Planning and Development Workflow

Category	Workshop
Content Type	Workshop
Status	Not Started
Workshop Day	Day 1

Workshop Overview

Purpose:

Introduce participants to the workflow they will use throughout the hackathon, beginning with understanding application requirements and planning how the system should function before moving into design and development.

Participants will learn how to break down requirements, visualize application interactions with Excalidraw, create an initial interface design with Variant, set up their development environment, establish version control with GitHub, and begin using Codex and Claude Code as AI engineering assistants.

The goal is not to complete an application during this workshop. The goal is to establish a repeatable development process participants can independently apply during the competition.

Duration: 60 to 120 minutes

Instructor(s): @Shoug Alomran

Workshop Day: Day 1

Learning Objectives

By the end of this workshop, participants should be able to:

- Break application requirements into users, pages, features, actions, and data
- Create a functional application diagram using Excalidraw
- Explain how the major parts of their application interact

- Translate a functional plan into an initial UI/UX design using Variant
- Set up and navigate a project using Visual Studio Code
- Understand the basic role of Node.js and npm in the development environment
- Create and connect a GitHub repository
- Use Git for basic version control
- Use Codex and Claude Code as AI engineering assistants
- Write clear prompts that provide AI with useful context and requirements
- Review, run, test, and debug AI-generated changes rather than assuming they are correct
- Follow a structured development workflow from planning through implementation

---

Prerequisites

Participants should have:

- A laptop capable of running the required development tools
- Basic familiarity with websites and web applications
- Basic familiarity with files and folders
- Access to a modern web browser
- Required software installed before the workshop when possible

Prior experience building a complete web application is not required.

Instructor Preparation

What needs to be prepared before delivering this workshop?

- [ ] Prepare a small example application requirement for the workshop
- [ ] Create an example Excalidraw functional diagram
- [ ] Prepare an example Variant design
- [ ] Prepare a small starter project or repository for demonstration
- [ ] Verify that the starter project runs correctly

- [ ] Test Codex and Claude Code with the demonstration project
- [ ] Prepare at least one example of debugging an AI-generated implementation
- [ ] Verify all required tools and accounts before the workshop
- [ ] Prepare participant setup instructions
- [ ] Prepare backup screenshots or examples in case a live service is unavailable

Required Accounts

- GitHub
- Claude
- OpenAI / Codex
- Variant (optional for design)

Required Software

- Visual Studio Code
- Git
- Node.js
- npm
- Modern web browser
- Claude Code
- Codex

Required Files / Repositories

- Workshop starter repository
- Example application requirements
- Example Excalidraw diagram
- Example Variant design
- Participant setup guide

Workshop Outline

Time	Section	Format	Notes
0 to 10 min	Development Workflow	Teach	Introduce the overall process participants will use
10 to 25 min	Requirements and Excalidraw	Teach / Demo / Exercise	Break requirements into application interactions
25 to 40 min	Variant and UI Planning	Demo / Exercise	Translate functionality into interface design
40 to 55 min	VS Code, Node.js, and npm	Teach / Demo	Establish the local development environment
55 to 70 min	Git and GitHub	Teach / Demo / Exercise	Create the repository and establish version control
70 to 90 min	Codex and Claude Code	Teach / Demo / Exercise	Begin AI-assisted implementation
90 to 120 min	Practice and Instructor Support	Exercise	Participants work while instructors provide individual assistance

If the workshop is limited to 60 minutes, reduce demonstrations and use the remaining material as guided practice or supplemental material.

Concepts to Teach

Concept 1: Think Before You Build

What participants need to understand:

AI can generate code quickly, but participants still need to understand what they are trying to build.

Before implementation begins, they should be able to explain the application's users, functionality, pages, interactions, and data.

Key points:

- Start with requirements, not code
- Identify what the user needs to accomplish
- Break large requirements into smaller features
- Identify relationships between different parts of the application
- Clarify unclear requirements before implementation

- Planning gives both the developer and AI better context

Concept 2: Functional Planning with Excalidraw

What participants need to understand:

Excalidraw is being used to visualize how the application functions. The objective is not to create formal enterprise architecture.

The diagram should help participants answer:

What happens when the user does something?

Key points:

- Identify users
- Identify major pages or application areas
- Identify important actions
- Identify where data is needed
- Show how major components interact
- Show important user flows
- Keep diagrams understandable rather than unnecessarily complex

Example mental model:

User → Interface → Action → Application Logic → Data → Result

Concept 3: UI/UX Planning with Variant

What participants need to understand:

The functional diagram describes how the application works. Variant helps participants explore how that functionality should be presented to the user.

