Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook – a Grey Literature Review

Ahmed Fawzy
Massey University
New Zealand
a.fawzy@massey.ac.nz

Amjed Tahir
Massey University
New Zealand
a.tahir@massey.ac.nz

Kelly Blincoe
The University of Auckland
New Zealand
k.blincoe@auckland.ac.nz

Abstract

Vibe coding is the practice where coders rely on AI code generation tools through intuition and trial-and-error without necessarily understanding the underlying code. Despite widespread adoption, there has been no systematic investigation into why vibe coders engage in vibe coding, what they experience while doing so, and how they approach quality assurance (QA) and perceive the quality of AI-generated code. To this end, we conduct a systematic grey literature review of 101 practitioner sources, extracting 518 firsthand behavioral accounts about vibe coding practices, challenges, and limitations. Our analysis reveals a speed–quality trade-off paradox, where vibe coders report being motivated by speed and accessibility, often experiencing rapid "instant success and flow", yet most perceive the resulting code as fast but flawed. Our analysis suggests that QA practices are frequently overlooked, with many reporting skipping testing, relying on the models' or tools' outputs without modification, or delegating checks back to the AI code generation tools. While vibe coding lowers barriers and accelerates prototyping, it can introduce reliability and maintainability concerns. These insights carry implications for tool designers and software development teams. Understanding how vibe coding is practiced today is crucial for guiding its responsible use and preventing a broader QA crisis in AI-assisted development.

Keywords

Vibe coding, AI-assisted programming, AI-generated code

ACM Reference Format:
Ahmed Fawzy, Amjed Tahir, and Kelly Blincoe. 2026. Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook – a Grey Literature Review. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-SEIP '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3786583.3786866

1 Introduction

Recent progress in large language models (LLMs), accessible through AI code generation tools, such as GitHub Copilot and ChatGPT, is rapidly transforming software development. These tools enable developers to describe functionality in natural language and receive executable code, thereby speeding up routine work and lowering the barrier to entry for individuals with limited programming experience [15, 27, 30]. With these tools, even individuals without formal training can increasingly develop functional applications [11]. This change represents a broader shift in developer roles, which now involve orchestrating, supervising, and integrating rather than writing every line of code [25, 32]. However, while these tools are transforming how software is created, less is known about the new coding practices emerging from their everyday use.

Amid the rapid adoption of AI code generation tools, a new practice known as *vibe coding* has emerged. Coined by Karpathy in 2025 [18], vibe coding is a new programming approach in which vibe coders employ AI code generation tools to write code by describing their desired outcome (in natural language) without fully understanding the AI-generated code. For example, a recent report noted that 25% of Y Combinator's Winter 2025 startups had codebases written almost entirely by AI code generation tools, illustrating how quickly this practice is spreading [24]. In contrast to AI-assisted programming, vibe coding prioritizes speed and experimentation over understanding.

In this paper, we define vibe coding as the practice of using AI code generation tools to produce software primarily by describing goals in natural language and iteratively prompting, while relying on minimal review of the generated code. The definition is derived from Karpathy's original introduction of the term [18] and further grounded in how practitioners themselves describe the practice in grey literature (e.g., [80]). This definition applies across vibe coder groups, including both professional developers (who may use vibe coding to prototype quickly) and non-software developers (who may attempt to build applications without coding knowledge) [9, 38]. Throughout this paper, we use the term "vibe coder" to refer to anyone who engages in vibe coding, regardless of prior software engineering experience.

Vibe coding has introduced advantages such as accessibility, speed, and creative potential. For example, non-software developers can now build working applications, and professionals can explore ideas faster than before [28, 29, 41]. However, prior research and industry commentary warn of severe limitations. Educational studies show that students often adopt AI-generated code without understanding it, risking poor learning outcomes [15, 30]. Security research has demonstrated that AI-generated code frequently contains vulnerabilities [23, 26], which have also been identified across real-world projects [13]. Practitioners have also highlighted risks of technical debt [9, 80]. These findings suggest that while vibe coding lowers barriers and accelerates development, it also introduces concerns regarding quality, security, and maintainability.

Despite the widespread adoption in practice, there is still no research that systematically examined vibe coding as a distinct

ICSE-SEIP '26, Rio de Janeiro, Brazil
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2426-8/2026/04
https://doi.org/10.1145/3786583.3786866

practice. Existing studies either focus on general AI code generation tools use without isolating the intuition-driven trial-and-error style that defines vibe coding [19, 20, 27, 36]. Vibe coding diverges from general AI-assisted programming in both workflow and level of oversight. In vibe coding, vibe coders describe the goal in natural language, run the generated code immediately, and iterate through trial-and-error without reviewing or fully understanding the code. In contrast, traditional AI-assisted programming uses AI as a helper while the developer still plans, reviews, tests, and maintains deliberate control over the development process. To address this gap, we conduct a systematic grey literature review (GLR) to analyze firsthand behavioral accounts of vibe coding documented in blogs, forums, media articles, and other publicly available sources. In this review, we attempt to answer the following four research questions:

RQ1: What are the motivations behind vibe coding?
This question examines why vibe coders opt to vibe code, delving deeper into their motivations for choosing to code in this manner (such as speed, ease of use, creativity, learning, or accessibility). We also explain the contexts in which vibe coders find the practice valuable (e.g., when rapid prototyping, working on personal projects, or empowering non-software developers).

RQ2: What is the vibe coder experience while vibe coding?
In this question, the focus is on vibe coders' hands-on experiences while vibe coding, exploring what worked well and what went wrong. We aim to capture how vibe coders actually work with AI code generation tools during vibe coding.

RQ3: What is the perception of the AI-generated code quality with vibe coding ?
The question investigates how vibe coders perceive the outputs they produce with AI code generation tools. It captures how vibe coders judge the usefulness and reliability of the code they obtain.

RQ4: What quality assurance (QA) practices are applied when vibe coding?
Here, we examine how vibe coders assess or manage the quality of AI-generated code, and whether they carefully review it before acceptance or instead fully trust the output without verifying its correctness and quality.

By synthesizing insights from diverse practitioners and settings, our study offers a grounded account of how vibe coding is practiced in real-world contexts. We find that vibe coders in our dataset are primarily motivated by speed and accessibility, but their reliance on minimal review is associated with reports of fragile or error-prone code. These insights offer implications for tool designers and software teams (e.g., designing tools that encourage review and validation rather than uncritical acceptance of AI-generated code).

2 Related Work

Research on AI code generation tools has grown rapidly, focusing on different vibe coder groups and concerns. While these studies provide valuable insights, no direct investigation has been done on vibe coding as a distinct practice.

Some studies have examined how students and novices use AI code generation tools in learning contexts. Prather et al. [28, 29] found that students often accept AI suggestions without deeply considering or understanding them, leading to confusion when errors arise. Zviel-Girshin et al. [41] studied a full class of beginner software developers and reported that while students felt more confident using AI code generation tools, they often did not fully grasp the concepts behind the AI-generated code. Sheard et al. [31] interviewed instructors, many of whom expressed concern that students might submit AI-generated work without understanding it. Zi et al.[40] similarly found that CS1 students struggled to understand LLM-generated code, with only 32.5% success in comprehension tasks due to unfamiliar coding styles, automation bias, and limited experience. Overall, these findings suggest that while AI code generation tools can support confidence and accessibility, over-reliance may harm learning outcomes [15, 30].

Other work has examined individuals with no formal programming training. Feldman and Anderson [11] studied how non-software developers use AI code generation tools to generate working code. While they were able to produce basic programs, they struggled to articulate their intent in prompts clearly and to verify whether the resulting AI-generated code was correct. These findings suggest that while AI code generation tools open access to new vibe coder groups, significant barriers remain in effectively prompting and validating AI outputs.

Ferino et al. [12] investigated how junior developers adopt GitHub Copilot, finding that many relied on trial-and-error and accepted suggestions without full understanding. At the professional level, Barke et al. [3] found that experienced developers used AI code generation tools in two main ways: (1) to accelerate code they already knew how to write, and (2) to explore unfamiliar ideas, while generally testing results carefully. Vaithilingam et al. [36] found that developers enjoyed using Copilot but often struggled to fix generated code when bugs appeared. Peng et al. [27] reported productivity benefits, particularly for less experienced developers, though the quality of AI-generated code was not evaluated. Together, these studies suggest that developer experience plays a role in how carefully AI code generation tools are used.

In prompt engineering, Kruse et al. [20] investigated how developers craft prompts for code generation and how their experience influenced outcomes. While this sheds light on the importance of prompt design, it does not specifically examine vibe coding, where trial-and-error prompting and intuition often dominate.

Several studies highlight risks associated with adopting AI code generation tools without sufficient code review. Pearce et al. [26] found that approximately 40% of Copilot outputs (out of 1,689 programs) contained security vulnerabilities. Majdinasab et al. [23] showed that even with additional safety layers, insecure code was still frequently produced. Fu et al. [13] further identified security weaknesses in AI-generated code across GitHub projects. These findings show that while AI code generation tools increase accessibility and speed, the resulting code may introduce significant risks if not carefully checked.

Beyond technical outcomes, some studies explore how developers perceive the role of AI code generation tools. Kuhail et al. [21] examined developers' views on ChatGPT, focusing on job security, role changes, and whether AI would replace or augment programming work. Weisz et al. [37] studied IBM's use of an AI code assistant (watsonx Code Assistant) and found that, although many developers reported productivity gains, these benefits were uneven and often raised concerns about authorship, responsibility, and loss of skills. While relevant to understanding adoption, this research does not investigate how developers actually check or test AI-generated code in practice.

In summary, prior research has examined educational contexts, non-software developers, juniors, professionals, prompting, security, and developer perceptions. However, none directly addresses vibe coding as an intuition-driven trial-and-error practice distinct from general AI-assisted software development. This gap motivates our GLR, which synthesizes firsthand accounts of vibe coding behaviors across diverse vibe coder groups.

3 Methodology

Grey literature encompasses sources that are not published in traditional peer-reviewed venues, including blogs, technical reports, and web articles. In software engineering, practitioners often share their experiences and practices via these online channels rather than academic journals [16, 17]. Several studies have demonstrated that GL is becoming a significant source of evidence in SE. For example, a review found that approximately one in five secondary studies already utilize GL, as it incorporates perspectives from real practitioners that may not be evident in peer-reviewed research [16]. Kamei et al. [17] also argue that GL reviews are useful because they capture insights from materials that practitioners themselves read (like blog posts and technical reports).

3.1 Search Strategy

Before conducting the review, we performed a preliminary scoping search across major academic databases (ACM DL, IEEE Xplore, SpringerLink, Scopus), which returned no similar peer-reviewed studies on vibe coding. This was expected, given that the term was coined only recently by Karpathy in early 2025 [18], and therefore, limited academic research has emerged since then. Because an SLR or multivocal review requires a substantial body of scholarly work, we concluded that a GLR would be the only feasible method for capturing current practitioner-led accounts of vibe coding.

In conducting this review, we closely followed the review guidelines proposed by Garousi et al. [14]. As grey literature is spread across the web, we used *Google* as our search engine. Prior work has shown that Google is particularly effective for discovering diverse forms of grey literature across blogs, forums, and media posts [1, 2, 35]. Although using Google search may not show all grey literature in the top results, we reduced this risk 1) through backward snowballing (i.e., following links and references inside included sources) to find further relevant sources (see Section 3.2.5), 2) a quasi–gold standard to validate recall, and 3) an iterative refinement of the search string until all benchmark sources were consistently retrieved. These steps helped mitigate the limitations of Google's ranking and reduce the likelihood that relevant materials were overlooked. Figure 1 outlined the full steps of the GL process. We discussed our search string development, data extraction, and analysis steps below.

3.2 Search String

We followed an iterative, multi-stage process to develop an effective search string. Our goal was to maximize both recall and practical relevance. We explained our string development in the following stages:

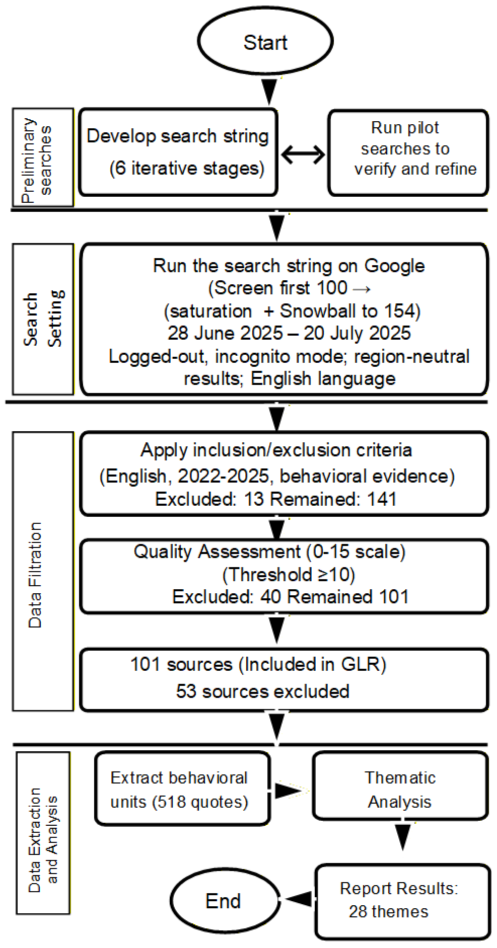
Start
Preliminary searches
Develop search string (6 iterative stages)
Run pilot searches to verify and refine
Search Setting
Run the search string on Google (Screen first 100 → (saturation + Snowball to 154) 28 June 2025 – 20 July 2025 Logged-out, incognito mode; region-neutral results; English language
Data Filtration
Apply inclusion/exclusion criteria (English, 2022-2025, behavioral evidence) Excluded: 13 Remained: 141
Quality Assessment (0-15 scale) (Threshold ≥10) Excluded: 40 Remained 101
101 sources (Included in GLR) 53 sources excluded
Data Extraction and Analysis
Extract behavioral units (518 quotes)
Thematic Analysis
Report Results: 28 themes
End

Figure 1: Grey Literature Process

3.2.1 *Stage 1: Problem Framing and Concept Definition.*

Vibe coding refers to situations where software developers and other vibe coders (e.g., novices / non-software developers) rely on AI code generation tools based on intuition and trial and error, often without a thorough understanding of the code or conducting a rigorous review and testing. This distinguishes it from more structured or expert-guided AI-assisted programming. Therefore, our search needed to focus on behavioral indicators and practitioner phrasing, not just technical descriptions.

3.2.2 *Stage 2: Initial Candidate Terms.*

We began with domain-informed candidate terms commonly seen in early articles, consisting of the following terms:

("AI-assisted programming" OR "AI code generation" OR "prompt-based programming" OR "LLM-assisted development") + tool names (e.g., "ChatGPT" or "Copilot") + role (e.g., "developer" OR "student" OR "non-programmer")

However, this term combination had limited success. The search retrieved mostly unrelated sources (e.g., more technical AI articles, promotional content, or conceptual content) and missed the primary target of our GLR. The first 100 results (a manageable representation of relevant grey literature [2, 7]) of this stage were screened against our inclusion and exclusion criteria (see Section 3.3), but since they did not contain firsthand evidence of vibe coding behaviors, they were excluded (screened 100, 8% relevant). This stage nevertheless helped us refine our search strategy by clarifying which terminologies were too broad or non-practitioner-focused.

3.2.3 Stage 3: Pivot to Practitioner Language. We then shifted focus to how practitioners describe AI coding tools in everyday usage. We noticed that several terms were frequently used, including: "coding with AI", "AI writes code", "prompting for code", "using AI to code", "programming with an AI assistant". We also incorporated the term "`vibe coding`" directly, as it had become widely adopted following its introduction in 2025 [18]. In this stage, we screened 100 sources, with approximately 50% considered relevant.

3.2.4 Stage 4: Iterative Expansion and Saturation. As part of an iterative design and test process, we expanded the candidate string to capture semantically adjacent behaviors. We added terms such as "`LLM-assisted coding`", "`AI programming tool`", and "`AI pair programmer`", which returned relatively few additional relevant hits (12%) when we screened 100 results, based on their appearance in pilot searches and known grey literature sources (Quasi-Gold Standard see Section 3.2.6). Each term was tested for recall and specificity using Google search, and retained only if it increased the relevance of retrieved documents.

