# AI-Powered Development Workflow

**Piseth Chhoeuy, Jun Kim, Matthew Tinnel, Eileen Yee**

College of Engineering, Electrical Engineering and Computer Science, Oregon State University

CS467: Online Capstone Project

Instructors: Samarendra Hedaoo, William Pfeil

Date: November 28, 2023

## Introduction

The rise of AI-powered tools in software development promises to revolutionize the way developers create, test, and deploy their applications. This document presents our findings from experimenting with various AI-assisted development workflows while developing a software development project. The aim is to identify the most efficient practices that can be generalized and adopted for a range of development tasks.

## Research Goals

Our research aimed to explore the practical applications and limitations of AI tools in software development. We focused on assessing how AI can optimize the development workflow, improve code quality, and enhance overall productivity. By comparing traditional methods with AI-assisted approaches, we sought to uncover insights that could shape future software development practices.

## Methodology

To accurately gauge the effectiveness of AI in programming tasks, we undertook a series of experiments involving three distinct approaches for each task:

> **Traditional Coding**: Manually writing code without AI assistance.
> **AI-Pair Programming**: Using AI tools like GitHub Copilot for real-time code suggestions while writing code.
> **AI-Led Development**: Starting with a high-level description and relying on AI to generate the first draft of code, which is then refined by a developer.

Efficiency was measured by the time taken to complete tasks, the quality of code produced, and the ease of integration into existing workflows. Quality was judged by code readability, maintainability, and the absence of bugs.

## Findings

### Project and Task Organization

- **Task Granularity**: AI tools were more effective when tasks were broken down into smaller, more granular problems.
- **Clear Definitions**: AI-generated better results when provided with clear, concise, and detailed descriptions of the desired outcome.

### Team Organization

- **AI as a Team Member**: Treating AI as a junior developer, best for generating code from detailed descriptions and for initial drafts.
- **Review Process**: Code generated by AI should go through the same review process as code written by human developers.

### AI Tool Application

- **Initial Development**: AI can rapidly create boilerplate code and implement standard patterns.
- **Refactoring**: AI tools are efficient in suggesting refactoring opportunities for better code quality.
- **Bug Fixes**: AI shows promise in identifying and suggesting fixes for common bugs.

### General Principles

- **AI Augmentation**: AI should be used to augment, not replace, human developers.
- **Iterative Refinement**: AI-generated code often requires iterative refinement to meet specific standards.
- **Continuous Learning**: Both AI models and human developers benefit from continuous feedback loops.

### Recommendations

- **Tool Integration**: Integrate AI tools directly into the development environment for best results.
- **Task Allocation**: Assign AI tools tasks that are well-defined and repetitive, reserving complex problem-solving for human developers.
- **Training and Onboarding**: Developers should be trained to effectively interact with and provide feedback to AI tools.

### Conclusion

AI-powered tools can significantly enhance development workflows when used appropriately. By assigning the right tasks to AI and maintaining a robust review process, teams can achieve a higher degree of efficiency while maintaining code quality. Our team anticipates that the rapid improvements of AI tools will quickly shift this landscape and allow AI to produce high quality products in very short periods of time.

For future research considerations, our team is interested in “assistant” type AI tools, which frequently have the ability to be assigned tasks and then self-prompt in order to complete them. Such products also have the ability to create documents and reference their own documents when completing future prompts. This is an immediate work-around to current AI tool memory limitations [1][2].

## Detailed Case Study: Implementing Express Backend With AI-Assistance

### Overview

This case study examines the development of an Express Backend system with user authentication, comparing traditional development methods with AI-assisted approaches. The task was to create an authentication feature that allows users to register, log in, and log out of a web application.

### Traditional Approach

The traditional development process involved the following steps:

**Planning**: Outlining requirements, creating flow diagrams, and establishing database schemas.
**Coding**: Writing all code manually, including user model, routes, controllers, and views.
**Testing**: Writing and running unit tests, integration tests, and conducting manual testing.
**Debugging**: Identifying bugs through user feedback and log analysis, then manually fixing them.

**Metrics**:

- **Time**: 20 hours to complete.
- **Lines of Code**: Approximately 1000 lines.
- **Bugs**: 4 bugs found during testing.
- **Code Quality**: High cohesion and low coupling with well-documented code.

## AI-Assisted Approach

The AI-assisted development process was divided into three experiments: AI-Pair Programming with GitHub Copilot, AI-Led Development with ChatGPT with specific prompting, and AI-Led Development with ChatGPT with general prompting.

## Experiment 1: AI-Pair Programming with GitHub Copilot

**Planning**: Same as the traditional approach.
**Coding**: Using GitHub Copilot to suggest code snippets and entire functions while coding.
**Testing**: Leveraging AI to suggest test cases and potential edge cases.
**Debugging**: Using AI to identify common patterns in bugs and suggest fixes.

**Metrics**:

- **Time**: 12 hours to complete.
- **Lines of Code**: Approximately 800 lines, due to more efficient code suggestions.
- **Bugs**: 7 bugs found, with some preemptively caught by AI.
- **Code Quality**: Similar to traditional, with some areas improved via AI suggestions.

## Experiment 2: AI-Led Development with ChatGPT4 with Specific Prompting

**Planning**: Outlining requirements in a detailed prompt for AI. The tech stack and all requirements were given to ChatGPT in prompts before asking it to begin coding.
**Coding**: ChatGPT asked questions to clarify requirements and made suggestions. Suggestions made by ChatGPT were generally accepted. ChatGPT was then prompted to create the endpoints needed for the project.
**Testing**: Generating test cases based on ChatGPT suggestions.
**Debugging**: Debugging was mostly manual with AI providing insights on error messages.

**Metrics**:

- **Time**: 18 hours to complete due to additional time refining AI-generated code.
- **Lines of Code**: Approximately 1200 lines, including initial AI-generated code which was later pruned.

- **Bugs**: 12 bugs found, likely due to initial unfamiliarity with AI-generated code.
- **Code Quality**: Required significant refinement for readability and maintainability.

## Experiment 3: AI-Led Development with ChatGPT4 with General (Vague) Prompting

**Planning**: Giving AI a vague prompt and allowing it to come up with what it deemed necessary. The prompt to chatgpt4 was that this was a new project, the application intentions (i.e. job tracking that allowed users to track skills and network with contacts), we were planning to use a MERN tech stack and that I was tasked to work on the backend.
**Coding:** AI helped with initializing the new project which included creating new directories, installing necessary packages and setting up the connection from our machine to mongodb using express. Besides changing some file/router names, there were little to no bugs that I found. However, I did have to ask the AI to reorganize the folder structure to make it more modular (i.e. it combined the router and controller file into one). Once that was all set, I prompted the AI to generate one endpoint at a time and it did this without any errors. Although it only generated the get and post (create) operations, I had to prompt it at a later point to create the update and delete portion.
**Testing:** This was a really interesting part because on the initial endpoint request, I prompted the AI to generate a test case but after the first one, the AI actually asked if I wanted to generate a test case for each endpoint afterwards.
**Debugging:** I had only a couple of bugs which by providing the AI the error message, it was able to help me debug. One issue was because I had renamed a file without relaying that back to ChatGPT and subsequent code was calling/importing the incorrect function/file name; this was more of a human error.

**Metrics**:

- **Time**: 12-15 Hours to complete (mainly due to prompting in small bite size chunks)
- **Lines of Code**: About 1100 with test cases included (900 without)
- **Bugs**: 2 maybe 3 bugs found and easily corrected.
- **Code Quality**: Good - although I had to make small refinements/adjustments to make things more modular and easier to read.

## Analysis

AI-Pair Programming proved to be more efficient than the traditional method, reducing development time and resulting in fewer bugs. GitHub Copilot provided contextually relevant code that reduced the coding effort and preemptively addressed some potential issues. The code quality remained high, with improvements in areas influenced by AI suggestions.

AI-Led Development initially increased the time due to the need to understand and refactor the AI-generated code. However, it showed potential in rapidly creating a starting point for development, and with increased familiarity, it may lead to faster development cycles.

## Backend Case Study Conclusion

The integration of AI in the development workflow can lead to significant efficiency gains. AI-Pair Programming was notably effective, suggesting that AI can serve as a valuable tool in assisting experienced developers. AI-Led Development can be beneficial for quickly scaffolding projects, but it requires careful oversight and an understanding that AI-generated code will likely require modification.

This case study demonstrates that AI-assisted development can enhance the software development lifecycle, but it is essential to leverage AI tools in the right context and complement them with human expertise.

## Detailed Case Study: Implementing React Frontend With AI-Assistance

### Overview

This case study examines the development of an Express Frontend system for a web application to track users' internship/job hunting efforts. This study compares traditional development methods with AI-assisted approaches. The task was to create pages that users can interact with to register, log in, create jobs and skills, and maintain contacts.

### Traditional Approach

The traditional development process involved the following steps:

**Planning**: Outlining application requirements, creating flow diagrams and designing frontend.
**Coding**: Writing code manually, including frontend pages for user actions, html/css design and connecting frontend/backend.
**Testing**: Testing frontend to backend connection, onclick event testing, and other manual tests as needed.
**Debugging**: Identifying bugs through developer tools, searching the internet for fixes and manually fixing them.

**Metrics**:

- **Time**: 20 hours to complete.
- **Lines of Code**: Approximately 1000 lines of code.
- **Bugs**: 4 bugs found during testing.
- **Code Quality**: Easily linked and well-documented code.

### AI-Assisted Approach

The AI-assisted development process was divided into two experiments, AI development with Bard, AI-led Development with ChatGPT4 with specific prompting and AI-Led Development with ChatGPT4 with general prompting.

### Experiment 1: AI-Led Development with Bard

**Planning**: Same as traditional approach.

**Coding**: Prompting ChatGPT4 and Bard with the same prompt
**Testing**: Leveraging AI for suggestions
**Debugging**: Utilizing AI to determine the issue.

**Metrics**:

- **Time**: 15 hours
- **Lines of Code**: Approximately 400 lines of code
- **Bugs**: 2 bugs found
- **Code Quality**: Similar code quality, improved by AI suggestions, harder to integrate frontend and backend.

## Experiment 2: AI-Led Development with ChatGPT4 with General Prompting

**Planning**: This method involved providing ChatGPT-4 with broader, more general prompts. The focus was on having ChatGPT-4 outline the necessary components and structure of the project, rather than responding to specific task prompts.
**Coding**: ChatGPT-4's role expanded to suggesting a comprehensive structure and outline for the project. It offered a more holistic view of the development process, encompassing a wider range of functionalities and integrations. It would then provide me with the code snippets needed to make the overall project come together with a commented out section that described what should be replaced in there with my own code.
**Testing**: Each phase of the AI-suggested development was subjected to rigorous testing and debugging. However, given the broader nature of the prompts, this might involve more complex integration and functionality testing.
**Debugging**: The bug numbers themselves won't many but each time I fixed

**Metrics**:

- **Time**: 30 hours
- **Lines of Code**: Approximately 300 lines.
- **Bugs**: 7
- **Code Quality**: Poor - ChatGPT's code suggestions were fundamentally sound but lacked cohesion. The individual code snippets, while having good structure, were not functional and did not seamlessly integrate, resulting in an intensive debugging process

## Experiment 3: Developer-Led Development with ChatGPT4 with Specific Prompting

**Planning**: The questions asked to ChatGPT4 were very simple tasks that did not have much context behind it. They were prompts that had a specific goal and once the initial prompts were created, I would ask questions that would build upon the ideas and code generated as ChatGPT slowly learned about the overarching task.
**Coding**: ChatGPT gave me code snippets that I could use to add to the frontend server whenever I prompted it to give me code to complete a certain task. Then, I would feed that same code to ChatGPT and add another functionality to it. I would try to make the addition as easy as

possible so that I could have different versions of the code that I could always go back to if I decided I didn't like where the development was heading.
**Testing**: After each variation of a code snippet I ask ChatGPT, I tested each version of the code so that it wouldn't throw any errors before adding any new features. After a test successfully gave the version of a code snippet I was happy with, I would then use that code snippet as the new "version" to further test with after adding more features.
**Debugging**: If I saw an error while testing, I would write into the ChatGPT the code snippet that was relevant and the error code it threw at me. I would then ask it to help me resolve the error. It would be a cycle of testing and debugging and modifying prompts to make things simpler if it seemed to be stuck on an issue. .

**Metrics**:

- **Time**: 25 hours
- **Lines of Code**: Approximately 800 lines.
- **Bugs**: 18
- **Code Quality**: Good - Having very straightforward and simple tasks made the code snippets readable and optimized. There was an increased number of bugs, but each bug was much easier to fix and took much less time.

## Analysis

**Planning and Structure:** AI-led programming with general prompting significantly enhanced the ability to decompose complex tasks into manageable parts. This method leverages the broad understanding and pattern recognition capabilities of AI, allowing for a more holistic view of the project. AI's proficiency in identifying interdependencies and potential bottlenecks early in the development process enabled a more structured and efficient workflow.

**Efficiency and Control**: Developer-led programming with specific prompting emerged as the superior method in terms of efficiency and control. This approach allowed for a more nuanced understanding and management of the code, ensuring that each component was tailored to the specific needs of the project. By focusing on incremental steps and building outwards from a smaller core, the development process was streamlined. This method facilitated a clear progression, allowing for immediate adjustments and refinements, which is crucial in a dynamic development environment.

**Code Quality and Adaptability:** The code quality under the Developer-led method remained consistently high. The incremental nature of development meant that each new piece of code was a direct response to the preceding one, creating a tightly integrated and coherent system. AI suggestions were incorporated more effectively, as they were introduced in a controlled manner. This selective integration of AI insights led to tangible improvements without overwhelming the existing code structure.

**Testing and Debugging Efficiency:** The focused nature of specific prompting allowed for more efficient testing and debugging. Each code segment could be evaluated in the context of its immediate function and its place in the larger system. This made identifying and resolving issues quicker and more straightforward. With a clear understanding of each part of the code, the risk of introducing bugs during new feature implementation was significantly reduced. This approach ensured a stable and reliable development process.

## Frontend Case Study Conclusion

In summary, developer-led programming with specific prompting proved to be more effective for maintaining control over the development process, ensuring high code quality, and managing complexity. This method's incremental nature allowed for a more deliberate and thoughtful approach to coding, resulting in a more refined and optimized end product. It provided the necessary balance between innovation and stability, making it an ideal choice for complex software development projects.

The main advantage we found in using AI tools was as a learning aid and for kick starting a project. AI helps in coding, troubleshooting, and explaining concepts. But we noticed a downside: relying heavily on AI for coding hinders learning effective troubleshooting for issues. In addition, AI sometimes provided limited or abbreviated code, requiring frequent prompts for specific changes. We concluded that while AI is a powerful tool for development and learning, there are tasks where human skills and understanding are more efficient.

However, combining AI-led planning and structure with developer-led programming and specific prompting creates a synergistic approach. The AI-led method sets the stage with a strategic, well-structured plan and a comprehensive understanding of the project scope. The developer-led approach then takes over, focusing on the intricacies of coding, quality control, and adaptability to changes. This integration ensures that the project benefits from both high-level planning and meticulous execution, making it well-suited to handle the complexities of modern software development.

**References**

1. How assistants work - openai API - platform.openai.com. (n.d.). https://platform.openai.com/docs/assistants/how-it-works/agents
2. Significant-Gravitas. (n.d.). *AutoGPT.* GitHub. https://github.com/Significant-Gravitas/AutoGPT