Key points:

- Functionality should influence interface design
- Consider navigation and page hierarchy
- Consider forms, buttons, feedback, and important actions
- AI-generated designs are starting points
- Participants should evaluate and modify generated designs
- Visual consistency matters

Concept 4: Development Environment

What participants need to understand:

Participants should understand the basic environment in which their application is being developed rather than treating it as a collection of mysterious commands.

Key points:

- Visual Studio Code is the development environment
- The project folder contains the application
- The terminal allows commands to be executed
- Node.js provides the JavaScript runtime used by the development tooling
- npm manages project packages and dependencies
- The development server allows the application to run locally
- Browser Developer Tools provide information when something goes wrong

Concept 5: Version Control

What participants need to understand:

Git and GitHub provide a history of the project and allow participants to safely manage changes during development.

Key points:

- Repository
- Commit
- Push
- Pull
- Version history
- Commit meaningful checkpoints
- Avoid making large amounts of untracked AI-generated changes
- GitHub also connects the project to later deployment workflows

Concept 6: AI-Assisted Engineering

What participants need to understand:

Codex and Claude Code are engineering assistants. They can implement, explain, investigate, review, refactor, and debug code, but participants remain responsible for evaluating the result.

Participants should follow:

Describe → Generate or Modify → Review → Run → Test → Debug → Verify

Key points:

- Give AI context
- Describe the objective
- Explain expected behavior
- Include relevant constraints
- Break large tasks into smaller tasks when necessary
- Read what the AI changed
- Test the result
- Provide errors and observations when debugging
- Do not assume generated code is correct

Live Demonstration

Objective

Demonstrate the complete workflow on a small example feature, beginning with a requirement and ending with a working implementation committed to GitHub.

The demonstration should focus on the process rather than the complexity of the feature.

Steps

1. Present a small application requirement and identify the user, required behavior, pages, actions, and data.
2. Create a simple Excalidraw diagram showing how the feature should function.
3. Use the functional plan to create or refine an interface concept in Variant.

4. Open the project in Visual Studio Code and explain the basic project structure.
5. Create or connect the GitHub repository and establish an initial version-control checkpoint.
6. Give Codex or Claude Code a clearly defined implementation task.
7. Review the changes produced by the AI.
8. Run the application locally.
9. Test the implemented behavior.
10. If a problem occurs, demonstrate the debugging process instead of immediately replacing the implementation.
11. Verify the final behavior.
12. Commit the working change to GitHub.

Expected Result

Participants should see the complete relationship between planning and implementation:

Requirement → Functional Diagram → UI Design → Project → AI-Assisted Implementation → Testing → Debugging → Git Commit

Participants should understand that AI is one component of the engineering workflow rather than the entire workflow.

Hands-On Exercise

Objective

Participants will apply the demonstrated workflow to a small practice application or feature.

The exercise should require participants to make decisions rather than simply copy the instructor.

Instructions

1. Read the provided application requirements.
2. Identify the application's users, pages, features, actions, and required data.

3. Create a functional diagram in Excalidraw.
4. Create or refine an interface concept using Variant.
5. Open or create the project in Visual Studio Code.
6. Verify that the project runs locally.
7. Create or connect the project to a GitHub repository.
8. Create an initial commit.
9. Select one small feature to implement.
10. Use Codex or Claude Code to assist with the implementation.
11. Review the generated changes.
12. Run and test the application.
13. Investigate and debug any problems.
14. Verify the feature works as expected.
15. Commit the working implementation.

Success Criteria

Participants have successfully completed the exercise when:

- [ ] They can explain what their application or feature is supposed to do
- [ ] They have a functional Excalidraw diagram
- [ ] They have an initial interface design
- [ ] Their project runs locally
- [ ] Their project is connected to GitHub
- [ ] They have used Codex or Claude Code to implement or modify a feature
- [ ] They have tested the result
- [ ] They have committed a working checkpoint to GitHub

AI Prompts

Store prompts that will be demonstrated or given to participants here.

Prompt 1: Understand Before Implementing

> I am working on the following requirement:
>
> [REQUIREMENT]
>
> Before writing any code, help me break this requirement down into the users involved, required functionality, pages or components, user actions, data requirements, and important interactions. Identify anything that is unclear or that I should decide before implementation. Do not implement anything yet.

Purpose:

Demonstrate that AI can assist with reasoning and planning rather than only generating code.

Prompt 2: Implement a Defined Task

> I want to implement the following feature:
>
> [FEATURE]
>
> Expected behavior:
>
> [EXPECTED BEHAVIOR]
>
> Relevant constraints:
>
> [CONSTRAINTS]
>
> First inspect the existing project and determine which files are relevant. Explain your implementation approach, then make the required changes. Do not modify unrelated functionality.

Purpose:

Teach participants to provide objective, context, expected behavior, and constraints rather than using vague prompts such as "build this for me."