3.2.5 Stage 5: Final Search String. The final version of the search string retrieved all items in the quasi-gold standard set and produced strong recall without adding excessive unnecessary information. We initially screened the first 100 results of the final search string, then continued until we reached thematic saturation (additional screening no longer surfaced substantially new related sources). Backward reference (Snowballing) chaining was applied to the first 100 results of the final search string retrieved sources, following the guidelines of Wohlin et al [39]. This involved tracing hyperlinks, named references, or cited resources within blogs and forums (e.g., [79, 80],[59, 60]). This led to a total of 154 sources.
The String:

"vibe coding" OR "coding with AI" OR "AI writes code" OR "AI code assistant" OR "LLM-assisted coding" OR "AI programming tool" OR "prompting for code" OR "using AI to code" OR "AI pair programmer" OR "programming with an AI assistant"

3.2.6 Stage 6: Quasi-Gold Standard Validation and Pilot Search. To ensure the effectiveness of the final string, we employed a *quasi-gold standard* evaluation approach. Additionally, this involved testing each search iteration against a curated set of known relevant grey literature sources, posts we had previously identified as clear examples of vibe coding behavior. These sources were identified during our pilot search stages and met all inclusion criteria (see Section 3.3) with high quality scores (see Section 3.4). The quasi-gold set included firsthand accounts involving AI prompting, minimal oversight, trial-and-error acceptance, or skipped QA practices. Specifically, it comprised: Andrej Karpathy, who coined the term *vibe coding* in early 2025 and demonstrated it by developing a restaurant menu application using this approach [52], including clear experiential reflections on the process and deployment; Simon Willison, who emphasized that not all AI-assisted programming is vibe coding [80], and described his own prototyping workflow and personal experiences using LLMs to vibe code [79]; and Maxime Najim, An experienced software engineer, whose reflective post documented the intentional and risky use of vibe coding in practice [57]; and Tyler Shields, an early practitioner account that described vibe coding as relying entirely on LLMs without verification [69]. This validation process increased our confidence that the final search string would capture the full range of behavioral evidence needed to address our research questions, and was aligned with the guidelines of Garousi et al. [14].

3.3 Inclusion and Exclusion Criteria

We applied a set of inclusion and exclusion criteria. Our inclusion criteria were:

- English-language sources
- Sources published between 2022 and 2025
- Relevance to the research questions
- Identifiable author or publishing entity
- Publicly accessible full text
- Behavioral evidence of vibe coding, including: AI-generated code used as-is or with light edits, prompting and reprompting, minimal or no testing, review, or improvement
- Critical or negative reflections are accepted if based on firsthand experience with vibe coding

We excluded material that met any of the following exclusion criteria:

- Duplicates or mirrored content
- Irrelevant topics or purely technical focus
- Promotional content
- Undated or anonymous sources
- Tool descriptions without vibe coder insight (e.g., Copilot/ChatGPT feature summaries)
- Very short content (e.g., tweets or headlines without commentary)
- Abstract speculation or general opinions unsupported by observable vibe coding actions such as prompting, accepting, skipping QA.
- No firsthand behavior involving vibe coding, such as:
  - No prompting or AI code generation
  - No code acceptance or editing
  - Experienced developers applying critical oversight without skipping QA

We included sources that contained at least one firsthand description of vibe-coding behaviour aligned with any of our RQs. A source did not need to address all RQs; a single relevant behavioural unit was sufficient for inclusion. Sources that mentioned vibe coding only superficially or without behavioural evidence were excluded. For identifiable authorship, we required a named individual or an attributable organisation (e.g., a company blog or media outlet). Anonymous posts, content without a traceable origin, or posts from unverifiable accounts were excluded to ensure source credibility, consistent with the authority criterion in the grey literature quality assessment framework we applied.

3.4 Quality Assessment

The quality assessments were mainly conducted by the first author, with each source scored from 0 to 3 across five quality dimensions, following Garousi et al.'s GL guidelines [14] with a maximum score of 15. These five quality criteria (Authority, Evidence, Objectivity, Currency, and Purpose) were adapted from Garousi et al.'s [14] framework for grey literature quality assessment, which has been widely adopted in software engineering GLR [16, 33]. To increase reliability, a subset of sources was cross-validated by another co-author. Any disagreements during the quality assessment were resolved through discussions until consensus was reached. Sources scoring ≥ 10 were retained for further analysis. The full scoring descriptions, including the results and justification for each article quality score, have been included in our dataset [10].

Scoring Criteria:

- **Authority:** Author identity and credibility (e.g., known developer, reputable organization)
- **Evidence:** Use of examples, rationale, or empirical support
- **Objectivity:** Neutral and balanced presentation of ideas
- **Currency:** Published between 2022 and 2025
- **Purpose:** Intended to inform, reflect, or explain (not promote or market)

3.5 Data Filtration

Our search string retrieved 154 grey literature sources (e.g., blog posts, online articles, and technical reports), of which 101 met our inclusion criteria and quality thresholds. All other 53 sources were excluded. Among the exclusions, 40 sources were excluded because they did not reach the minimum quality threshold (QA score ≥ 10/15), and 13 were excluded based on the inclusion and exclusion criteria (see Section 3.3). For example, [77] (a Wikipedia entry), [61] (a promotional Replit blog post), and [70] (a satirical news commentary in The Register) were excluded for not meeting our quality thresholds for behavioral detail/provenance. Similarly, [51] (an explanatory article in The Conversation), [43] (a Cloudflare vendor learning page), and [44] (a Google Cloud conceptual overview) were likewise excluded under our inclusion/exclusion criteria and for lacking firsthand behavioral evidence (no prompting, code acceptance, or QA evidence).

3.6 Behavioral Unit Extraction

To move from raw articles to analyzable data, we systematically extracted 518 *behavioral units* from the 101 included sources. A behavioral unit is a single coded instance that captures something relevant to the RQs. This may be a practitioner describing their own vibe coding, or an author/reporter documenting or commenting on vibe coding practices. Extraction was performed at the quote level rather than the article level, coding each instance of behavior (motivation, experience, perceptions of code quality, or QA practice) separately. This ensured that multiple distinct behaviors within the same article (e.g., a motivation to save time and a later description of skipping QA) were each represented as separate units in our dataset [10].

Each behavioral unit was extracted, classified and recorded in an external spreadsheet with the following fields:

- **Verbatim Quote:** the exact wording from the source.
- **Attribution:** who expressed the behavior (author or quoted vibe coder)
- **RQ Mapping:** whether the unit described motivation (RQ1), experience (RQ2), perceptions of code quality (RQ3), or QA practice.
- **Interpretation:** a short analytic summary of what the behavior revealed
- **Metadata:** vibe coder type, tool mentioned, vibe coding definition (if given), and notes

Behavioral unit extraction was conducted mainly by the first author. To increase reliability, a subset of units was cross-validated by another co-author. This process yielded 140 motivation units (RQ1), 132 experience units (RQ2), 114 perception of code quality units (RQ3) and 132 QA practice units (RQ4). These units formed the raw data corpus for our subsequent thematic analysis (Section 3.7).

3.7 Data Analysis

We used thematic analysis to analyze the extracted behavioral units of grey literature data, following the procedures recommended by Braun & Clarke [4–6]. Behavioral insights were organized and coded, guided by our research questions, according to patterns in motivations, experiences, perceptions of code quality, and quality assurance practices related to vibe coding. The data analysis steps, including theme development, were mainly conducted by the first author. To enhance reliability, reviews and discussions with other co-authors were carried out. We followed a set of steps to develop the themes, as described below:

- **Familiarization with the Data:** We read through all the extracted 518 behavioral units multiple times to understand patterns. For example, *"I just trust it works."* [65] and *"They're accepting the responses in a very trusting way."*[50] suggested a common pattern of uncritical trust in AI-generated code without any checks.
- **Generating Initial Codes:** We then labeled each behavioral unit with a potential theme. For example, *"Copy and paste them in… usually, that fixes it."* we labeled it as *"Reprompting Instead of Debugging"*. We set the potential theme "Speed" to *"Alex Finn famously created a Call of Duty style shooter game in just 87 minutes using AI tools."* [63]. These initial codes summarize the behaviors, but they do not group them yet.
- **Searching for Themes:** We grouped similar initial codes (potential themes) into larger candidate themes. This means we started clustering behaviors that reflected the same idea or pattern. The potential themes such as "Uncritical Trust" [75], "False Confidence" [48], and "Uncritical Security Trust" [47] were grouped into the final theme "Uncritical Trust".
- **Reviewing Themes:** We examined whether our potential themes were consistent throughout the entire dataset. Sometimes we split, rename, or reassign behaviors to better-fitting themes. For example, some quotes originally coded as "Skipped QA" were actually different in behavior. For instance, *"Businesses often trust AI-generated code uncritically, leading to vulnerabilities and technical debt"* [63] were reassigned to the final theme "Uncritical Trust" as they choose to trust without understanding.
- **Defining and Naming The Final Themes:** For each theme we identified, we wrote a clear description that explained what

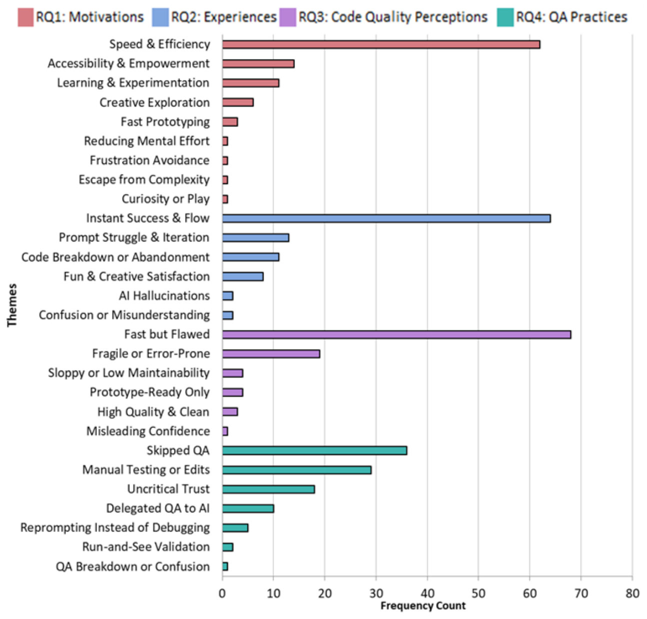

Figure 2: The identified themes from the collected sources

the theme represented and how it was different from the other themes. A consolidated overview of all theme definitions is provided in Table 1.

- **Producing the Report:** Finally, we prepared the results for presentation. Each final theme is reported in Section 4 together with: (its definition, frequency statistics, and illustrative explanations).

Table 1 summarizes the theme definitions used to code behavioral units across RQ1 (motivations), RQ2 (experiences), RQ3 (perceptions of AI-generated code quality) and RQ4 (quality assurance practices). Figure 2 displays all the vibe coding behavior themes we have constructed, along with their corresponding frequencies.

4 Results

From these 101 sources, we identified 518 behavioral units, which we coded into the themes defined earlier (see Table 1 and Figure 2), including motivations (140 units, RQ1), experiences (132 units, RQ2), perceptions of code quality (114 units, RQ3), and QA practices (132 units, RQ4). For each RQ, we identified several distinct themes, which are summarized with definitions in Table 1. Due to space limitations, we describe in detail only themes that account for at least 10% of the behavioral units for that RQ. Less frequent themes are still included in Table 1 and distribution tables (Tables 2–5). Percentages reported here reflect the distribution of coded behavioral units within our grey literature dataset and should not be interpreted as population-level prevalence.

4.1 Motivations for Vibe Coding (RQ1)

There were 140 behavioral units related to motivation, from which we identified nine distinct themes through thematic analysis. A summary of those motivational themes is shown in Table 2. A distribution of the motivational themes, including their frequencies and percentages, is shown in Table 2.

Speed & Efficiency: Within our dataset, the most common motivation theme for vibe coding (62%) is speed & efficiency, with vibe coders highlighting rapid development. Practitioners consistently described how AI code generation tools enabled them to produce working software in dramatically less time, often hours instead of weeks (e.g., [56, 80]). For example, one source reported that they have built over 140k lines of code (LOC) with tests and documentation in just under 15 days using vibe coding practices [64]. This acceleration was valued not only by beginners but also by experienced developers and even organizations [58]. Across these groups, speed was not just about coding faster; it translated into tangible productivity outcomes. For instance, one source expects that the company will reduce manual testing time by 25%, achieve complete test coverage on projects 20% faster, and fix significantly more bugs early in the development cycle [73].

Accessibility & Empowerment: Another important motivation theme (14%) was accessibility and empowerment, where vibe coding lowers the barrier to entry for software development. Non-software developers described being able to turn their ideas into functional applications simply by expressing them in natural language to AI code generation tools (e.g., [56, 78]). This democratization represents a major shift in who can participate in software creation: tasks that once required teams of technical specialists can now be achieved by individuals without programming training. Evidence of this appeared not only in personal projects, but also in organizational contexts, where analysts and policy staff reported building automations or public-facing forms without relying on IT departments (e.g., [42, 54]). Overall, this theme highlights how vibe coding is not only about speed but also about expanding access to software creation for entirely new groups of vibe coders.

Learning & Experimentation: A further motivation theme (11%) was learning and experimentation, where vibe coding was used as both a hands-on tutor and a sandbox for trying out ideas. Practitioners described how experimenting with prompts and outputs helped them quickly build intuition about what works and what does not [79]. Vibe coding also enabled coders to explore new programming languages and frameworks without formal training, with novices and career changers reporting that it accelerated their learning and confidence in coding tasks [46, 52]. Even experienced developers reported using code-generation tools to familiarize themselves with unfamiliar domains or frameworks more efficiently - a complementary way to learn through practice [45, 79].

4.2 Experiences During Vibe Coding (RQ2)

There were 132 behavioral units related to experiences, from which we identified six distinct themes through thematic analysis. A summary of these themes is provided in Table 3. A distribution of the experience themes, including their frequencies and percentages, is shown in Table 3.

Instant Success & Flow: In our dataset, the most common experience theme (64%) was instant success and flow, where vibe coders described the process as fast, easy, and often "Magical." Practitioners reported being able to build working applications in minutes through simple conversations with AI code generation tools [53, 67, 68]. This immediacy created a sense of addictive momentum, with some describing it as a "dopamine hit" when prototypes, QA checks, and deployments came together quickly

Table 1: Description of the identified vibe coding themes

Theme	Description
Motivation for vibe coding (RQ1)	
Speed & Efficiency	The ability to produce working software much **faster, reducing development cycles** from weeks to only hours.
Accessibility & Empowerment	Enabling **Non-Software Developers** ability to create applications by describing goals in natural language.
Learning & Experimentation	Using LLM coding models as a **tutor or sandbox** to explore new coding tools, concepts, and frameworks.
Creative Exploration	Turning imaginative or artistic **ideas into functional projects** with AI support.
Fast Prototyping	Quickly assembling MVPs or **demos to test feasibility**, concepts, or market interest.
Reducing Mental Effort	Offloading syntax and boilerplate to AI, **easing the developer's cognitive load**.
Frustration Avoidance	**Bypassing programming pain points** e.g., repetitive debugging, setup issues etc.
Escape Complexity	Avoiding difficult design or architecture work by **delegating detail to AI**.
Curiosity or Play	Engaging casually with **AI for fun**, experimentation, or open-ended exploration.
Experience with vibe coding (RQ2)	
Instant Success & Flow	Experiencing **rapid, seamless progress** that creates momentum, leading to satisfaction.
Prompt Struggle & Iteration	**Refining prompts through repeated trial-and-error** until usable results appear.
Code Breakdown or Abandonment	**Abandoning projects** when AI outputs become too buggy or complex.
Fun & Creative Satisfaction	**Enjoying the excitement of fast software development**, though the novelty can fade.
AI Hallucinations	Facing **false, inaccurate, or misleading code suggestions** that look plausible but fail in execution or introduce bugs.
Confusion or Misunderstanding	**Misaligned prompts and outputs** causing frustration and loss of trust
Code Quality Perception (RQ3)	
Fast but Flawed	Useful for **quick tasks but unsuitable for production** deployment.
Fragile or Error-Prone	Containing **hidden bugs**, inconsistencies, or potential security risks.
Sloppy or Low Maintainability	Functional but **poorly structured**, undocumented, and hard to extend.
Prototype-Ready Only	Adequate for **demos and proofs-of-concept** but not for long-term systems.
High Quality & Clean	Actually **well-structured**, and close to production quality
Misleading Confidence	Appearing reliable while concealing deeper flaws, **creating false trust**.
QA Practices (RQ4)	
Skipped QA	**Bypassing structured testing** or review, relying only on execution success.
Manual Testing or Edits	Applying systematic checks (**manual code reviews**) and edits before adopting AI outputs.
Uncritical Trust	Accepting AI-generated code without validation or verification, **based on the assumption that it works**.
Delegated QA to AI	Relying on the **AI code generation tools to detect** and correct **its own mistakes**.
Reprompting Instead of Debugging	**Feeding errors back into AI code generation tools** rather than fixing them manually.
Run-and-See Validation	**Running code** to check if it works, equating success with correctness.
QA Breakdown or Confusion	**Failing to validate** the outputs due to complexity or lack of clarity.

Table 2: Distribution of Motivations for Vibe Coding (n=140)

Theme	Frequency	Percentage
Speed & Efficiency	87	62%
Accessibility & Empowerment	20	14%
Learning & Experimentation	15	11%
Creative Exploration	8	6%
Fast Prototyping	4	3%
Reducing Mental Effort	2	1%
Frustration Avoidance	2	1%
Escape Complexity	1	1%
Curiosity or Play	1	1%

Table 3: Distribution of Experiences During Vibe Coding (n=132)

Theme	Frequency	Percentage
Instant Success & Flow	85	64%
Prompt Struggle & Iteration	17	13%
Code Breakdown or Abandonment	14	11%
Fun & Creative Satisfaction	11	8%
AI Hallucinations	3	2%
Confusion or Misunderstanding	2	2%

[74]. Accounts ranged from developers building complete applications within hours [66] to non-software developers expressing amazement at creating functional tools and even cultural translation projects without prior coding expertise [82].

Prompt Struggle & Iteration: Not all experiences were positive. A common experience (13%) was prompt struggle and iteration, where vibe coders needed to refine their instructions repeatedly to achieve a satisfactory result. Practitioners described cycles of prompt adjustment and code refinement, with some projects even requiring hundreds of iterations before the output was usable [49, 60]. This process highlighted the emergence of prompt engineering skills, where vibe coders learned to craft, adapt, and even collect effective prompts as reusable patterns to improve their results.

Code Breakdown or Abandonment: Another experience theme (11%) was code breakdown or abandonment, where vibe coding sessions failed. When AI-generated code produced outputs that were too complex, buggy, or inconsistent to fix, some practitioners reported abandoning projects altogether rather than attempting to debug the code [78]. These breakdowns often occurred when task complexity exceeded the AI's ability to generate reliable solutions, leading to frustration and project abandonment [82].

Table 4: Distribution of Perceived Code Quality (n=114)

Theme	Frequency	Percentage
Fast but Flawed	78	68%
Fragile or Error-Prone	22	19%
Sloppy or Low Maintainability	5	4%
Prototype-Ready Only	5	4%
High Quality & Clean	3	3%
Misleading Confidence	1	1%

Table 5: Distribution of QA Practices in Vibe Coding (n=132)

Theme	Frequency	Percentage
Skipped QA	48	36%
Manual Testing or Edits	38	29%
Uncritical Trust	24	18%
Delegated QA to AI	13	10%
Reprompting Instead of Debugging	6	5%
Run-and-See Validation	2	2%
QA Breakdown or Confusion	1	1%

4.3 Perceptions of the Generated Code's Quality (RQ3)

There were 114 behavioral units related to perceptions of code quality, from which we identified six distinct themes through thematic analysis. A distribution of these code quality perception themes, including their frequencies and percentages, is shown in Table 4.

Fast but Flawed: We found that the most common reported perception of the AI-generated code quality theme is "fast but flawed" (68%), where practitioners acknowledged clear trade-offs between speed and long-term quality. Vibe coders noted that while AI code generation tools could quickly generate most of a solution, the remaining critical work to make code production-ready often became a challenge [68]. They accepted these flaws as an inevitable cost of rapid development, keeping the code as long as it worked but recognizing that this created technical debt over time [67].

Fragile or Error-Prone: Another common perception (19%) was that AI-generated code is fragile or error-prone, raising concerns about hidden issues. Practitioners cautioned that such code was often excluded from reviews or security checks, creating the risk of undetected vulnerabilities [49]. Others emphasized that while the outputs might appear clean and functional, they could conceal subtle logic errors, performance bottlenecks, or serious security flaws that only become apparent later [71].

4.4 QA Practices in Vibe Coding (RQ4)

There were 132 behavioral units related to QA practices, from which we identified seven distinct themes through thematic analysis. A distribution of these QA practice themes, including their frequencies and percentages, is shown in Table 5.

Skipped QA: The most common QA practice in our dataset (36%) was skipped QA, where vibe coders accepted AI-generated code without validation. Practitioners reported bypassing traditional testing entirely, for example, they did not write unit or integration tests, perform structured reviews, or systematically verify correctness beyond simply running the code. Instead, this meant relying on whether the code executed without errors as a proxy for quality [49]. Even experienced developers described pasting error messages back into AI code generation tools and letting it generate fixes, rather than debugging or testing the code themselves [56].

Manual Testing or Edits: The second most common QA practice (29%) was manual testing or edits, where practitioners applied careful quality control to AI-generated code. Some emphasized the risks of pushing generated code directly to production, warning that without review it could introduce bugs, security issues, or performance problems [58]. Experienced developers described establishing stricter review protocols, treating every generated change as something that needed to be understood and verified, often supported by testing and automated checks [57].

