AI-Powered Development: Promise and Perils

Executive Summary

This white paper presents the findings from an exercise to evaluate the productivity potential of AI-powered development tools. Using Windsurf by Codeium with Claude 3.5 Sonnet as the underlying large language model (LLM), we developed Scopic People - an internal tool designed to consolidate employee information scattered across various systems. The results demonstrated a remarkable 90% reduction in development time compared to traditional coding methods, producing high-quality code with minimal human intervention. This paper details the methodology, development process, challenges encountered, and lessons learned, providing valuable insights for organizations considering the adoption of AI-assisted development practices.

Contents

1. Introduction & Motivation

1.1 The Promise of AI-Powered Development

The software development landscape is undergoing a significant transformation with the emergence of AI-powered integrated development environments (IDEs). These sophisticated tools leverage large language models to automate coding tasks, potentially revolutionizing how software is created. By reducing the need for manual coding, these tools promise substantial productivity improvements, cost reductions, and accelerated time-to-market for software products.

1.2 Windsurf and AI-Assisted Development

Windsurf by Codeium represents the cutting edge of AI-powered development environments. Utilizing Claude 3.5 Sonnet as its underlying language model, Windsurf enables developers to create software through natural language instructions rather than traditional coding. This approach, sometimes referred to as "vibe coding," shifts the developer's role from writing code to effectively instructing the AI about desired functionality and reviewing its output.

1.3 Purpose of This White Paper

This white paper documents an experimental exercise conducted to evaluate the efficiency, effectiveness, and quality of code produced using Windsurf for developing an internal tool called "Scopic People." The exercise aimed to:

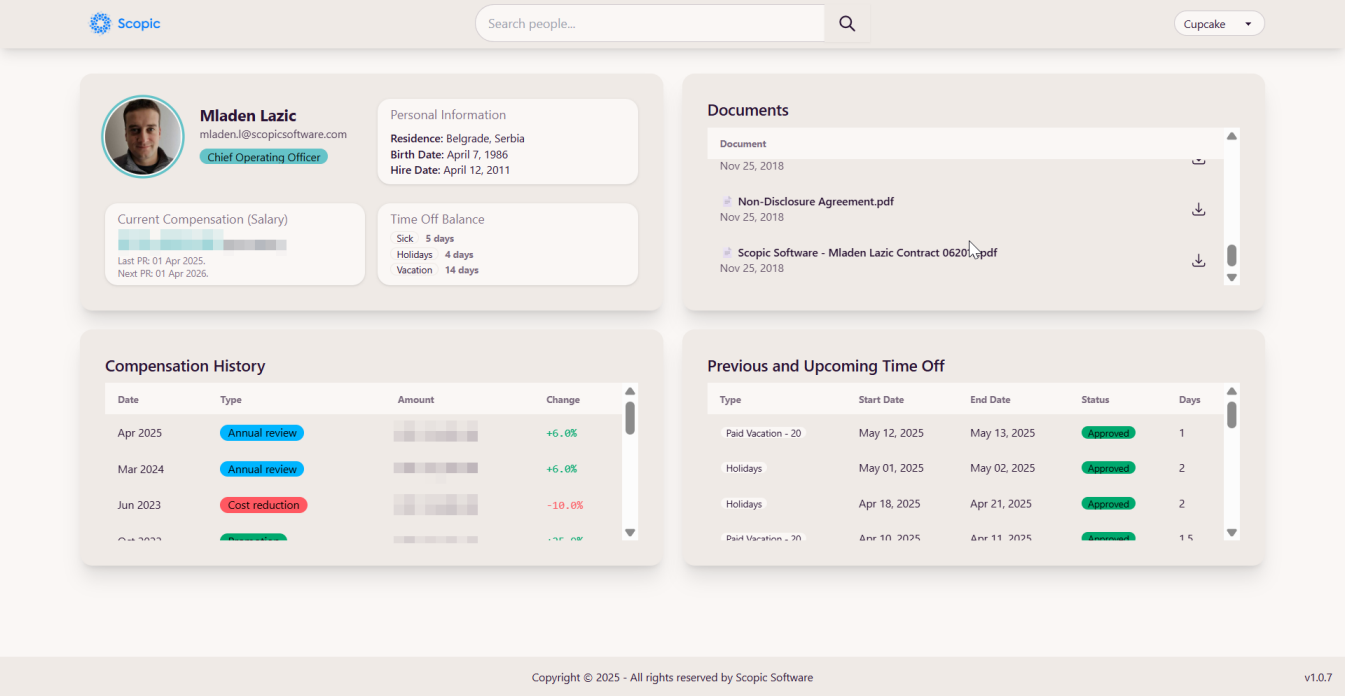
1. Assess the productivity improvements possible with AI-powered development tools
2. Evaluate the quality of code generated by an LLM
3. Identify best practices and potential pitfalls when using AI-assisted development
4. Formulate initial opinions about the future of AI in software development

