16. Requirements Engineering

Logistics ⚙️

- Layoffs hit this week
- Capstone 2 will PROBABLY still happen but today is my last day

CNBC

Meta laying off more than 11,000 employees: Read Zuckerberg's letter…

7 hours ago

Logistics ⚙️

- JK about the above (except layoffs, those did actually happen)
- **Remaining assignments are group project-related**
  - Roadmap
  - Specs doc
- HW10 is due next week
  - Please bother me if you’re having trouble!
- Extra credit assignment has been posted!

Recap: JavaScript bad, ??? good

How did our app in the last demo work? How do we send information from the frontend to the backend? How did we remove stuff from the todo list?

Recap: Changing DB schemas?

Which of the following do you NOT have to do if you want to add or remove a column from a table in your SQLite database, using the tech stack from class?

1) Add the column as an instance variable of the model class in Python
2) Restart your app
3) Manually delete the table
4) Rename the table

Recap: Changing DB schemas?

Which of the following do you NOT have to do if you want to add or remove a column from a table in your SQLite database, using the tech stack from class?

1) Add the column as an instance variable of the model class in Python
2) Restart your app
3) Manually delete the table
4) Rename the table

Poll: Help Settle a Relationship Dispute

Should John grow a mustache in November? (Assuming this is possible)

1) Yustache
2) Nustache

Agenda 📅

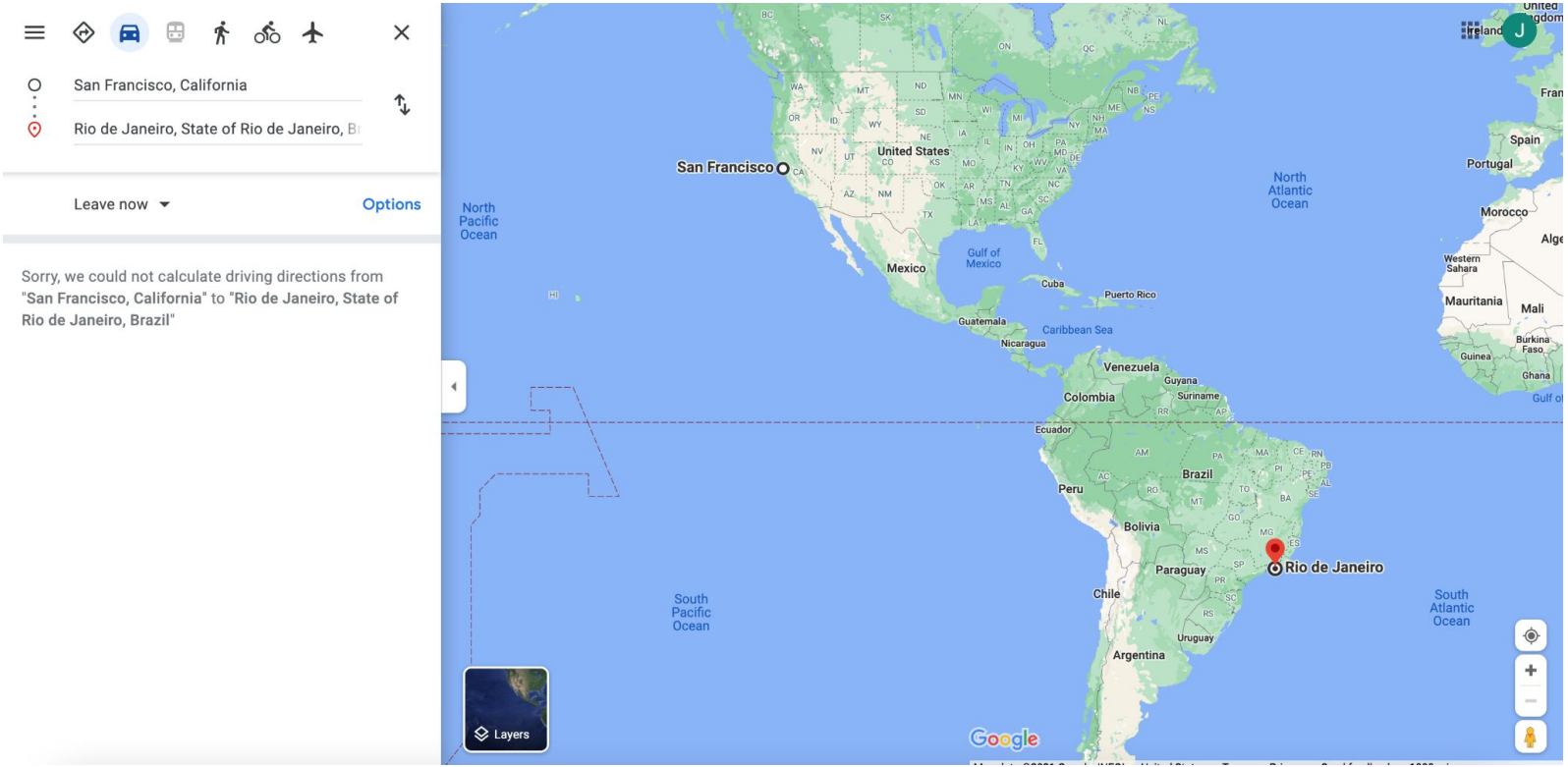
1. Requirements Engineering
2. User Stories

