Thinking Critically as a SWE

As a Software Engineer, it's important to consider the design of the system before you build it. What concepts or items are needed for the system to work, and how can you represent these concepts in your code? This is what the design phase of the SDLC is for. This phase follows the Requirements stage, which means we have the client expectations to act as a foundation for our implementation.

User Stories

Say you're at a Hackathon where you and your team have spent about four hours finalizing an idea, a meme database, and now you have about eight hours to get a working product. You're in charge of finding the best way to store the memes and you've come up with the following requirements:

- We need a quick way to get the memes database up and running
- It should be easy to find memes

As discussed in the SDLC Prereq, requirements may seem a little abstract, but breaking them up into user stories will make them much more feasible. User stories specify the path a user might take while interacting with your product. They traditionally take the form:

As a [stakeholder type], I want to [action], so that I can [goal],

But as explained by Gojko Adzic and David Evans, a much better format is:

In order to [goal], the system will [action] instead of [alternative action or existing implementation]

Tasks

Once you've established your user stories, they can be broken down even further into actionable tasks. Tasks include the technical details that go hand in hand with the code a developer will write, commit, and push to a shared repository. These tasks will specify the implementation details needed for a developer to get the job done.

There can be tons of ways to implement a single feature, so it's important to consider the pros and cons of each potential implementation. It's also not enough to just evaluate the pros and cons of an implementation, you also have to compare the pros and cons of each implementation in the context of your system's tech stack. This is where tradeoffs come in.

Tradeoffs play a huge role in mapping user stories to tasks. They take into consideration the pros and cons of an implementation in order to determine which one works best in the given context for the task at hand. Tradeoffs also go hand in hand with optimizations and sacrifices.

*In order to optimize for one or more aspects and gain a **pro**, you'll have to **make a sacrifice** and deal with a **con**.*

The Cambridge Dictionary uses the word tradeoff in this sentence: "For some car buyers, lack of space is an acceptable trade-off for a sporty design." Given this sentence, it's safe to say that...

*In order to optimize for aesthetics and gain **a sporty design**, some car buyers will sacrifice **a roomier cabin** and deal with **the lack of space**.*

Returning to our Hackathon example, say you have three implementations in mind: create a database using MySQL, connect to an open source meme database (that requires signing up for access to their API), or have users provide the memes via upload.

The easiest way to analyze these implementations and come up with tradeoffs by following the simple steps listed below.

Step One: Implementation Analysis

Create a pros and cons list for each implementation.

	Pros	Cons
MySQL database - create a local MySQL database server and populate it with memes	+ **Pro #1**: Easy to spin up a MySQL database server within the eight hours left in the Hackathon, optimizes for time + **Pro #2**: Custom-built and therefore flexible, so it's easy to add metadata (like categories, tags, and comments) as features are developed while progressing through the hackathon, optimizes for flexibility	- **Con #1**: It'll take time to populate the database with memes and their metadata - **Con #2**: Brings on new database administration responsibilities like permissions, schema design, network, security, etc Since the database is hosted locally,
Open Source database - setup a developer account for the meme database API and connect that way	+ **Pro #1**: An organized meme database with tags already set up, optimizes for structure + **Pro #2**: Database admin responsibilities like network and security are taken care of, optimizes for maintenance	- **Con #1**: Takes time to set up access to the database since it's hosted externally - **Con #2**: Permission is required to add or remove memes and metadata from the database
User Provided memes - have users upload the meme and its metadata via HTML, then store the image as a JPG and its metadata as JSON	+ **Pro #1**: No need to worry about database schema or details since users will upload the memes themselves, optimizes for simplicity	- **Con #1**: Input/Output with files can introduce more computation time - **Con #2**: The local machine will get clogged with JPGs and JSON files, many

	+ **Pro #2**: Ultimate flexibility since users can upload the meme and any associated metadata without adhering to any database schema, optimizes for flexibility	of which may be duplicates

Step Two: Tradeoffs Matrix

Compare the implementations based on their optimizations.

