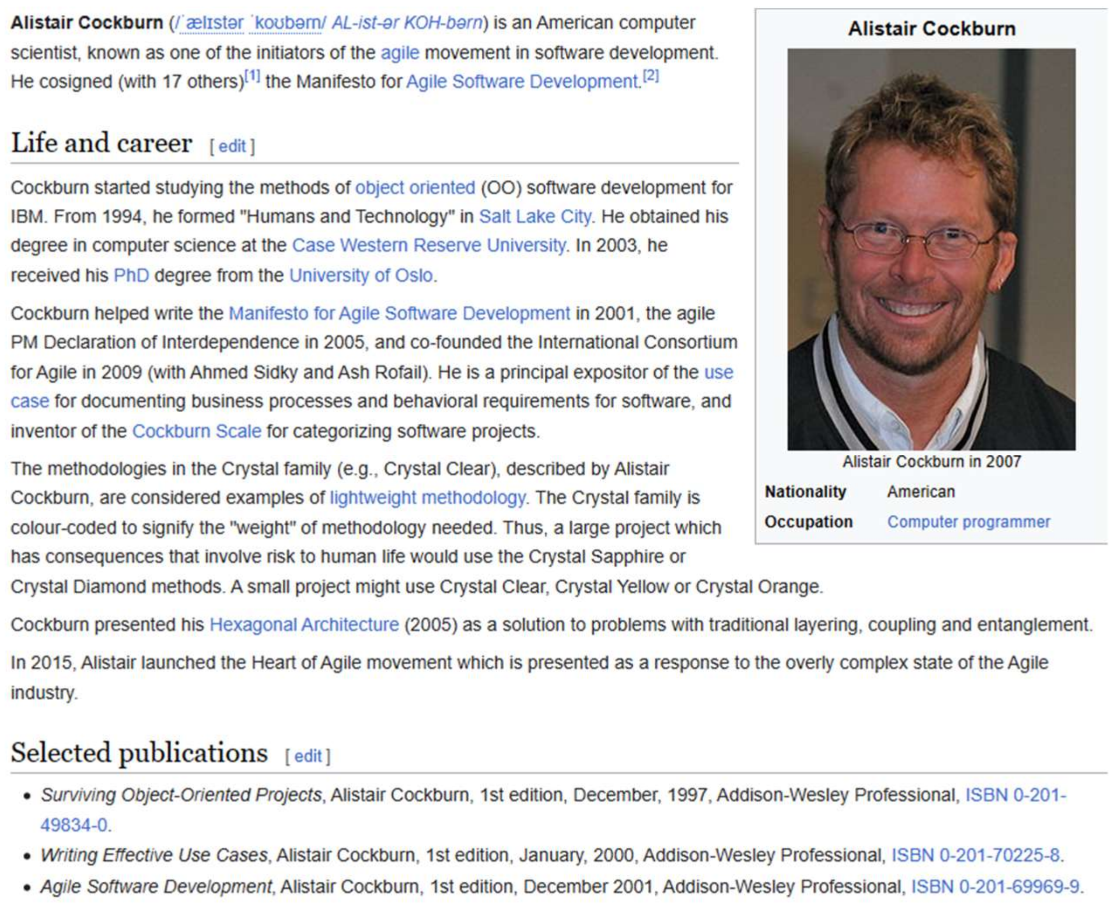
- Can put your project delivery at risk
- Too many options, more bugs, more delays, less testing, …

Additional Material

Requirements can be extensive – leveraging existing frameworks (categories, templates) can help

- User features
- Performance and System Health
- Reliability
- Scalability
- Warranties or maintenance goals
- Possible or likely future goals
- Target platforms or environments
- Regulatory and legal
- External documentation, user "help"
- Marketing claims
- Logging and success metrics
- Manual testing guides
- Accessibility
- Internationalization, localization, language support
- Troubleshooting guides
- Leak prevention
- Threat models and security guarantees
- User privacy
- Simplicity and usability

Meet Alistair Cockburn – Requirements SME



Alistair Cockburn (/ˈælɪstər ˈkoʊbərn/ *AL-ist-ər KOH-bərn*) is an American computer scientist, known as one of the initiators of the agile movement in software development. He cosigned (with 17 others)[1] the Manifesto for Agile Software Development.[2]

Alistair Cockburn	
Alistair Cockburn in 2007	
Nationality	American
Occupation	Computer programmer

Life and career [edit]

Cockburn started studying the methods of object oriented (OO) software development for IBM. From 1994, he formed "Humans and Technology" in Salt Lake City. He obtained his degree in computer science at the Case Western Reserve University. In 2003, he received his PhD degree from the University of Oslo.

Cockburn helped write the Manifesto for Agile Software Development in 2001, the agile PM Declaration of Interdependence in 2005, and co-founded the International Consortium for Agile in 2009 (with Ahmed Sidky and Ash Rofail). He is a principal expositor of the use case for documenting business processes and behavioral requirements for software, and inventor of the Cockburn Scale for categorizing software projects.

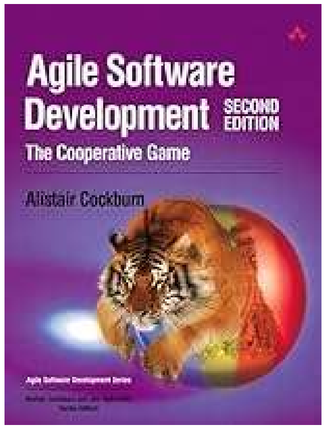
The methodologies in the Crystal family (e.g., Crystal Clear), described by Alistair Cockburn, are considered examples of lightweight methodology. The Crystal family is colour-coded to signify the "weight" of methodology needed. Thus, a large project which has consequences that involve risk to human life would use the Crystal Sapphire or Crystal Diamond methods. A small project might use Crystal Clear, Crystal Yellow or Crystal Orange.

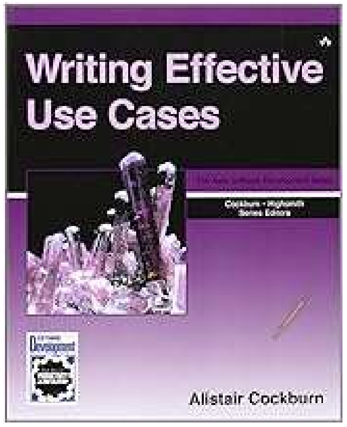
Cockburn presented his Hexagonal Architecture (2005) as a solution to problems with traditional layering, coupling and entanglement.

In 2015, Alistair launched the Heart of Agile movement which is presented as a response to the overly complex state of the Agile industry.

Selected publications [edit]

- *Surviving Object-Oriented Projects*, Alistair Cockburn, 1st edition, December, 1997, Addison-Wesley Professional, ISBN 0-201-49834-0.
- *Writing Effective Use Cases*, Alistair Cockburn, 1st edition, January, 2000, Addison-Wesley Professional, ISBN 0-201-70225-8.
- *Agile Software Development*, Alistair Cockburn, 1st edition, December 2001, Addison-Wesley Professional, ISBN 0-201-69969-9.





https://en.wikipedia.org/wiki/Alistair_Cockburn

Cockburn requirements template

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

Many companies will have a template for you to use

Uniformity is good for you and the customer