By sharing the methodology, process, results, and lessons learned from this exercise, this paper provides valuable insights for organizations considering the adoption of AI-powered development tools.

2. Project Overview

2.1 The Scopic People Tool

Scopic People was conceived as an internal application designed to solve a common organizational challenge: accessing consolidated information about contractors that is typically scattered across multiple systems. The tool aims to create a centralized dashboard where users can view all relevant information about themselves or others (with appropriate permissions) in a single interface.



2.2 Key Requirements

Based on the requirements documentation, the Scopic People tool needed to implement the following specifications:

1. **Single Source Integration**: Initially, the tool would pull data exclusively from our main HR system, with architecture designed to accommodate additional data sources in future versions.
2. **Caching Mechanism**: To prevent overloading the HR system's API and stay within usage limits, a caching system would store retrieved data in a local database.
3. **Authentication**: The application would be secured behind Scopic's Keycloak Single

Sign-On (SSO) system, ensuring only authorized personnel could access the tool.

4. **Role-Based Access Control (RBAC)**: The system would implement two roles:
   - Regular users: Access limited to viewing their own profile
   - Management users: Access to search functionality and ability to view all contractor profiles
5. **User Interface**: The interface would be built using Tailwind CSS with DaisyUI components, following the Cupcake theme as shown in the provided mockups.

The long-term vision for Scopic People included extending data integration to additional sources and implementing proper group-based access management to replace the initial hardcoded access controls.

3. Methodology

3.1 Approach to AI-Assisted Development

The exercise began with clear parameters to evaluate AI-powered development:

1. **Starting Point**: A completely empty folder (greenfield project)
2. **Tool Selection**: Windsurf by Codeium with Claude 3.5 Sonnet as the underlying LLM
3. **Technology Choice**: Vanilla PHP without frameworks, ensuring the tool would have no head start from existing libraries
4. **Documentation**: The entire interaction with Windsurf was recorded for analysis

Prior to beginning the exercise, a senior engineer estimated that building the Scopic People tool through traditional development methods would require 80-100 man-hours for development, plus approximately 80% overhead for project management, technical leadership, quality assurance, and bug fixing—equating to roughly two weeks of total project time.

3.2 Development Workflow

The exercise utilized an iterative approach with the following workflow:

1. **Task Definition**: Break down the project into smaller, manageable tasks
2. **Instruction**: Provide Windsurf's Cascade feature with natural language instructions

3. **Code Generation**: Allow the LLM to generate the necessary code
4. **Review and Iteration**: Evaluate the generated code, accepting it or providing further instructions for improvement
5. **Version Control**: Commit accepted changes to Git to maintain a clear history

This process was repeated for each component of the application until the complete system was implemented and functioning according to requirements.