Today is a lot of talking. I'll try to break it up!

Why do we care? 🤔

Why is it important for us to learn how to plan and learn requirements if most of us are going into software engineering?

Story Time


San Francisco, California
Rio de Janeiro, State of Rio de Janeiro, Br
Leave now
Options
Sorry, we could not calculate driving directions from "San Francisco, California" to "Rio de Janeiro, State of Rio de Janeiro, Brazil"
San Francisco
Rio de Janeiro
United States
Mexico
Gulf of Mexico
Cuba
Puerto Rico
Caribbean Sea
Guatemala
Nicaragua
Venezuela
Guyana
Suriname
Colombia
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Argentina
Uruguay
North Pacific Ocean
South Pacific Ocean
North Atlantic Ocean
South Atlantic Ocean
Ireland
Spain
Portugal
Morocco
Western Sahara
Mauritania
Mali
Burkina Faso
Guinea
Ghana
Layers
Google

Why do we care? 🤔

Why is it important for us to learn how to plan and learn requirements if most of us are going into software engineering?

In industry...

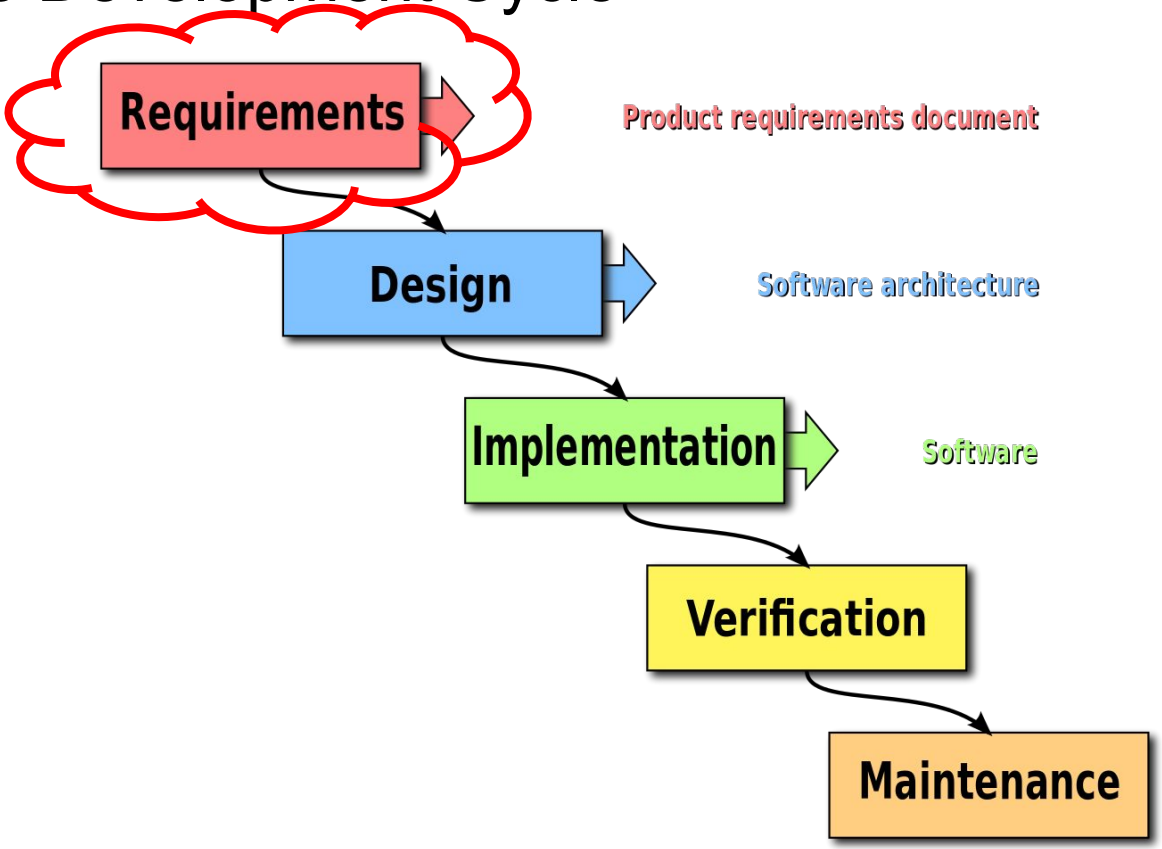
If you stay in tech, but don’t want to code, a product manager, UX researcher, or technical business analyst may be an option for you!

