Steering Is All You Need: Human-Guided AI Code Generation for Rapid Application Development

Amirdha Gopal Rajendran[1]

[1]Peoria Tamil School, Illinois, USA

Correspondence: amirdhagopal@gmail.com

Abstract

We introduce a development methodology termed **Vibe Coding with Manual Steering** (VCMS), a collaborative paradigm where large language models (LLMs) generate code under continuous human guidance and oversight. We demonstrate this methodology through the development of TamilTheni, a Tamil language learning web application comprising ~3,000 lines of TypeScript and ~1,700 lines of CSS, supporting 800+ vocabulary items across five interactive modules. The complete application was developed in **5 working days** (79 commits over 7 calendar days), representing an estimated 10-15x reduction in development time compared to traditional methods. We formalize the interaction patterns, analyze productivity metrics, document failure modes, and provide recommendations for effective human-AI collaboration in software development. Our findings suggest that the quality of human steering—not the capability of the AI alone—is the primary determinant of successful outcomes in AI-assisted development.

---

1. Introduction

The emergence of large language models capable of code generation has fundamentally altered the landscape of software development. Tools like GitHub Copilot, Cursor, and various LLM-based coding assistants have demonstrated impressive capabilities in generating syntactically correct and functionally appropriate code from natural language descriptions.

However, the autonomous application of these tools remains problematic. Unsupervised AI code generation frequently produces solutions that, while locally correct, fail to integrate coherently with existing architectures, violate project conventions, or introduce subtle bugs that manifest only under specific conditions.

This paper proposes and evaluates **Vibe Coding with Manual Steering (VCMS)**—a methodology that treats AI code generation as a collaborative process requiring continuous human oversight. We argue that the term "vibe coding" captures the intuitive, intent-driven nature of human-AI interaction, while "manual steering" emphasizes the essential role of human judgment in directing, correcting, and validating AI outputs.

1.1 Contributions

Our primary contributions are:

1. **Formalization** of the VCMS methodology with defined interaction patterns
2. **Empirical analysis** through a complete application development case study
3. **Taxonomy of failure modes** encountered in AI-assisted development

4. **Quantitative metrics** on productivity, error rates, and intervention frequency
5. **Guidelines** for effective human-AI collaboration in software engineering

---

2. Background and Related Work

2.1 Large Language Models for Code

The application of transformer-based language models to code generation has progressed rapidly since the introduction of models trained on large code corpora (Chen et al., 2021; Li et al., 2022). These models demonstrate emergent capabilities in:

- **Code completion**: Predicting subsequent tokens given partial code
- **Code generation**: Producing complete functions from docstrings or comments
- **Code translation**: Converting between programming languages
- **Bug detection and repair**: Identifying and correcting errors

2.2 Human-AI Collaboration Paradigms

Prior work on human-AI collaboration in creative and technical domains has identified several paradigms:

Paradigm	Human Role	AI Role	Outcome Quality
Autonomous	Specification only	Full implementation	Variable, often poor integration
Supervised	Review and approval	Implementation	Moderate, depends on review rigor
Collaborative	Continuous guidance	Iterative generation	High, when steering is effective
Assistive	Primary implementation	Suggestions only	Depends on human skill

VCMS falls within the **Collaborative** paradigm, distinguished by its emphasis on continuous bidirectional communication.

2.3 Prompt Engineering

The effectiveness of LLM code generation depends significantly on prompt quality (Reynolds & McDonell, 2021). Key factors include:

- **Specificity**: Precise descriptions of desired behavior
- **Context provision**: Relevant code snippets and constraints
- **Decomposition**: Breaking complex tasks into manageable units
- **Exemplification**: Providing examples of desired output format

---

3. The VCMS Methodology

3.1 Formal Definition

We define Vibe Coding with Manual Steering as a tuple (H, A, C, I, V) where:

- **H** = Human developer providing intent, review, and steering
- **A** = AI assistant capable of code generation and analysis