4. Tool Development Process

4.1 Initial Setup

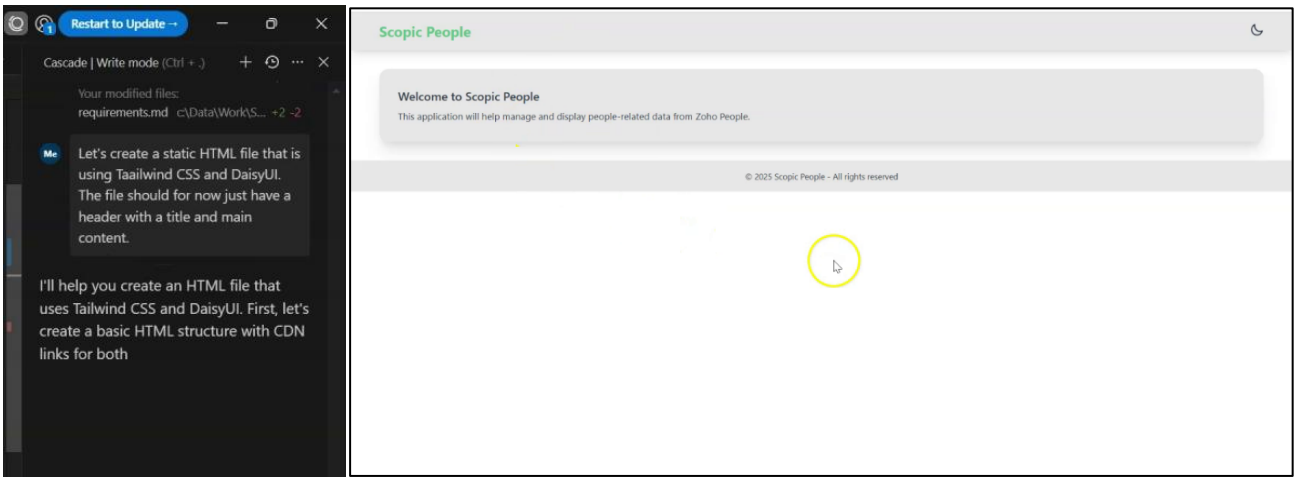
The development process began with initializing the project and setting up version control:

1. **Repository Initialization**: The LLM was instructed to create a new Git repository with appropriate README and .gitignore files for a PHP project. This step was crucial for establishing version control, enabling easy tracking of changes and the ability to revert modifications when necessary.

4.2 User Interface Development

The UI development process was broken down into smaller tasks after an initial attempt to generate the entire interface at once proved too complex for the LLM:

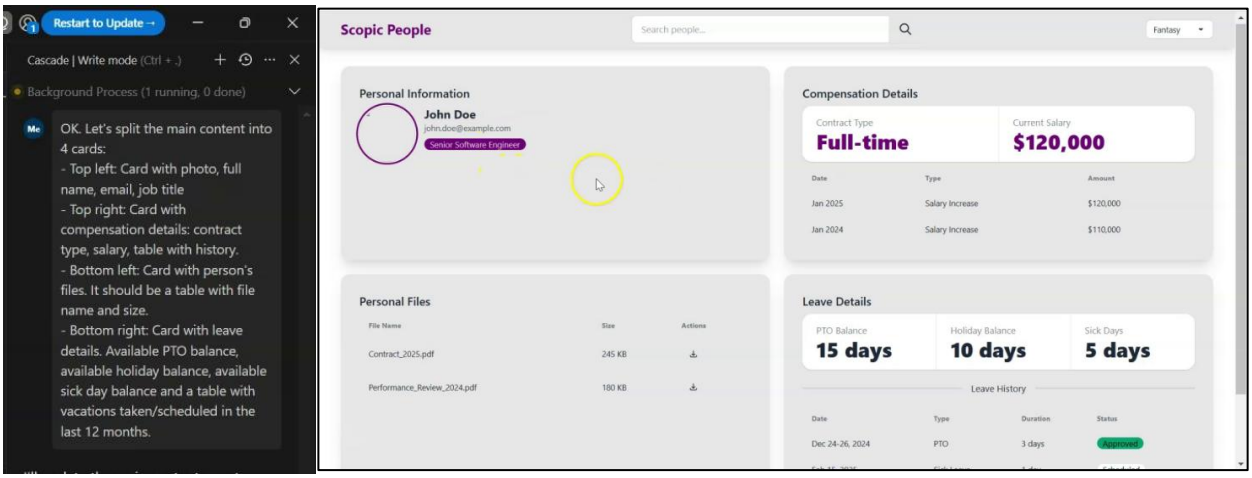
1. **Header Creation**: First, a page header was generated using Tailwind CSS and DaisyUI.



2. **Footer and Theme Selection**: A footer was added along with a theme selector for DaisyUI themes.

3. **Search Functionality**: A search bar was incorporated into the header. When the LLM produced a layout with the search bar and button on separate lines, it was instructed to fix this issue, which it accomplished correctly.

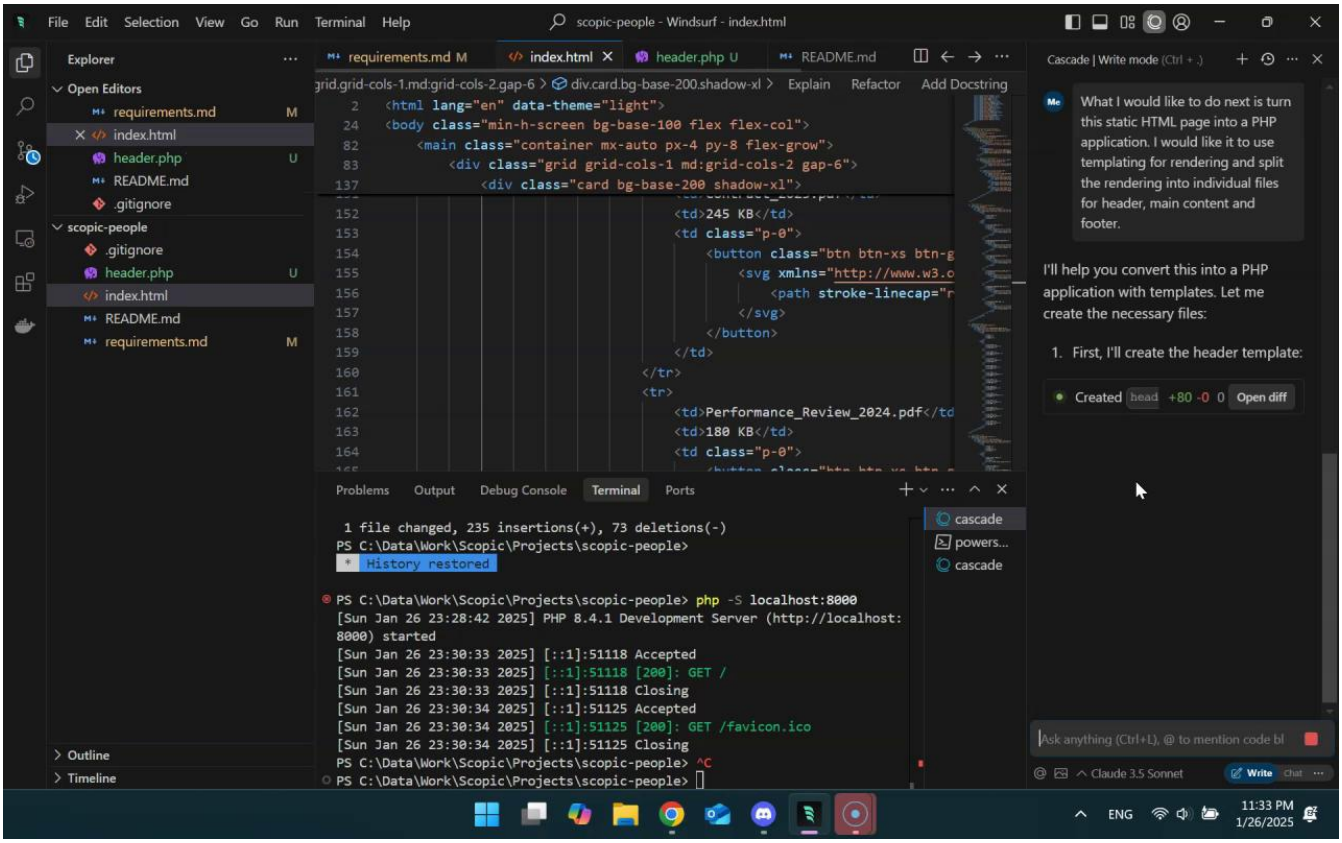
4. **Main Content Area**: The main content section required more detailed instructions and several iterations. The initial output showed promise but needed refinement.



During this phase, an interesting challenge emerged when Cascade began generating solutions in circular patterns. This required stopping the process, reverting changes, and restarting with modified instructions.

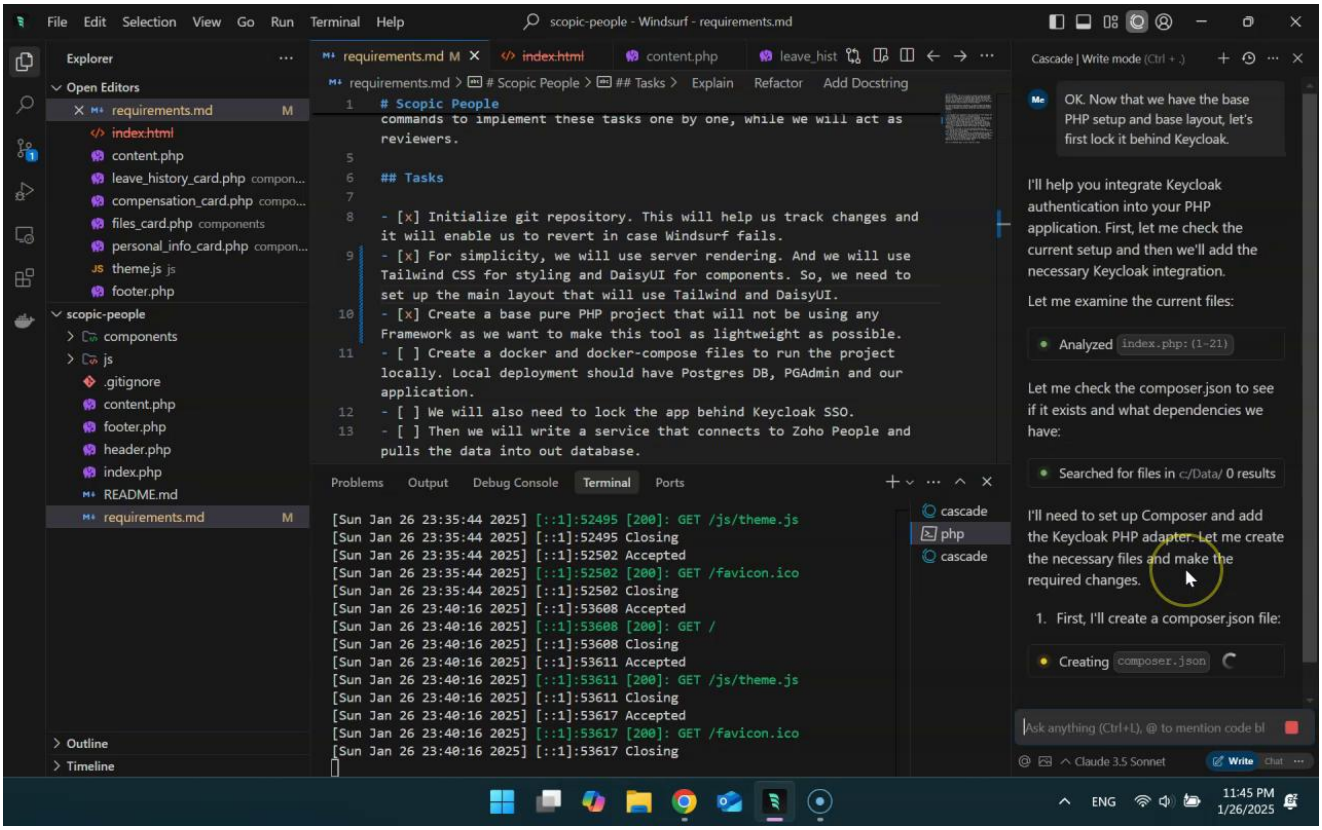
4.3 Converting to Dynamic Application

Once the static HTML interface was completed, the next step was transforming it into a dynamic PHP application. The static HTML page was converted into dynamic PHP templates, maintaining the same visual design while enabling server-side processing.



4.4 Authentication Implementation

Adding authentication was a critical security component.



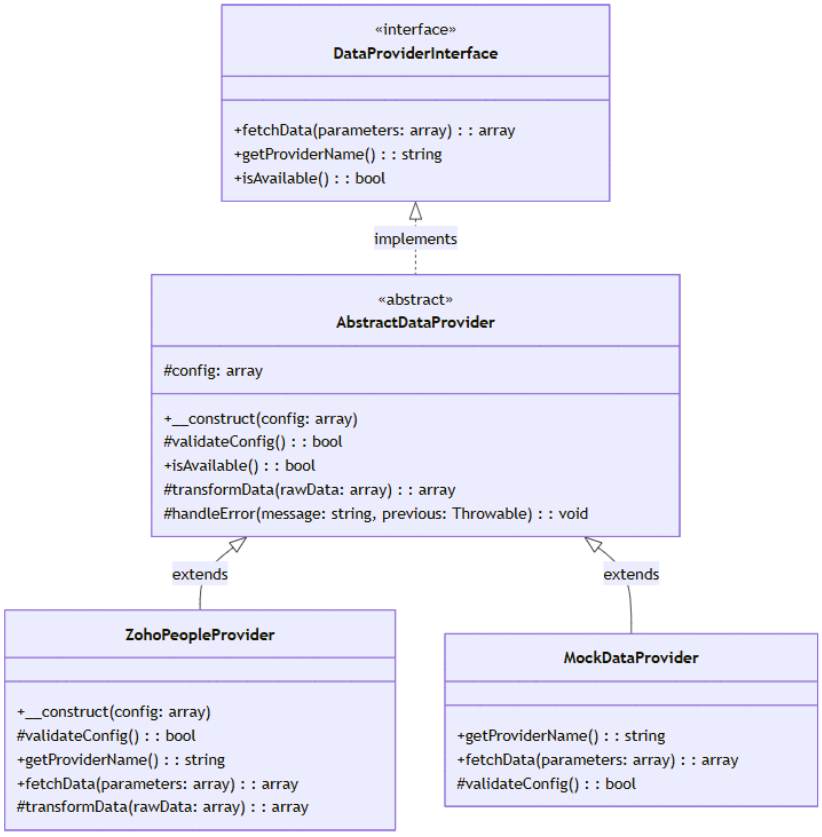
1. **Composer Integration**: The LLM added Composer to manage PHP dependencies required for SSO integration.
2. **Authentication Middleware**: An AuthMiddleware class was created to handle authentication requirements.
3. **Keycloak Integration**: The necessary callback endpoint for Keycloak was implemented, along with the OAuth flow configuration.

This step required more specific instructions due to the complexity of OAuth and SSO systems, demonstrating that the effectiveness of LLM-based development varies based on the task's complexity and domain-specific knowledge requirements. One important consideration to keep in mind is that 3rd party libraries must be reviewed manually to make sure that there are not security issues introduced by including outdated or unverified 3rd party libraries.

4.5 Data Model and Service Layer

To prepare for data integration, the application needed appropriate abstractions:

1. **Mock Data Model**: A temporary data model was created to populate the UI, facilitating easier replacement with actual data later. During this phase, Windsurf unexpectedly modified the UI, requiring a reversion and explicit instructions to maintain the existing interface.
2. **Service Architecture**: The LLM designed an extensible service architecture to support multiple data sources in the future.



3. **HR System Integration**: When instructed to implement a concrete data provider for our HR tool, the LLM attempted to search for API documentation. When this failed (likely due to inadequate public documentation), manual exploration of the API was required, after which the LLM successfully implemented the integration based on the provided endpoint information.

4.6 Containerization and Database Integration

To ensure easy deployment and data persistence:

1. **Docker Configuration**: The LLM generated Docker and docker-compose files to containerize the PHP application, PostgreSQL database, and pgAdmin for database management.
2. **Database Connection**: Database connection code was implemented with appropriate configuration parameters.
3. **Migration Scripts**: Database schema migration files were created to set up the tables needed for caching and role-based access control.
4. **Caching Implementation**: A caching service was implemented to store person data, limiting API calls to once per day per person.

4.7 Role-Based Access Control

The final major component implemented was access control:

1. **Access Restrictions**: The LLM added code to disable search functionality and prevent access to other users' profiles for non-admin users.
2. **Admin Management**: Rather than implementing a hardcoded list of administrative users as specified in the original requirements, the LLM created a database-driven approach for managing admin access. This solution exceeded the initial requirements by providing a more flexible and maintainable system for role management.

Upon completing these steps, the Scopic People application was fully functional and ready for testing, having been developed entirely through LLM-generated code with human guidance.

5. Results and Outcomes

5.1 Development Time Comparison

The results of this exercise were remarkably promising:

- **Traditional Development Estimate**: 80-100 hours of development time, plus approximately 80% overhead (144-180 total hours)

- **AI-Assisted Development Actual**: Approximately 9 hours to reach a fully functional application

This represents a 90% reduction in development time compared to traditional methods. When considering the reduced overhead due to faster development, the overall productivity boost is estimated between 75-80%.

5.2 Code Quality Assessment

The quality of the code produced by the LLM was evaluated as meeting acceptable standards. Key observations included:

- **Structure**: The code followed logical organization patterns and appropriate abstraction levels
- **Maintainability**: The code was readable and well-documented
- **Functionality**: All specified requirements were successfully implemented
- **Extensibility**: The architecture properly supported future additions as specified

5.3 Additional Validation

Following this successful experiment, two additional exercises were conducted:

1. An application with a different technology stack
2. Modifications to legacy code

Both exercises showed significant productivity improvements ranging from 40% to 80%, further validating the potential of AI-assisted development across various scenarios.

6. Challenges and Solutions

6.1 Task Complexity Management

Challenge: Initially, providing comprehensive instructions for complex UI components overwhelmed the LLM, resulting in incomplete or incorrect implementations.

Solution: Breaking down tasks into smaller, more manageable steps proved highly effective. Instead of requesting an entire interface at once, the process was divided into discrete components (header, footer, main content), each reviewed and refined before proceeding.

6.2 LLM Loop Prevention

Challenge: In certain scenarios, Cascade entered repetitive loops, making the same changes or undoing previous work without progress.

Solution: When loops were detected, the process was halted, changes were reverted to the last stable state, and instructions were reformulated with greater specificity. Additionally, frequent commits to version control proved invaluable for recovering from these situations.

6.3 Balancing AI and Manual Intervention

Challenge: Determining when to rely on the LLM versus making manual changes required judgment calls throughout the process.

Solution: Although this particular exercise focused on using the LLM for all code changes to evaluate its full capabilities, the findings clearly revealed that for minor adjustments or simple changes, direct manual editing would be more efficient than instructing the LLM. Relying on AI assistance for even small modifications can be counterproductive, consuming more time than necessary and potentially introducing unintended changes to other parts of the codebase. The exercise demonstrated that optimal productivity would likely result from a hybrid approach—using AI for boilerplate generation, complex logic, and architectural design while reserving simple tweaks for direct human intervention.

6.4 QA & Bug Fixing Process

Challenge: Determining how to approach bug fixing and whether to rely on LLM or solve bugs manually.

Solution: This problem sounds similar as the problem with balancing AI and manual intervention, but when it comes to QA our experiments have shown that starting with LLM by giving it a bug report can speed up the process of identifying where the problem is and in some cases even resolve the problem fully.

6.5 External API Integration

Challenge: The LLM struggled when attempting to integrate with the HR system via their API due to insufficient public documentation.

Solution: Human intervention to manually explore the API and provide specific endpoint information allowed the LLM to successfully complete the integration. This highlights the importance of accurate and complete information when instructing LLMs, particularly for third-party integrations.

7. Conclusion and Future Implications

7.1 Key Takeaways

This exercise demonstrated the transformative potential of AI-powered development tools. The key findings include:

1. **Significant Productivity Gains**: Significant overall improvement in productivity represents a paradigm shift in how software can be developed.
2. **Effective Practice Patterns**: The most effective approach involved breaking tasks into small steps, reviewing changes after each LLM iteration, committing frequently, and strategically balancing AI and manual coding.
3. **Quality Preservation**: The code produced with AI assistance maintained acceptable quality standards, suggesting that productivity gains need not come at the expense of code quality.
4. **Changing Developer Role**: The role of developers is evolving from writing code to effectively instructing AI and reviewing its output, emphasizing the importance of clear communication and review skills.

7.2 Best Practices for AI-Assisted Development

Based on the lessons learned, the following best practices are recommended:

1. **Task Granularity**: Break complex tasks into small, discrete steps rather than providing large, complex instructions.
2. **Continuous Review**: Review every change generated by the AI before proceeding to catch issues early.
3. **Version Control Discipline**: Commit changes frequently to maintain a clear history and enable easy recovery from unsuccessful iterations.

4. **Strategic AI Usage**: Use AI for generating boilerplate, complex logic, and architectural patterns, while handling simple modifications manually.
5. **Clear Instructions**: Provide explicit, unambiguous instructions with necessary context, especially for domain-specific functionality.

7.3 Future of Scopic People

The Scopic People tool, developed through this exercise, provides a solid foundation for future enhancements:

1. **Multiple Data Sources**: Integration with additional systems beyond our HR tool to provide a more comprehensive view of contractor information.
2. **Dynamic Access Control**: Implementation of a proper group-based access management system to replace the initial hardcoded administrator list.
3. **Enhanced UI Features**: Additional visualizations and interactive features to improve the user experience.
4. **Automated Updates**: Scheduled data refreshing and notification systems for important changes.

7.4 Broader Implications

The success of this exercise suggests several important implications for the software development industry:

1. **Workforce Transformation**: As AI tools become more capable, the skills valued in software development will likely shift toward system design, requirements analysis, and effective AI instruction.
2. **Economic Impact**: The dramatic productivity improvements demonstrated could significantly reduce development costs and accelerate time-to-market for software products.
3. **Democratized Development**: AI-assisted development may lower the barrier to entry for software creation, enabling a broader range of individuals to build functional applications.
4. **Quality Considerations**: While the initial results are promising, long-term studies on

maintainability, performance, and security of AI-generated code will be essential.

In conclusion, the development of Scopic People using Windsurf demonstrates that AI-powered development tools have matured to a point where they can deliver substantial productivity gains without compromising code quality. As these tools continue to evolve, organizations that effectively integrate them into their development processes may gain significant competitive advantages through faster, more cost-effective software delivery.

For full disclosure, it is important to note that while this exercise showed promising results for a tool of moderate complexity, it remains inconclusive how these AI-powered development tools would perform when building substantially larger and more complex software systems. The scaling of this approach to enterprise-grade applications with numerous integrations, complex business logic, and stringent performance requirements has not been thoroughly evaluated. This limitation represents an important area for future research and would be a valuable topic for subsequent studies. Organizations should consider these constraints when determining how to incorporate AI-assisted development into their workflows.

This white paper was prepared based on an actual development exercise conducted in Q1 2025. The findings represent a point-in-time assessment of AI-powered development capabilities and should be considered alongside the rapidly evolving landscape of AI technologies.