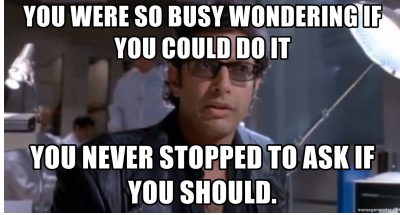
Expectations for software engineers move from “building stuff correctly” to “building the correct stuff” as you get more senior!

For personal goals...

If you do decide to start a business, you will need to do your own requirements engineering process

Software Development Cycle





Search Bloomberg Sign In Subscribe

[redacted]

Do You Need a $400 Juicer?

Silicon Valley's $400 Juicer May Be Feeling the Squeeze

Two investors in Juicero were surprised to learn the startup's juice packs could be squeezed by hand without using its high-tech machine.

By **Ellen Huet** and **Olivia Zaleski**

April 19, 2017, 5:00 AM EDT

“Software Engineering” vs. Requirements Engineering

Requirements Engineering

- What do we need to build? (Build the **right system**!)
- Specify the system that fits to the user’s problem
- Problem-oriented

Other parts of the Development Cycle

- How do we build it correctly? (Build the **system right**!)
- Design, implement, verify, maintain the system you created to fit the user’s problem
- Solution-oriented

Requirements: Questions to Ask

- Who is our user?
- What does the user want (at the moment)?
- What is the purpose of our product?
- How will we implement the product?
  - Do we need data? What type of data? Do we need to persist the data?
- Who will implement the product?
- When will we deliver the product?

Elicitation

- In practice, you will most likely do your requirements engineering by talking to customers or industry experts
- The major challenge here is asking precise enough questions that you get the information you need, but WITHOUT asking leading questions

Elicitation - Interview Example (Consumer)

- “How will you use this feature?”
  - Good question. Allows the interviewee to make their own discovery for feature use

- “Will you use this feature to….?”
  - Bad question. This is a leading question.
  - Interviewee may have never even considered to use a feature in the manner described in the question

Practice

- Think of an app that exists
- Get a partner
- Try to figure out your partner’s app by asking **five or fewer questions!**
    - This is basically what elicitation is like, in my experience. You never get as much information as you want, and you have to make creative/logical leaps to decide what to do.

What is the output of requirements engineering?

Specifications

- “A description (document or model) of a system to be developed”.
- They describe WHAT is being built, not HOW it is built
- Two extremes:
  - An informal outline of the requirements using a few paragraphs or simple diagrams
  - A long list of specifications that contain thousands of pages of intricate requirements

Specification Modeling Types

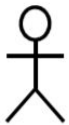
Informal language models are better understood by all stakeholders

- Good for user requirements and contracts
- Increased possibility for ambiguities
- Lack of tool support

Formal language models are more precise

- Fewer possibilities for ambiguities
- Offer tool support (e.g. automated verification and transformation)
- Intended for developers

Specification Modeling Types - Use case model (UML)