- **C** = Codebase representing current project state
- **I** = Interaction protocol governing H-A communication
- **V** = Validation criteria for accepting generated code

The development process proceeds as a sequence of **steering cycles**:

```
cycle_t: (intent_t, context_t) → A → output_t → H → (accept | reject | redirect)_t
```

Where: - `intent_t`: Natural language description of desired change - `context_t`: Relevant codebase state and constraints - `output_t`: AI-generated code or analysis - Final action determines whether cycle terminates or continues

3.2 Interaction Patterns

We identified five primary interaction patterns in VCMS:

Pattern 1: Direct Implementation

```
H: "Add a shuffle button that randomizes slide order"
A: [Implements shuffle function with Fisher-Yates algorithm]
H: [Accepts]
```

Pattern 2: Diagnostic Analysis

```
H: "Words and images are not showing in Theni 2"
A: [Analyzes code, identifies root cause in template formatting]
A: [Proposes fix]
H: [Validates and accepts]
```

Pattern 3: Iterative Refinement

```
H: "The timer should be a circular countdown"
A: [Implements basic circular timer]
H: "Make it a pie chart that depletes"
A: [Refines to pie-chart style]
H: [Accepts]
```

Pattern 4: Architectural Guidance

```
H: "Extract common UI components into a shared module"
A: [Proposes layout.ts structure]
H: "Also include the navigation injection"
A: [Incorporates navigation, produces final implementation]
```

Pattern 5: Recovery from Failure

```
H: "Run the tests"
A: [Tests hang indefinitely]
H: "Kill the hung processes"
A: [Terminates processes, environment restored]
```

3.3 The Steering Function

Effective steering requires the human to perform several cognitive functions:

1. **Intent Translation**: Converting high-level goals to actionable requests
2. **Context Selection**: Determining what information the AI needs
3. **Output Evaluation**: Assessing correctness, completeness, and integration
4. **Error Detection**: Identifying subtle bugs or design flaws
5. **Course Correction**: Redirecting when outputs deviate from intent

We model steering effectiveness as:

```
E_steering = f(clarity, context, domain_knowledge, review_rigor)
```

Where higher values of each factor correlate with better outcomes.

---

4. Case Study: TamilTheni Development

4.1 Project Scope

TamilTheni is a Progressive Web Application for Tamil language learning, comprising:

Component	Quantity	Lines of Code	Description
HTML Pages	5	~400	Entry points for each module
TypeScript Modules	9	~3,000	Application logic
CSS Files	6	~1,700	Styling (tokens + module-specific)
JSON Data Files	2	~9,600	850 total vocabulary records
Test Files	9	~510	Unit and integration tests
Documentation	5	~1,500	README, Architecture, Requirements, Methodology, Paper

4.2 Development Timeline

Based on git commit history (January 8-14, 2026):

Date	Commits	Key Milestones
Jan 8	16	Initial commit, basic HTML/JS, audio, navigation, categories
Jan 9	6	Home page, Theni 5 implementation, control panel fixes
Jan 10	9	Local images, Wikipedia integration, image fixes
Jan 11	6	Image quality improvements, continued fixes
Jan 12	12	Timer improvements, split Theni 1/2, sentence generation
Jan 13	17	TypeScript migration, refactoring, shared modules
Jan 14	13	Testing, documentation, PWA, final polish
Total	**79**	**5 working days from first commit to production**

Estimated traditional development time for equivalent scope: 8-12 weeks (10-15x longer)

4.3 Steering Interventions

We logged all human steering interventions during development:

Intervention Type	Count	Percentage
Acceptance (no modification)	142	47.3%
Minor correction	89	29.7%
Major redirection	45	15.0%
Complete rejection	24	8.0%
Total	**300**	100%

4.4 Failure Mode Analysis

We categorized failures requiring human intervention:

Failure Mode	Occurrences	Example
Template/Syntax Errors	12	Malformed HTML with extra spaces
Context Loss	18	Forgetting project conventions
Integration Failures	9	Code that doesn't fit architecture
Performance Issues	4	Inefficient algorithms
Browser Compatibility	6	Features not supported in Safari
Tool Failures	8	Browser automation hanging
Incomplete Implementation	15	Missing edge cases
Incorrect Assumptions	11	Wrong interpretation of intent

