[CS3704] Software Engineering

Shawal Khalid

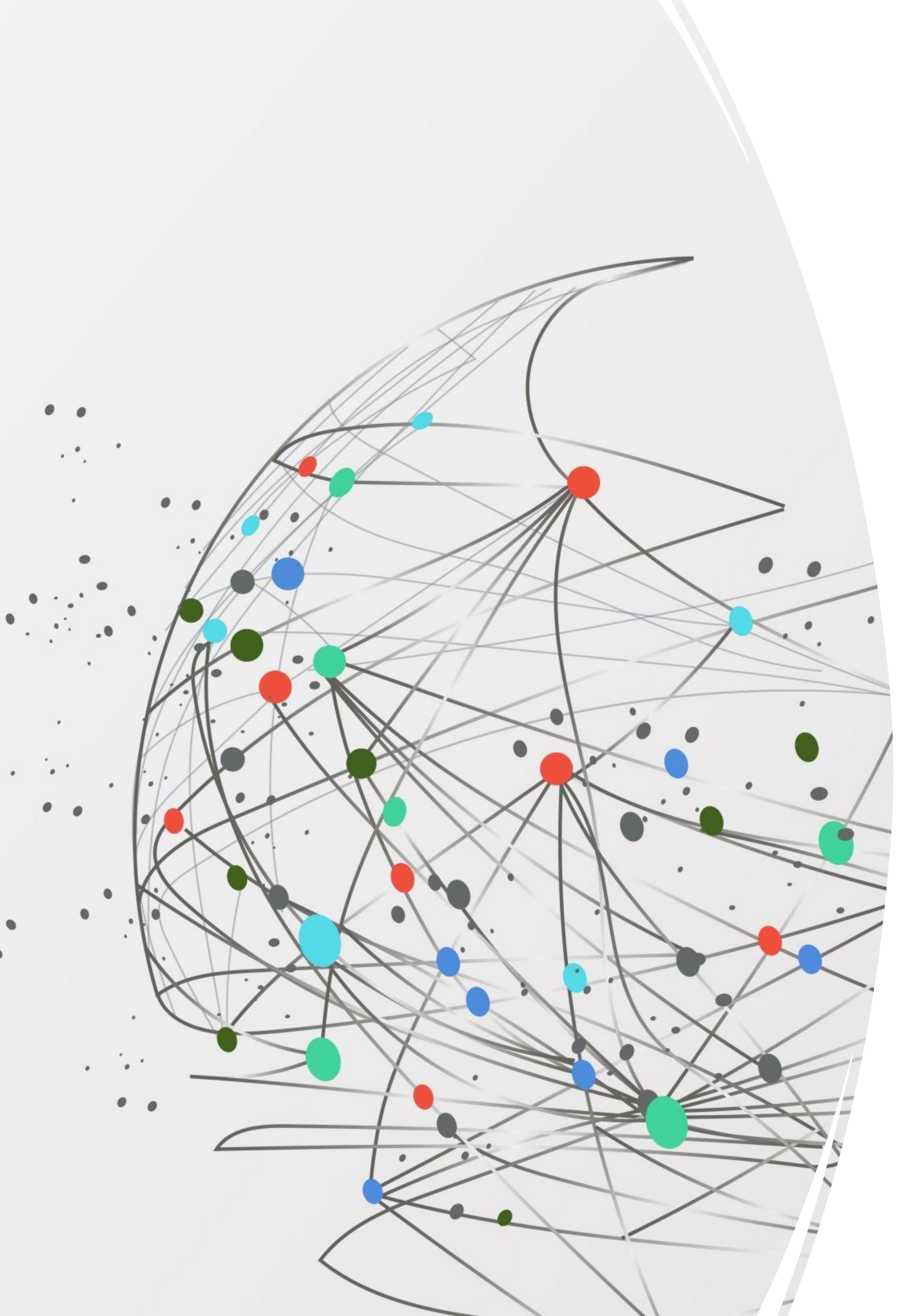
Virginia Tech

3/27/2024

Announcements

PM3 due by April 08 at 11:59pm.