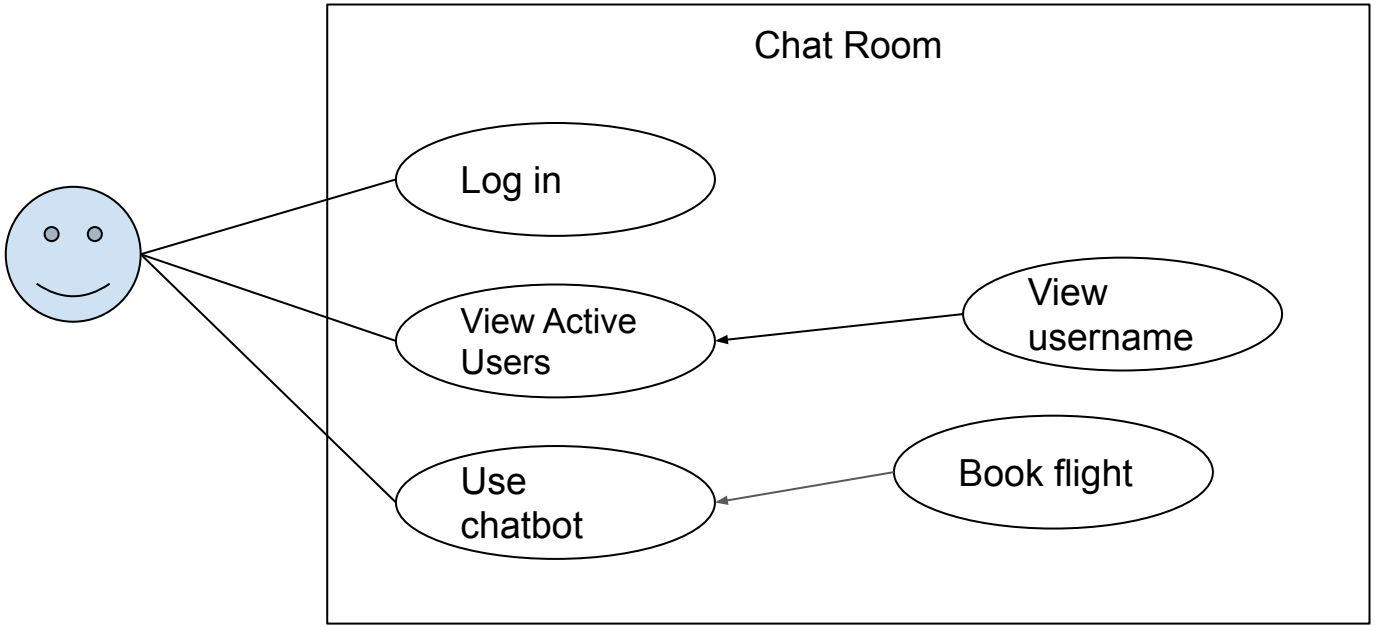


Actor: users or entities around the system that interact with the system

Use Case: describe one goal useful to one or several users

Communication: User's participation to a use-case

Specification Modeling Types - Use case model (UML)



Specifications

- The TL;DR on UML is that no one seems to really use it but we all use informal versions of it all the time.
- In this class, we're going to practice writing a Software Requirements Specification (SRS)
- It is definitely not the only way to write specs, but the format captures all critical information without being too painful to write
- Three major parts:
  - Introduction
  - Overall Description
  - System Requirements ← this is the real meat of the doc!

System Requirements Breakdown

Functional

- define the functions of the system under development. It describes what the system should do.

Non-Functional

- defines how the system should operate

System Requirements - Functional

- What inputs the system should accept
- What outputs the system should produce
- What data the system should store
- What other systems might use

- Remember, these questions aren't strictly technical!!

System Requirements - Non-Functional

Performance requirements (or system properties) - how the system should run, such as expected performance, capacity, reliability, robustness, usability, etc.

Design constraints (AKA called process requirements) - how the system should be designed and built – related to development process, documentation, programming language, maintainability, etc.

Commercial constraints - how the software should be implemented such as development time frame and costs.

Practice

Functional or non-functional?

- Your app should have a page where users can see a list of all the reviews/comments/ratings they've left.
- Your Python code should be formatted with Black.
- Your page should use React state to display and dynamically update the comments.
- Your page should not hard refresh when data is sent to the server.

Writing Great Specifications

- A great requirement in a specification ...
  - Provides specifics of a desired end goal or result
  - Contains measurable indication of the quality such as success criterion
  - Is testable
  - Is feasible (can actually be done)
  - Is viable (makes sense to be done)

Writing Bad Specifications

- Ambiguous statements
- Using vague terms
  - Flexible, quickly, user-friendly, easy
- Describing compound requirements and/or multiple systems
  - Caution using “and”, “also”, “or”, “with”

Break

Writing functional requirements

- Complex apps like the ones on your phone have a lot of different ways for users to interact with them
- Take Instagram as an example
  - Users can log in
  - Users can chat with multiple people on a thread
  - Users can scroll through a feed and see media
  - Users can video chat with people on a thread

Introducing user stories

- User stories are ways for you to write down what you expect users of your app to be able to do
  - They’re a central part of agile and agile-based development methodologies
- They are written in plain English sentences
  - As opposed to use cases, which are highly structured and may involve UML

Introducing user stories

- User stories are written from the perspective of the end user, a customer, or a stakeholder in the project
  - They focus on features that users will see, as opposed to behind-the-scenes technology
- Take note: this does not mean they're all frontend work!
  - Implementing them may very well involve backend work
  - A sample user story for HW10 could have been:
    - "As a user, I am able to see the list of books that I have saved so far."

Introducing user stories

- A lot of sources on user stories will introduce semantic “games” about how specifically you should write them.
- In this class, your only jobs with user stories are to make sure they’re written clearly and that they are scoped reasonably.

Case study: Google Docs

- As a viewer...
  - I can view public documents in order to use them for reference.
  - I can view private documents that I have the link to in order to confidentially receive information.

- As a collaborator...
  - I can invite friends to edit the document in order to have us both modify the document simultaneously.

Be specific

- As a regular user, inviting friends to a doc should work, so that we can work together on a project
  - This is vague: “should work” can mean a lot of things
- As a regular user, I should be able to collaborate on documents with others, so that we can work on a project together
  - This is vague: collaboration can mean a lot of things

Watch out for scope being too **broad** or narrow

- As a collaborator…
  - I should be able to invite, edit, and comment on docs, so that we can work together on a project
    - This is too broad, each of these are different features

- Avoid the word “and” in general
- Split this into three stories instead

Watch out for scope being too broad or **narrow**

- As a user…
  - I should be able to create a document for notes, so that I can keep track of notes in class
  - I should be able to create a document for project proposals, so that I can write project proposals

- Unless each of these are different features, this is too **narrow**
- You can join these together into one about creating empty documents

How do we know our user stories are done?

- Even the best user stories we wrote down are vague
  - What does “invite someone” mean, anyway?
- In addition to user stories, you’ll need to specify acceptance criteria
  - These are yes-or-no conditions that determine whether your user story has been implemented
  - These may also include privacy or security considerations
  - These conditions should more or less be independent of each other

Case study: Google Docs (continued...)

- As a regular user, I should be able to create a new document, so that I can jot down notes
  - Button for creating a new document is prominently on the home screen
  - Clicking on the button takes creates a new document in my drive and saves it
  - I should be shown the location of the document in my drive
  - If I am currently inside a folder and create a new document, the new document should be located inside that folder
  - The new document should be visible immediately in my Google Drive

Case study: Google Docs

- As a regular user, I should be able to create a new document, so that I can jot down notes in class
  - ...
- Privacy and security considerations
  - By default, my new document should be private and only viewable by me
  - If I am creating a new document in a shared folder, I should be warned that the document will be visible to the shared recipients

Be **specific**, but not too specific

- “X works as expected”
  - Avoid the word “works” and phrases like “as expected”; no one knows what you mean
- “X is easily doable”
  - How easy? Is there a big button? Does it take less than 3 clicks or taps?
- Watch out for putting things that are secretly additional user stories as acceptance criteria for one user story.

Be specific, but not **too** specific

- “X is fast”
  - How fast? Seconds? Minutes?
- “X is near the top, red, flashing, and 30% of the screen”
  - This is too specific -- you can simply say something like this element is prominent
  - You’ll want to communicate these layouts using mocks instead

Try it for yourself… (Participation)

1. Think of a user story for your project idea and write it down.
2. Try to think of 4 acceptance criteria for that user story.
3. Your user story can’t be related to login (too easy)!

Closing Thought: Trust No One

Closing Thought: Trust No One

- Requirements errors, conflicts, and inconsistencies
- Evolving customer/user knowledge of the system
  - Customers change their minds!
- Technical, schedule, or cost problems
- Changing customer priorities, new needs
  - May be caused by a change in the system environment (technological, business, political...), i.e., the context
- Business and strategic goals may change
  - New competitor
  - organizational changes
- Laws and regulations may change

Until next time… ✌🏽

- Finish HW10!
  - Any questions?
- Remaining project homework is due on the last day of class (Dec 2)
  - Deliverable 1 is an SRS doc
  - Deliverable 2 will be an engineering roadmap, which we'll talk about next class
- Next time, we'll talk more about the software development process (and technical interviews)!