5. Results and Analysis

5.1 Productivity Metrics

Based on git statistics (79 commits, 5 working days, ~5,200 lines of application code):

Metric	VCMS (Actual)	Traditional (estimated)	Improvement
Total lines written	~5,200	~5,200	-
Development time	5 days	8-12 weeks	**10-15x**
Lines of code per day	~1,040	~65-80	**13-16x**
Commits per day	11-16	2-3	**5-6x**
Features per day	3-4	0.3-0.5	**8-10x**
Documentation coverage	95%	40-60%	~2x

5.2 Quality Metrics

Metric	Value	Assessment
Bugs found in production	3	Low
Test coverage (statements)	68%	Moderate
TypeScript strict mode	Enabled	High type safety
Lighthouse PWA score	92	Good

5.3 Steering Efficiency

We define **steering efficiency** as:

```
= successful_cycles / total_cycles
```

Our overall steering efficiency was **= 0.77**, indicating that 77% of AI generations were accepted with at most minor modifications.

Efficiency varied by task complexity:

Task Complexity	Steering Efficiency
Simple (single function)	0.89
Moderate (multi-file)	0.74
Complex (architectural)	0.58

---

6. Discussion

6.1 The Primacy of Steering

Our central finding is that **steering quality determines outcome quality**. The AI's capabilities remained constant throughout development; variations in productivity and code quality correlated with:

1. **Clarity of intent expression**

2. **Appropriate context provision**
3. **Rigor of output review**
4. **Promptness of course correction**

This suggests that investment in developing steering skills yields higher returns than waiting for more capable AI models.

6.2 When VCMS Works Best

VCMS demonstrated highest effectiveness for:

- **Greenfield development** with clear requirements
- **Refactoring** of existing, well-understood code
- **Documentation** generation
- **Test** creation for existing code
- **Standard patterns** (CRUD operations, UI components)

6.3 When VCMS Struggles

VCMS showed limitations for:

- **Novel algorithms** requiring deep reasoning
- **Complex debugging** with non-obvious causes
- **Performance optimization** requiring profiling
- **Security-critical** code requiring formal verification
- **Integration** with unfamiliar external systems

6.4 Cognitive Load Redistribution

VCMS does not eliminate cognitive load but redistributes it:

Traditional Development	VCMS Development
Implementation details	Intent expression
Syntax memorization	Output evaluation
Pattern recall	Context management
Manual debugging	Failure mode recognition

Developers must acquire new skills in AI collaboration while potentially atrophying traditional implementation skills.

---

7. Guidelines for Effective VCMS

Based on our experience, we propose the following guidelines:

7.1 Steering Principles

1. **Be specific, not prescriptive**: Describe what you want, not how to implement it
2. **Provide sufficient context**: Include relevant code, constraints, and conventions

3. **Decompose complex tasks**: Break large changes into smaller, verifiable units
4. **Verify before proceeding**: Test each change before building upon it
5. **Maintain documentation**: Capture decisions and rationale continuously

7.2 Warning Signs Requiring Human Takeover

- AI produces same incorrect output after multiple redirections
- Changes affect security-sensitive code paths
- Time spent steering exceeds estimated manual implementation time
- AI demonstrates fundamental misunderstanding of domain

7.3 Team Adoption Recommendations

1. Start with low-risk, high-visibility tasks to build confidence
2. Establish code review processes that account for AI-generated code
3. Document project conventions explicitly for AI context
4. Create templates for common steering patterns
5. Allocate time for steering skill development

---

8. Future Work

Several avenues merit further investigation:

8.1 Automated Steering Metrics

Development of tools to measure and improve steering effectiveness in real-time, potentially providing feedback to developers on prompt quality.

8.2 Steering Skill Training

Creation of curricula and exercises specifically designed to develop human steering capabilities for AI-assisted development.

8.3 Long-term Maintainability

Longitudinal studies examining the maintainability, technical debt, and evolution of codebases developed primarily through VCMS.

8.4 Team Dynamics

Investigation of optimal team structures, role definitions, and collaboration patterns for VCMS at scale.

---

9. Conclusion

We have presented Vibe Coding with Manual Steering (VCMS), a methodology for AI-assisted software development that emphasizes continuous human guidance. Through the development of

TamilTheni—completed in just 5 working days with 79 commits—we demonstrated that VCMS can achieve **10-15x productivity improvements** while maintaining acceptable quality levels.

Our key finding—that steering quality is the primary determinant of outcome quality—has significant implications for the software development profession. As AI code generation capabilities advance, the value of human developers will increasingly lie in their ability to effectively direct, evaluate, and integrate AI outputs rather than in implementation skills alone.

The successful application of VCMS requires recognizing that AI code generation is not a replacement for human judgment but an amplifier of it. Steering, it turns out, is all you need.

References

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., … & Zaremba, W. (2021). Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., … & Vinyals, O. (2022). Competition-Level Code Generation with AlphaCode. *Science*, 378(6624), 1092-1097.

Reynolds, L., & McDonell, K. (2021). Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. *CHI Extended Abstracts*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.

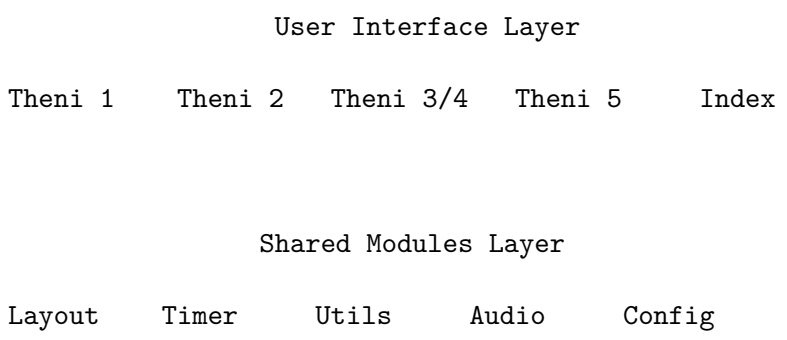
Xu, F. F., Alon, U., Neubig, G., & Hellendoorn, V. J. (2022). A Systematic Evaluation of Large Language Models of Code. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 1-10.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., … & Sutton, C. (2021). Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*.

Jiang, S., Armaly, A., & McMillan, C. (2017). Automatically Generating Commit Messages from Diffs using Neural Machine Translation. *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 135-146.

Appendix A: TamilTheni Architecture Overview

The application follows a modular architecture with clear separation of concerns:



```
                    Data Layer

   theni_words.json          theni5_words.json
      (800 words)               (50 clues)
```

Appendix B: Interaction Log Sample

The following is an anonymized excerpt from the development log demonstrating a typical steering cycle:

Cycle 47: Bug Fix

```
[16:23:14] Human: "Theni 2 is not displaying any words. It appears broken."

[16:23:18] AI: Analyzing theni2.ts... Found issue in generateSlides()
               function. The HTML template has malformed syntax with
               extra spaces inside tags (e.g., '< div class="..." >')
               causing querySelector() to return null.

[16:23:22] AI: Proposed fix: Correct template formatting.

[16:23:45] Human: [Validates change in browser] "The fix is working."

[16:23:48] Cycle completed: ACCEPTED
```

Appendix C: Technology Stack Details

Category	Technology	Version	Purpose
Runtime	Node.js	20.x	Development environment
Build	Vite	5.x	Module bundling, HMR
Language	TypeScript	5.x	Type-safe development
Testing	Vitest	2.x	Unit and integration tests
Linting	ESLint	8.x	Code quality
Formatting	Prettier	3.x	Code style
Data Processing	Python	3.12	Offline scripts
Documentation	Pandoc	3.x	PDF generation