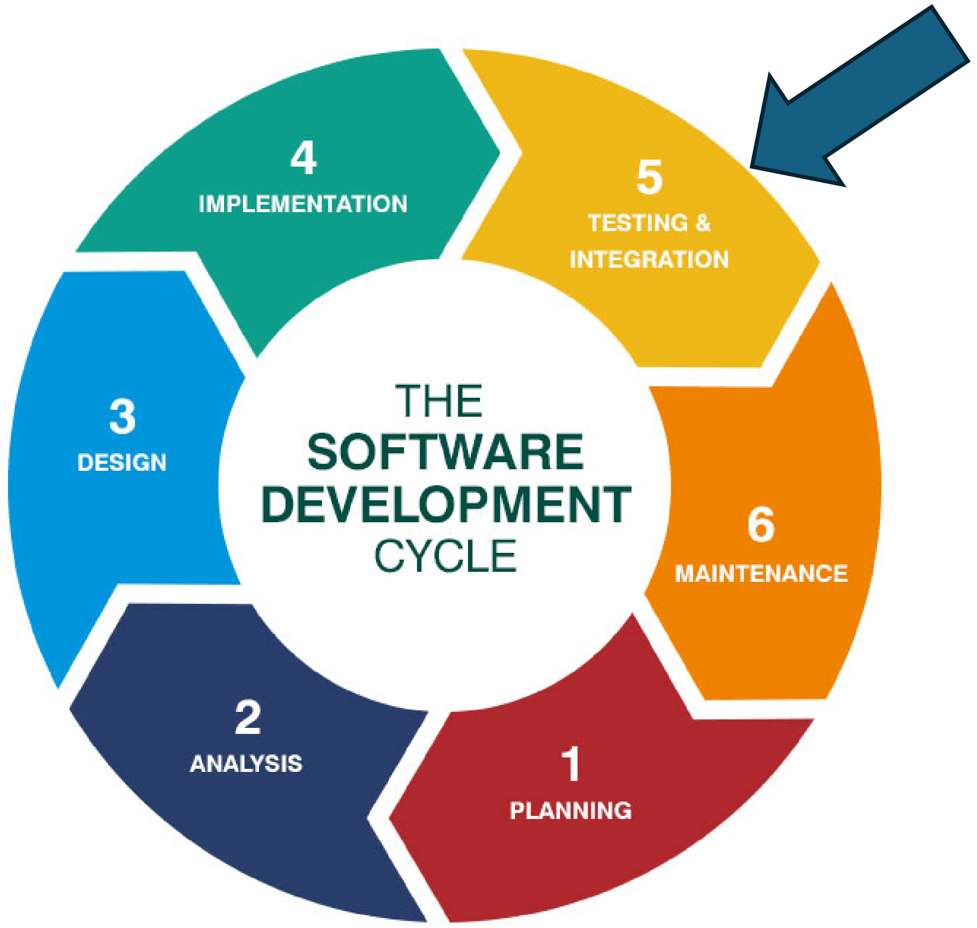
Mid semester project check in survey!



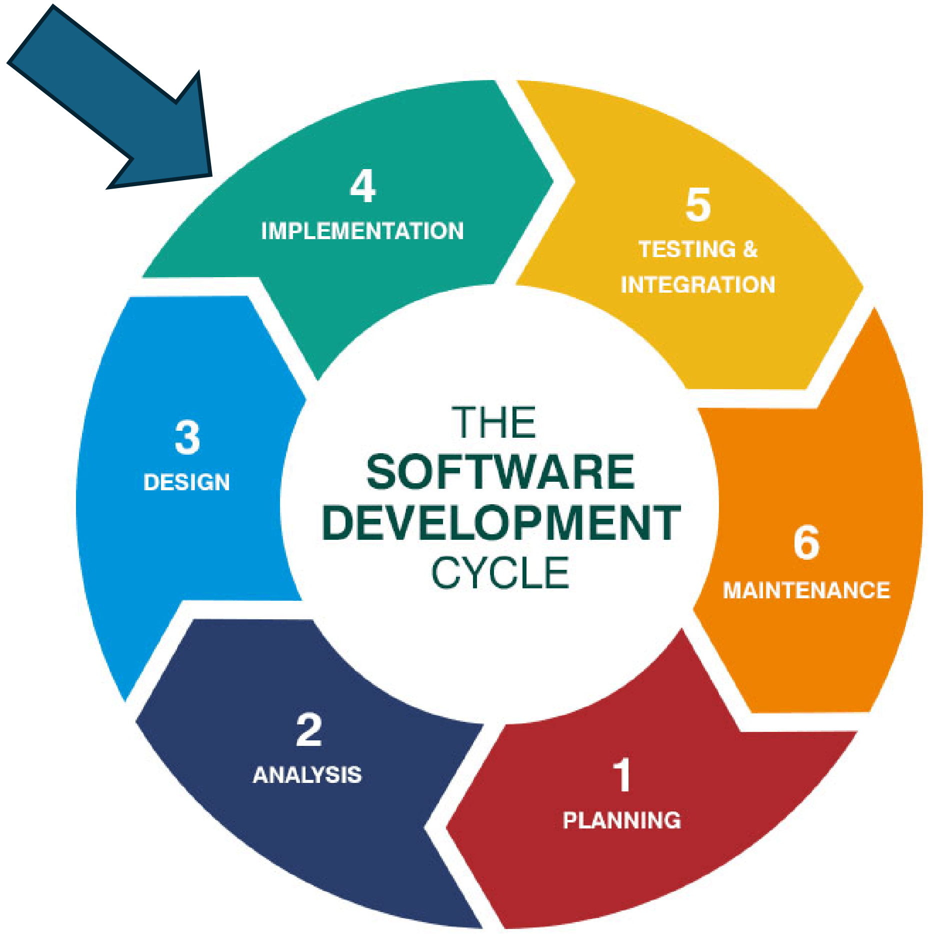
A Qualitative Study on the Implementation Design Decisions of Developers