Prompt 3: Debug a Problem

> I expected:
>
> [EXPECTED BEHAVIOR]
>
> Instead, I observed:
>
> [ACTUAL BEHAVIOR]
>
> Error message:
>
> [ERROR]

> Investigate the likely cause before changing the code. Explain what evidence supports your diagnosis, then propose the smallest appropriate fix. After making the change, tell me how I should verify that the problem is resolved.

Purpose:

Encourage investigation and verification instead of repeatedly asking AI to "fix it."

Common Problems

Problem 1: Participant immediately asks AI to build the entire application

Symptoms:

The participant provides a broad prompt and allows the AI to generate a large amount of code without understanding the implementation.

Likely Cause:

The participant is treating the AI assistant as an application generator rather than an engineering assistant.

How to Investigate:

Ask the participant to explain:

- What feature they are currently implementing
- What the expected behavior is
- Which part of the application is responsible
- How they will know whether the implementation works

Instructor Hint:

Ask them to reduce the problem to one clearly defined task.

Solution:

Return to the functional plan, select one feature, define its expected behavior, and provide the AI with a smaller implementation task.

Problem 2: Application does not run locally

Symptoms:

The development server fails, the page does not load, or terminal errors appear.

Likely Cause:

Possible causes include missing dependencies, incorrect commands, incorrect directory, configuration problems, or code errors.

How to Investigate:

Read the terminal output first. Verify the current directory, dependencies, project scripts, and error message.

Instructor Hint:

Ask:

"What information is the terminal giving us?"

Solution:

Determine the cause from the available evidence before changing code.

Problem 3: AI-generated feature does not behave correctly

Symptoms:

The code runs, but the feature behaves differently from the requirement.

Likely Cause:

The prompt may have been ambiguous, the AI may have misunderstood the existing architecture, or the implementation may contain a logic error.

How to Investigate:

Compare:

Expected behavior → Actual behavior → Generated implementation

Instructor Hint:

Ask the participant to explain exactly what is different between what they expected and what they observed.

Solution:

Provide the AI with the expected behavior, actual behavior, relevant code/context, and observed evidence.

Instructor Notes

Notes that are useful for whoever is delivering or assisting with the workshop.

- Keep lectures short and move quickly into practical work
- Do not solve every problem for participants
- Ask participants to explain what they think is happening
- Encourage participants to read errors before prompting AI
- Encourage small, testable implementation steps
- Use unexpected errors during demonstrations as teaching opportunities when practical
- Emphasize that planning does not need to be perfect before development begins
- Avoid turning Excalidraw into a formal architecture exercise
- Avoid turning Variant into a design competition
- Focus on the relationship between planning, implementation, testing, and iteration
- Leave sufficient time to walk around and assist participants individually

---

Participant Questions

Record useful questions that come up during practice sessions or the actual workshop.

- TBD during workshop

Resources

Documentation

- Excalidraw documentation
- Variant documentation
- Visual Studio Code documentation
- Git documentation
- GitHub documentation
- Node.js documentation

- npm documentation
- Codex documentation
- Claude Code documentation

References

- Workshop slides
- Participant setup guide
- Development workflow cheat sheet
- Git and GitHub cheat sheet
- AI prompting examples

Examples

- Example application requirements
- Example functional diagram
- Example Variant design
- Example repository
- Example AI prompts

Workshop Materials

Slides/Html

Create a short presentation covering:

- Development workflow
- Requirements and planning
- Functional diagrams
- UI planning
- Development environment
- Git and GitHub
- AI-assisted engineering
- Testing and debugging

Handout / Cheat Sheet

Create a one-page development workflow reference containing:

Requirements → Plan → Design → Set Up → Implement → Review → Run → Test → Debug → Verify → Commit

Include common Git commands and example AI prompts.

Repository

TBD

The repository should contain the small practice application used during the workshop.

Additional Files

- Participant setup guide
- Practice application requirements
- Example Excalidraw diagram
- Example Variant output
- AI prompt cheat sheet

End-of-Workshop Check

Participants should be able to demonstrate:

- [ ] A functional application diagram
- [ ] An initial interface design
- [ ] A locally running project
- [ ] A GitHub repository containing their project

- [ ] At least one meaningful Git commit
- [ ] At least one feature implemented or modified with Codex or Claude Code
- [ ] Evidence that they tested the implementation
- [ ] An understanding of where to begin investigating when something does not work

---

Improvements for Next Time

Complete this section after delivering the workshop.

What worked well:

TBD

What caused confusion:

TBD

What should be changed:

TBD

What took longer than expected:

TBD

Participant feedback:

TBD

Improvements for Next Time

Complete this section after delivering the workshop.

What worked well:

What caused confusion:

What should be changed:

What took longer than expected:

Participant feedback: