**Analysis of PairD and GitHub Copilot Performance**

**Introduction**

This report provides an analysis of the performance and effectiveness of PairD and GitHub Copilot in building an ecommerce website with essential features such as user registration, authorization, product management, and browsing capabilities. The analysis covers various aspects, including document generation, code creation, unit test integration, and overall reliability of the generated code.

**Time Comparison**

**1. Requirements Document:**

Manual: approx. 8 hours

AI Generated: approx. 5 minutes

**2. Product Backlog:**

Manual: approx. 8 hours

AI Generated: approx. 5 minutes

**3. Architecture Diagram:**

Manual: approx. 3 hours

AI Generated: approx. 5 minutes

**4. Code Source:**

Manual: approx. 16 hours

AI Generated: approx. 8 hours

**5. Unit Tests:**

Manual: approx. 8 hours

AI Generated: approx. 1 hour

**Issues Identified with PairD**

**1. Response Discontinuity:**

PairD often stopped mid-response or failed to respond entirely, requiring the user to start new chats to continue. This interrupted the workflow and slowed down development.

**2. Unnecessary File Creation:**

PairD occasionally generated unnecessary files or folders that added complexity to the project structure, even when they weren’t required for the solution.

**3. Limited Code Generation for Larger Projects:**

PairD struggled to generate long responses for more complex projects. It required smaller, incremental requests to avoid breaking the tool, making it less feasible for large projects without careful handling.

**4. Suboptimal Solution in Certain Cases:**

Although PairD would provide a solution, it didn’t always use the best practices for problem-solving. However, it still managed to provide a working solution, even if it wasn't the most optimal.

**Positive Aspects of PairD:**

**1. Structured Document Generation:**

PairD excelled at generating structured documents such as requirements documents, product backlogs, and architecture designs. It effectively created well-organized plans to guide development.

**2. Ability to Retain Information:**

PairD effectively generated and managed the project structure, placing code in the correct files without having to remind it of the project structure it generated. It also generated code specific to the use case, requiring no manual modification.

**3. Code Generation from Scratch:**

PairD was able to generate code for the project from scratch with minimal bugs. This made it highly effective for the initial code creation phase, delivering a solid starting point for further development.

**Issues Identified with GitHub Copilot**

**1. Inability to Retain Information:**

GitHub Copilot couldn’t retain information across the entire conversation, even within the same chat session. This made it difficult to reuse previous context or details, leading to inconsistent code generation and incomplete solutions. Copilot would generate the project structure, but it couldn’t remember or apply the structure correctly in later parts of the conversation. This led to issues where Copilot would suggest placing code in files that didn’t exist in the project structure.

**2. Code Specificity:**

Copilot struggled to generate code that was specific to the user’s workspace. It required manual adjustments to make the generated code fit the specific use case, although it did mention what needed to be changed.

**3. Code Generation Blocked:**

Copilot’s code generation was blocked repeatedly due to its responses matching public code, which is restricted in the Deloitte’s settings. This prevented Copilot from generating usable code from scratch.

**4. Code Generation Quality:**

The code generated by Copilot, when not blocked, often exhibited poor quality and contained numerous bugs. It frequently used modules that did not exist or were incompatible with the project setup. As a result, using this code required significant time and effort to debug and fix, often making it more time-consuming than simply rewriting the entire code from scratch in a proper and reliable way.

**5. Debugging Inefficiencies:**

Copilot struggled with debugging code it had generated in the chat. Instead of fixing the issues, it often made the code worse. However, it was more effective when used in the workspace with code generated by PairD or written by the user.

**Positive Aspects of GitHub Copilot:**

**1. Efficient Debugging for Existing Code:**

Copilot was effective at debugging and fixing existing code, especially when working within the workspace where context was maintained. It helped in fixing CLI, dependencies and syntax issues quickly.

**2. Comprehensive Unit Test Generation:**

Copilot excelled at generating unit test cases that covered a wide range of scenarios. Although the generated tests contained bugs, it did well in providing the initial structure.

**3. Quick Code Suggestions for Specific Issues:**

Copilot was effective at generating code suggestions quickly for specific issues and provided helpful guidance on what changes needed to be made.

**Do's**

**- Use PairD for Structured Document and Code Generation:** Utilize PairD to generate structured documents such as requirements, product backlog, and architecture design, and for generating code incrementally in small steps.

**- Use PairD for Building the Initial Code Creation Phase:** Leverage PairD to establish the initial code creation phase, ensuring a strong and reliable foundation for the development process.

**- Use GitHub Copilot for Debugging:** Utilize GitHub Copilot to assist in debugging and fixing problems in existing code. It is effective for fixing TypeScript and syntax issues when working in the workspace.

**- Reiterate Important Information with GitHub Copilot:** If there’s information that needs to be remembered throughout the entire conversation, make sure to mention it in every prompt you send to GitHub Copilot, as it struggles to retain context over time.

**- Provide Specific Prompts:** Give detailed and specific prompts to both tools to ensure accurate results. They respond well when prompted for small, step-by-step tasks with waiting between each step for feedback from the user.

**Don'ts**

**- Rely on Copilot for Generating Code from Scratch:** Do not depend on GitHub Copilot for generating code from scratch, as it may be blocked, fail to remember context, or contains lots of bugs. Instead, generate the initial code using PairD and use Copilot for minor adjustments.

**- Trust Copilot’s Memory:** Do not trust Copilot to remember the project structure or details from previous parts of the conversation, as it often loses track of context and provides incorrect suggestions.

**- Ignore PairD’s Limitations:** Do not assume PairD can generate complex responses without breaking. Generate code in simple and small increments to ensure a more stable output.

**Conclusion**

While both PairD and GitHub Copilot offer valuable functionalities in the development process, they each have distinct strengths and limitations. PairD excels in structured document generation, incremental code creation, and generating code from scratch with minimal bugs, providing a solid foundation for further development. However, it requires careful handling to avoid response interruptions. GitHub Copilot, on the other hand, excels in debugging and fixing specific code issues but faces significant limitations in code generation due to context retention problems and organizational restrictions. For larger projects, PairD is more effective for the bulk of the work, with Copilot being useful for debugging and fixing specific issues.