Jenny T. Liang, Maryam Arab, Minhyuk Ko, Amy J. Ko, Thomas D. LaToza

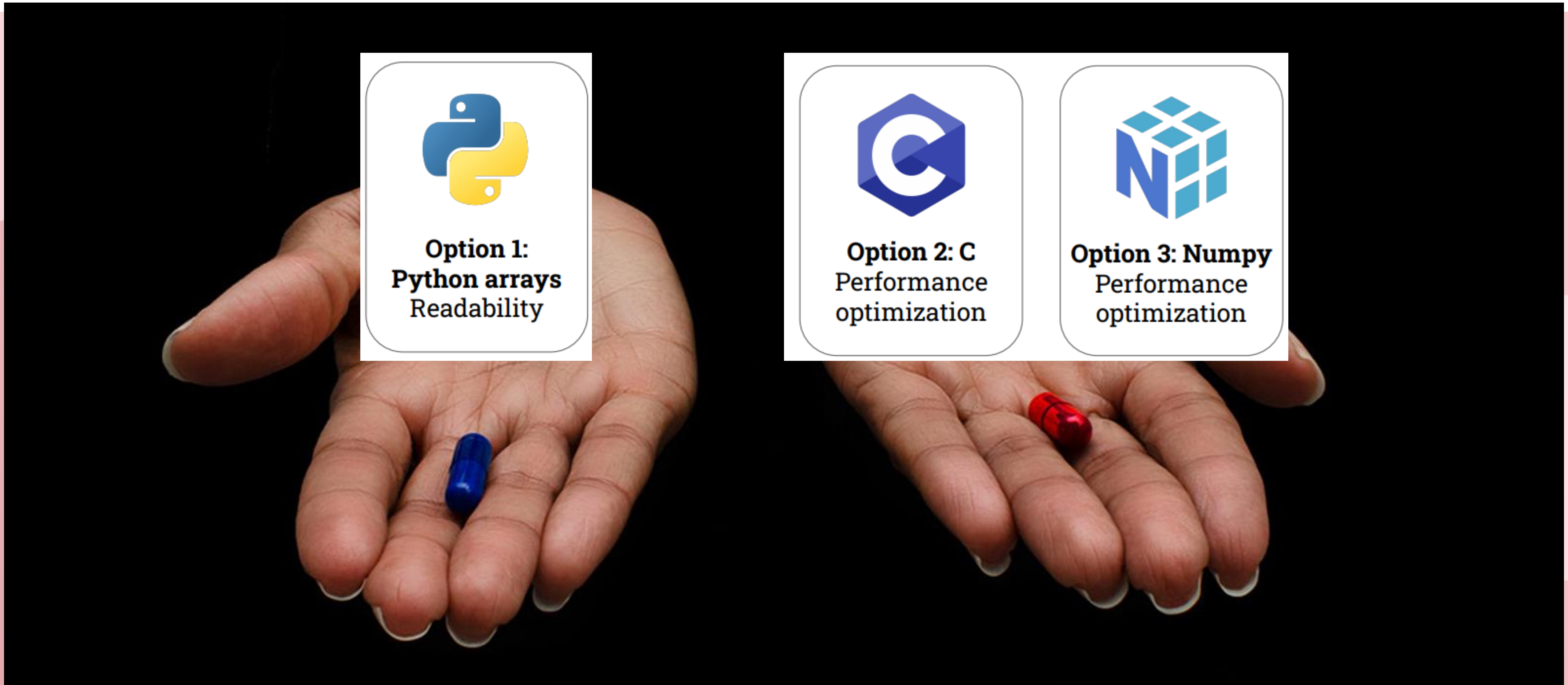
Developers Constantly Make Decisions



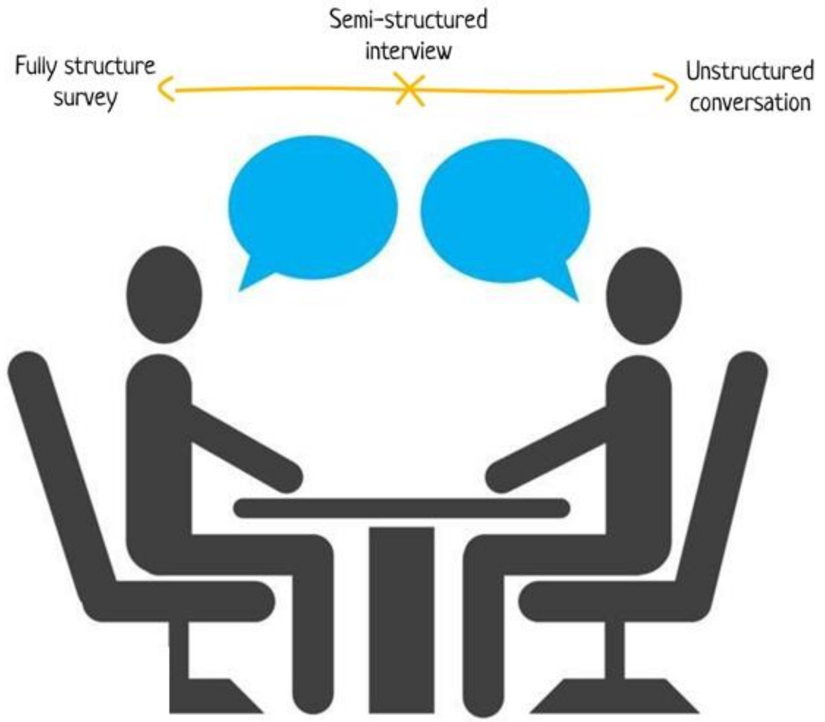
Implementation Design Decisions

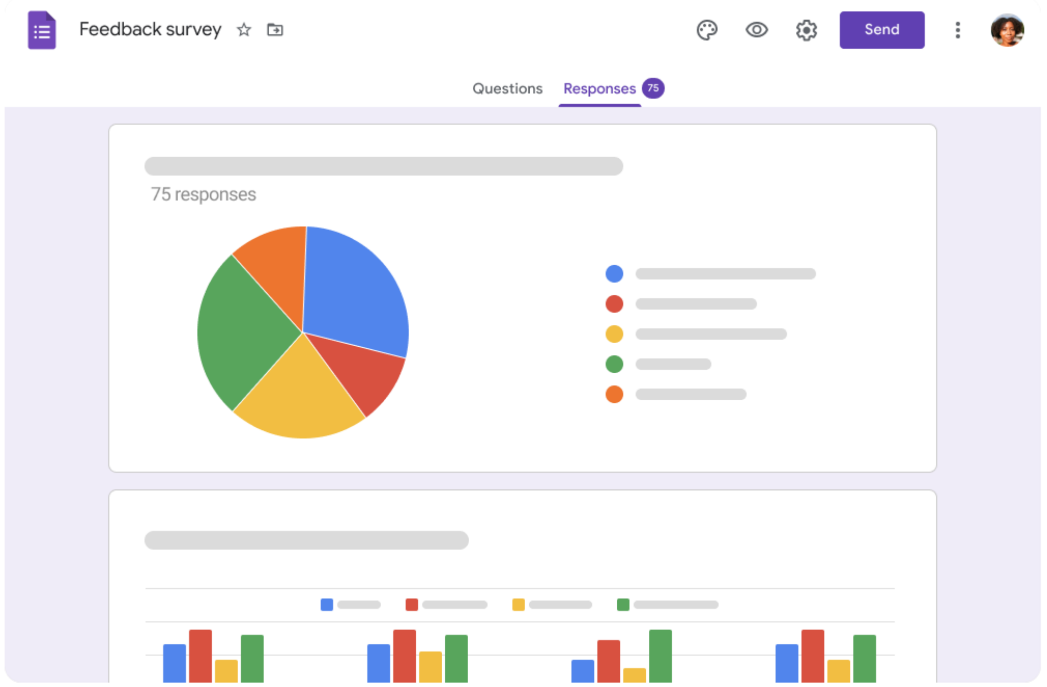


Decision: How should I represent my matrix data—Python arrays, C++ arrays, or third-party libraries?