Alternative / Choice	MySQL database	Open Source database	User Provided memes
MySQL database	x	In order to optimize for… - time and gain **MySQL database Pro #1 (an easy-to-spin-up database server)** - flexibility and gain **MySQL database Pro #2 (a custom-built server that allows for metadata to be added as features are introduced)** we'll have to sacrifice… - **structure (Open Source database Pro #1: an organized meme database with tags already set up)**	Both implementations optimize for flexibility, but in order to optimize for structure and gain **MySQL database Pro #2 (a custom-built server that allows for metadata to be added as features are introduced)**, we'll have to sacrifice **simplicity (User Provided memes Pro #1: having no worries about database schema or details since users will upload the memes themselves)** and deal with… - **MySQL database Con #1 (the extra time it takes to manually**

		- **maintenance (Open Source database Pro #2: an external database solution that handles database admin responsibilities)** and deal with… - **MySQL database Con #1 (the extra time it takes to manually populate the database with memes and metadata)** - **MySQL database Con #2 (database administration responsibilities)**	**populate the database)** - **MySQL database Con #2 (database administration responsibilities)**
Open Source database	In order to optimize for… - structure and gain **Open Source database Pro #1 (an organized meme database with tags already set up)** - maintenance and gain **Open Source database Pro #2 (an external database solution that handles**	x	In order to optimize for… - structure and gain **Open Source database Pro #1 (an organized meme database with tags already set up)** - maintenance and gain **Open Source database Pro #2 (an external database solution that handles**

	database admin responsibilities) we'll have to sacrifice… - **time (MySQL database Pro #1: an easy-to-spin-up database server)** - **flexibility (MySQL database Pro #2: a custom-built server that allows for metadata to be added as features are introduced)** and deal with… - **Open Source database Con #1 (the extra time it takes to set up access to the database since it's hosted externally)** - **Open Source database Con #2 (the fact that permission is required to add or remove memes and metadata from the database)**		**database admin responsibilities)** we'll have to sacrifice… - **simplicity (User Provided memes Pro #1: having no worries about database schema or details since users will upload the memes themselves)** - **flexibility (User Provided memes Pro #2: users can upload the meme and any associated metadata without adhering to any database schema)** and deal with… - **Open Source database Con #1 (the extra time it takes to set up access to the database since it's hosted externally)** - **Open Source database Con #2 (the fact that permission is required to add or remove memes and**

			metadata from the database)
User Provided memes	Both implementations optimize for flexibility, but in order to optimize for simplicity and gain **User Provided memes Pro #1 (freedom from worries about database schema or details)**, we'll have to sacrifice **time (MySQL database Pro #1: an easy-to-spin-up database server)** and deal with… - **User Provided memes Con #1 (Extra computation time due to Input/Output with files)** - **User Provided memes Con #2 (local machine getting clogged with JPGs and JSON files, many of which may be duplicates)**	In order to optimize for… - flexibility and gain **User Provided memes Pro #1: freedom from worries about database schema or details** - simplicity and gain **User Provided memes Pro #2 (users can upload the meme and any associated metadata without adhering to any database schema)** we'll have to sacrifice… - **structure (Open Source database Pro #1: an organized meme database with tags already set up)** - **maintenance (Open Source database Pro #2: an external database solution that handles database admin responsibilities)**	x

		and deal with… - **User Provided memes Con #1 (Extra computation time due to Input/Output with files)** - **User Provided memes Con #2 (local machine getting clogged with JPGs and JSON files, many of which may be duplicates)**	

The ability to come up with tradeoffs is an underrated skill that is worth mastering! As a beginner software engineer, considering implementations and their projects is a great way to train your critical eye and consider the nuances that are introduced when working on a project.

Step Three: Tasks Creation

Once you've taken the time to analyze potential implementations and apply your critical thinking skills to create tradeoffs, it's time to create tasks. Remember, tasks help us take out user stories from statements to actionable steps. Follow along to learn how to create actionable tasks!

First, select the implementation you'd like to follow through with:

❌	MySQL database - create a local MySQL database server and populate it with memes
✅	Open Source database - setup a developer account for the meme database API and connect that way
❌	User Provided memes - have users upload the meme and its metadata via HTML, then store the image as a JPG and its metadata as JSON

Now take note of *why* you've decided to go with this implementation. This will keep you honest as you work. If you know why you're doing something, you'll gain a deeper understanding of the actions you're taking (which you'll thank yourself for when a recruiter or interviewer asks about why you wrote your code the way you did!)

To optimize for structure and durability, we'll implement the Open Source database. We can follow the instructions given by the open source database and get connected, which will take some time, but will guarantee that we have a working source of memes to pull from!

Finally, break the implementation down into bite-sized tasks.

Setup a developer account for the meme database API and connect

- Register a developer account for the open source database's API
- Store developer credentials safely in a configuration file
- Create a class that uses the developer credentials to open source database
- Write methods/functions for each of the CRUD operations

And you're ready to roll! Now you have actionable tasks you can focus on for the rest of the Hackathon. Even if you don't get through all of them, it's always worth it to take the time to evaluate your options and formulate a plan. The ability to demonstrate these skills is worth more than simply having a working product!