Study Design





Overview of Interview Participants

- Average # projects: 43
- Average Org. Size: 1680
- Gender ratio: 11:3

Example Participant Data

Use this when: Using less common features in libraries instead of using the popular functions

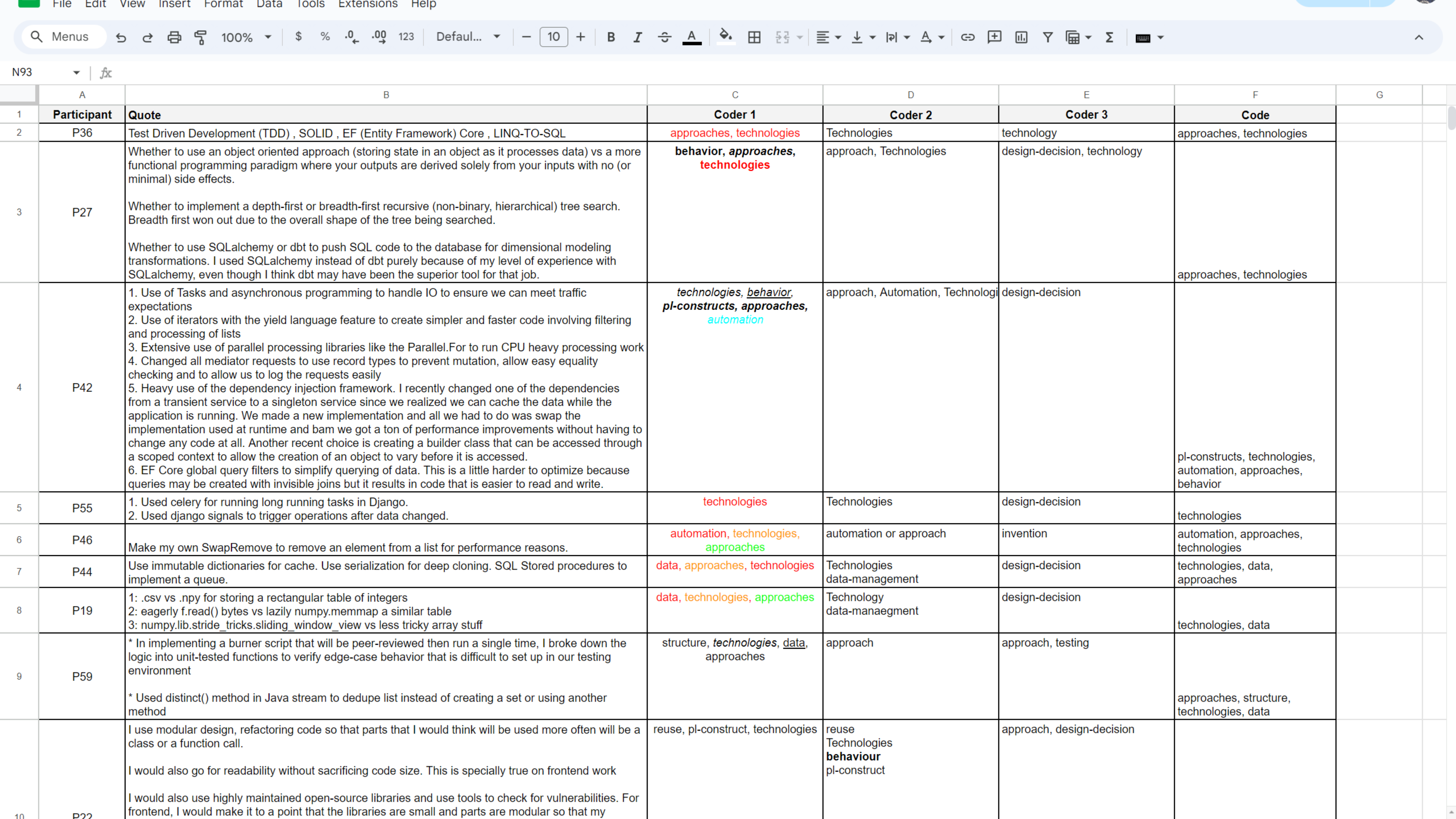
Tools/technologies: StackOverflow, Google, continuous learning

Prior knowledge: Common design patterns, popular libraries

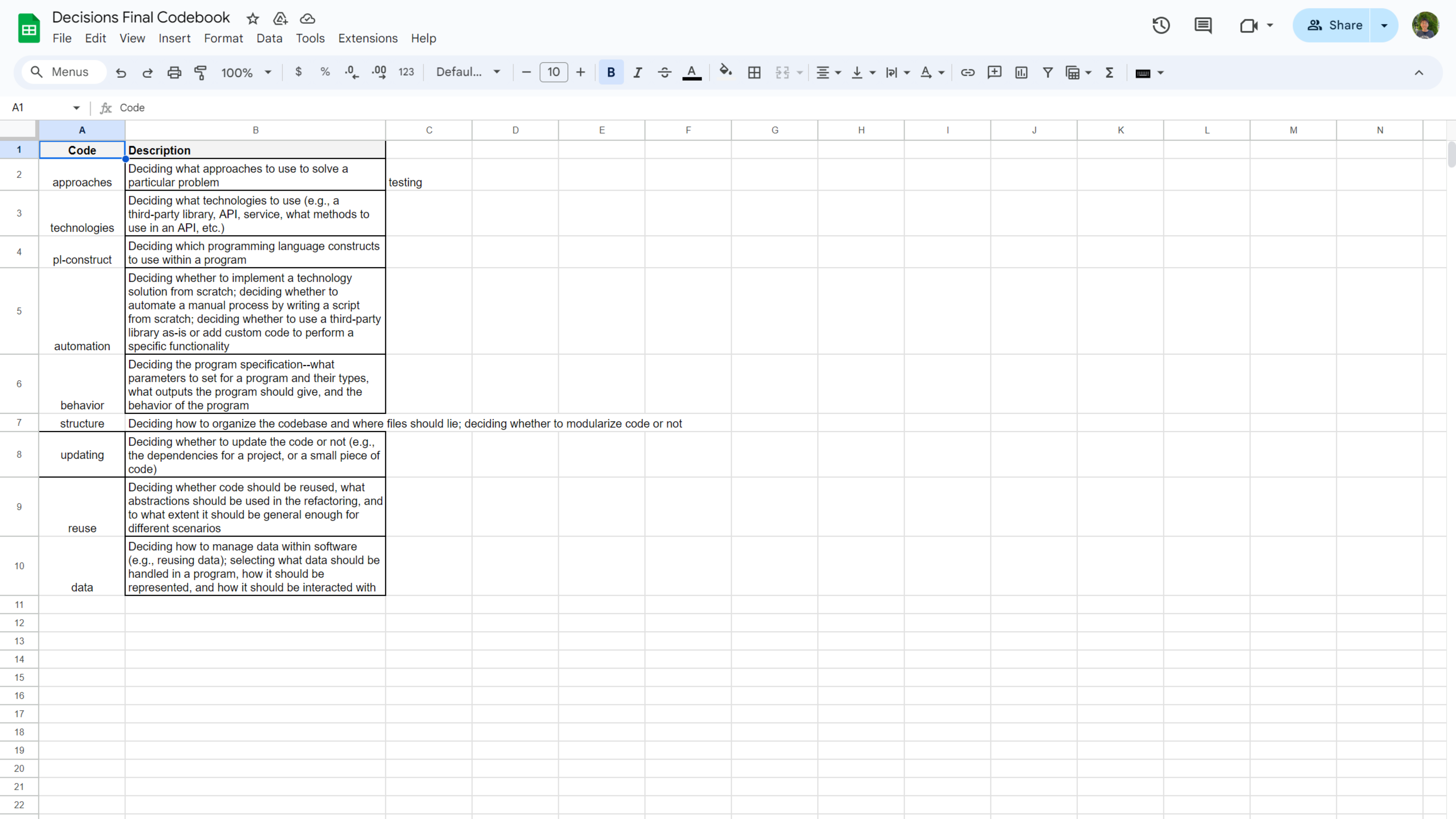
1) Decide what the goal of the program is.
2) Begin writing the program.
3) While writing the program, search online whether other libraries support your use case…
4) Choose a library which meets your use case….
5) Look at the features of the library and test the ones that you're interested in on small examples. Get a feel of the library and select a solution which achieves the desired behavior.
6) If you have code that works, show the solution to another individual for review.

Open Coding and Closed Coding

- Each of the group members code without seeing each other’s code.
- Merge them into one code, allowing to sort the responses to categories and groups.
- Makes it easy to analyze the responses.

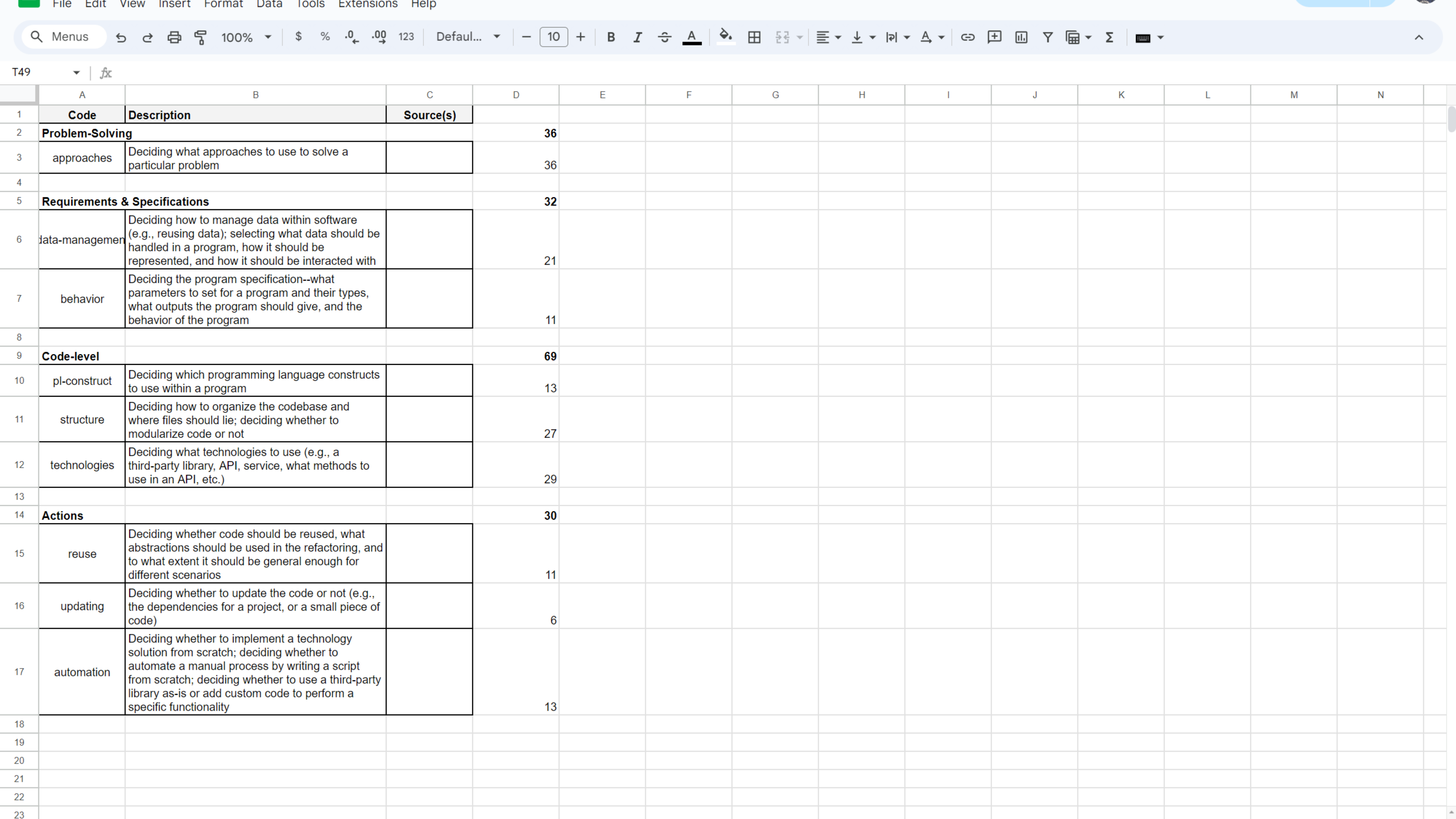


	Participant	Quote	Coder 1	Coder 2	Coder 3	Code
2	P36	Test Driven Development (TDD) , SOLID , EF (Entity Framework) Core , LINQ-TO-SQL	approaches, technologies	Technologies	technology	approaches, technologies
3	P27	Whether to use an object oriented approach (storing state in an object as it processes data) vs a more functional programming paradigm where your outputs are derived solely from your inputs with no (or minimal) side effects. Whether to implement a depth-first or breadth-first recursive (non-binary, hierarchical) tree search. Breadth first won out due to the overall shape of the tree being searched. Whether to use SQLalchemy or dbt to push SQL code to the database for dimensional modeling transformations. I used SQLalchemy instead of dbt purely because of my level of experience with SQLalchemy, even though I think dbt may have been the superior tool for that job.	**behavior, *approaches*, technologies**	approach, Technologies	design-decision, technology	approaches, technologies
4	P42	1. Use of Tasks and asynchronous programming to handle IO to ensure we can meet traffic expectations 2. Use of iterators with the yield language feature to create simpler and faster code involving filtering and processing of lists 3. Extensive use of parallel processing libraries like the Parallel.For to run CPU heavy processing work 4. Changed all mediator requests to use record types to prevent mutation, allow easy equality checking and to allow us to log the requests easily 5. Heavy use of the dependency injection framework. I recently changed one of the dependencies from a transient service to a singleton service since we realized we can cache the data while the application is running. We made a new implementation and all we had to do was swap the implementation used at runtime and bam we got a ton of performance improvements without having to change any code at all. Another recent choice is creating a builder class that can be accessed through a scoped context to allow the creation of an object to vary before it is accessed. 6. EF Core global query filters to simplify querying of data. This is a little harder to optimize because queries may be created with invisible joins but it results in code that is easier to read and write.	*technologies*, *behavior*, ***pl-constructs, approaches,*** *automation*	approach, Automation, Technologi	design-decision	pl-constructs, technologies, automation, approaches, behavior
5	P55	1. Used celery for running long running tasks in Django. 2. Used django signals to trigger operations after data changed.	technologies	Technologies	design-decision	technologies
6	P46	Make my own SwapRemove to remove an element from a list for performance reasons.	automation, technologies, approaches	automation or approach	invention	automation, approaches, technologies
7	P44	Use immutable dictionaries for cache. Use serialization for deep cloning. SQL Stored procedures to implement a queue.	data, approaches, technologies	Technologies data-management	design-decision	technologies, data, approaches
8	P19	1: .csv vs .npy for storing a rectangular table of integers 2: eagerly f.read() bytes vs lazily numpy.memmap a similar table 3: numpy.lib.stride_tricks.sliding_window_view vs less tricky array stuff	data, technologies, approaches	Technology data-manaegment	design-decision	technologies, data
9	P59	* In implementing a burner script that will be peer-reviewed then run a single time, I broke down the logic into unit-tested functions to verify edge-case behavior that is difficult to set up in our testing environment * Used distinct() method in Java stream to dedupe list instead of creating a set or using another method	structure, *technologies*, data, approaches	approach	approach, testing	approaches, structure, technologies, data
10	P22	I use modular design, refactoring code so that parts that I would think will be used more often will be a class or a function call. I would also go for readability without sacrificing code size. This is specially true on frontend work I would also use highly maintained open-source libraries and use tools to check for vulnerabilities. For frontend, I would make it to a point that the libraries are small and parts are modular so that my	reuse, pl-construct, technologies	reuse Technologies **behaviour** pl-construct	approach, design-decision	



Decisions Final Codebook

Code	Description	
approaches	Deciding what approaches to use to solve a particular problem	testing
technologies	Deciding what technologies to use (e.g., a third-party library, API, service, what methods to use in an API, etc.)	
pl-construct	Deciding which programming language constructs to use within a program	
automation	Deciding whether to implement a technology solution from scratch; deciding whether to automate a manual process by writing a script from scratch; deciding whether to use a third-party library as-is or add custom code to perform a specific functionality	
behavior	Deciding the program specification--what parameters to set for a program and their types, what outputs the program should give, and the behavior of the program	
structure	Deciding how to organize the codebase and where files should lie; deciding whether to modularize code or not	
updating	Deciding whether to update the code or not (e.g., the dependencies for a project, or a small piece of code)	
reuse	Deciding whether code should be reused, what abstractions should be used in the refactoring, and to what extent it should be general enough for different scenarios	
data	Deciding how to manage data within software (e.g., reusing data); selecting what data should be handled in a program, how it should be represented, and how it should be interacted with	



Code	Description	Source(s)	
Problem-Solving			**36**
approaches	Deciding what approaches to use to solve a particular problem		36
Requirements & Specifications			**32**
data-managemen	Deciding how to manage data within software (e.g., reusing data); selecting what data should be handled in a program, how it should be represented, and how it should be interacted with		21
behavior	Deciding the program specification--what parameters to set for a program and their types, what outputs the program should give, and the behavior of the program		11
Code-level			**69**
pl-construct	Deciding which programming language constructs to use within a program		13
structure	Deciding how to organize the codebase and where files should lie; deciding whether to modularize code or not		27
technologies	Deciding what technologies to use (e.g., a third-party library, API, service, what methods to use in an API, etc.)		29
Actions			**30**
reuse	Deciding whether code should be reused, what abstractions should be used in the refactoring, and to what extent it should be general enough for different scenarios		11
updating	Deciding whether to update the code or not (e.g., the dependencies for a project, or a small piece of code)		6
automation	Deciding whether to implement a technology solution from scratch; deciding whether to automate a manual process by writing a script from scratch; deciding whether to use a third-party library as-is or add custom code to perform a specific functionality		13

RQ1: Decision Types (9 total)

Decision Type	**Description**
Behaviors	Deciding the program specification (e.g., parameters or returns of a method).
Code constructs	Deciding which programming language constructs to use within a program.
Structure	Deciding how to organize the codebase, where files should lie, and how code should be modularized.
Languages, APIs, services	Deciding the programming languages, APIs, or third-party services to use in the software system or script.
Automation	Deciding whether to implement a technology solution from scratch.
Updates	Deciding whether to update the software or not.

RQ2: Considerations (25 total)

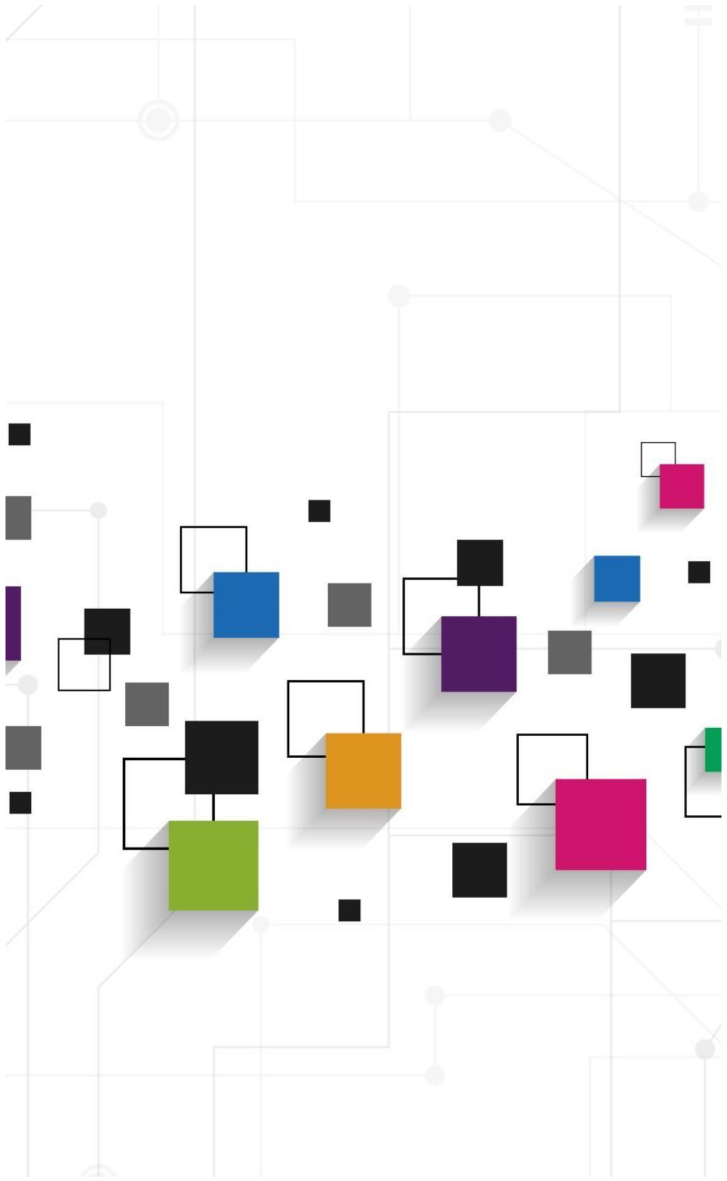
Code	Description
Community support	How well-supported by a developer community a technology is.
Consistency	Being consistent with the code style of the programming language or code base
Impacts	The impacts that the implementation may cause
Future requirements	Requirements or customer needs that may or may not occur in the future.
Maintainability	How easily maintenance actions (e.g., fixing defects, updating components) can be performed on software
System fit	How well the implementation fits in with an existing code base or system.
Requirements	The requirements of the software; customer needs.
Reliability	How reliable and correct the software is
Reusing resources	Reusing existing resources (e.g., code, practices).

RQ3: Process (15 total)

Decision Type	Description
Updating Requirement	Updating the requirements of the solution after they are initially defined.
Brainstorming	Brainstorming potential solutions that could solve the problem
Evaluating	Evaluating the developer's current situation; considering the pros and cons for each solution.
Proof-of-Concept	Building a proof-of-concept for a potential solution.
Implementing	Implementing a particular solution.
Researching	Learning more about the problem or potential solutions.

Participants’ Strategies and Containing Actions

Action	Median Position	% Strategies w/ Action
Providing Context	1.5	12.5
Researching	2	75.0
Defining Requirements	2	81.3
Brainstorming	3	62.5
Estimating	3	25.0
Evaluating	4	81.3
Choosing	5	81.3
Planning	6	25.0
Proof-of-Concept	6.5	31.3
Updating Requirements	7	43.8
Implementing	7	68.8
Reviewing	8	43.8
Testing	9	31.3
Updating Implementation	9	18.8
Deploying	10.5	12.5



Example Participant Data

Use this when: Using less common features in libraries instead of using the popular functions **- Providing Context**

Tools/technologies: StackOverflow, Google, continuous learning **- Defining Requirements**

Prior knowledge: Common design patterns, popular libraries **- Defining Requirements**

1) Decide what the goal of the program is. - **Defining Requirements**

2) Begin writing the program. **- Implementing**

3) While writing the program, search online whether other libraries support your use case... - **Researching**

4) Choose a library which meets your use case.... - **Choosing**

5) Look at the features of the library and test the ones that you're interested in on small examples. Get a feel of the library and select a solution which achieves the desired behavior. - **Updating Implementation**

6) If you have code that works, show the solution to another individual for review. - **Review**

RQ4: Developer Expertise (17 total)

Decision Making	**Count**
Knowledgeable about customers and business	56
Sees the forest and the trees	45
Knowledgeable about tools and building materials	39
Knowledgeable about their technical domain	38
Software & Designs	**Count**
Carefully constructed	26
Fitted	11
Evolving	10
Attentive to details	9

Discussion & Future Work

EDUCATORS

SOFTWARE ENGINEERS

RESEARCHERS

Best practices

- HCI Guidelines for Gender Equity and Inclusivity
- Experiments with human subjects in software engineering
- Open coding

Class Activity

Next class

- **Discussion Presentations**
  Implementation and Maintenance [03/29]