Uncritical Trust: Another QA practice (18%) was uncritical trust, where vibe coders believed the code worked even without checking it. As one source noted, humans tend to place more faith in generated code than warranted, failing to scrutinize it as carefully as code written by a fellow developer [75]. This trust also extended to complex systems, with coders no longer checking outputs line by line but simply accepting the responses without scrutiny [66].

Delegated QA to AI: Delegating quality checks back to the AI itself (10%) is one of the QA practices reported. As reported in [72], vibe coders were overly reliant on the same LLMs that introduced errors to fix them, creating a false sense of security.

5 Discussion

5.1 Key Observations

The Speed–Quality Trade-off Paradox: Our results highlight a paradox: Figure 3 demonstrated the speed–QA trade-off uncovered in our analysis. The horizontal axis reflects the extent to which sources emphasised speed, while the vertical axis indicates the level of QA engagement. The four bubbles represent the main behavioural patterns. Many sources describe knowingly accepting flawed AI-generated code in exchange for rapid progress. Development speed motivates all groups, but the trade-off manifests differently across backgrounds and experiences. Non- and novice software developers often acknowledge that they can build applications quickly while conceding that *"it's not really coding"* [56], reflecting both excitement and awareness of possible limitations. Experienced software developers also value speed, but balance it with caution, as 29% reported that they typically make some manual adjustments or add tests to the generated code, showing risk-aware behavior. Developers sometimes apply extensive modifications to the generated code to the level that they no longer consider it AI-generated [19]. This divide indicates that while vibe coders are willing to tolerate imperfect code for speed, only experienced vibe coders have the skills to address problems as they arise. This observation is also reflected in the *Stack Overflow Developer Survey 2025*, which reports both high adoption of AI code generation tools (84% use or plan to use them) and low trust in AI-generated code (~46% report distrust, 33% report trust, 3% report "high trust", while the rest are neutral or unsure) [34]. The implication is a divide between two groups of vibe coders: empowered novices who may

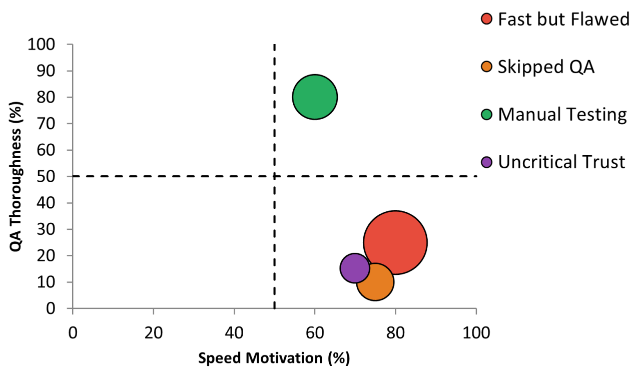

Figure 3: Speed vs QA Trade-off in Vibe Coding

remain dependent on AI code generation tools, and professionals who integrate selective QA practices.

Recommendation for Practitioners: *Use vibe coding to explore and prototype quickly, but never promote to production without adding guardrails: tests, code review, and traceable decision records (why the AI change was accepted, which checks passed, any accepted risks, and a short prompt/response ID log).*

The QA Crisis in AI-Assisted Development: A recurring concern in the reviewed sources is the frequent reports of skipping traditional QA practices. As shown in Section 4.4, QA-related behaviours such as Skipped QA, Uncritical Trust, Delegated QA to the AI, and Reprompting Instead of Debugging account for a large share of all behavioural units. Together, these patterns show that many vibe coders run or accept AI-generated code with minimal testing or review, indicating a consistent shift away from established QA practices. A majority of QA practices reflect a departure from code verification, with practitioners commonly skipping tests, placing uncritical trust in outputs, or delegating responsibility back to the AI code generation tools. These QA concerns are possibly due to multiple factors: *Technical barriers;* e.g., the AI-generated code is cited to be difficult to debug as it can lack architectural structure [49] and the contextual details software developers normally rely on, such as comments, assumptions, or information about how the code integrates with the larger system. *Confusion:* vibe coders report confusion when attempting to understand AI-generated code. *False confidence:* the "instant success" experience creates illusions of correctness. If current practices continue, vibe-coded apps may look fine but hide serious flaws. Practitioners already warn of this risk, with 19% admitting their code is "fragile or error-prone". This matters because once teams get used to shipping fragile or error-prone code without proper QA, they may lower the quality bar across the whole organization. Over time, this creates a culture where untested code is considered acceptable, raising the risk of costly failures, outages, and security breaches.

Recommendation for AI code generation tools designers: *AI code generation tools should incorporate lightweight verification processes and continuously remind vibe coders of the QA aspects (especially those with no formal software development experience) of the generated code. Including code (static and dynamic) analysis checks that can verify the generated code and remind developers of potential risks can be a useful tool for coders.* This can be offered as a real-time indicator (e.g., performance, security concerns, missing tests) of the generated code. Tools should address the *"Uncritical Trust"* issue by including features such as step-by-step code explanations ("walk-throughs"), visual diagrams, or inline explanations to clarify what the tool is doing and why.

New Class of Vulnerable Developers: Our analysis suggests the possible emergence of a new class of vulnerable developers. About 14% of vibe coders in our dataset are motivated by accessibility and empowerment, with non-software developers describing how AI code generation tools allow them to create applications without prior coding skills [82]. Yet, this democratization often leaves them unprepared when problems arise. Several sources illustrate how non-software developers quickly reach dead ends when faced with bugs or technical errors that they cannot diagnose or resolve [62, 81]. Others highlight how uncritical trust in AI code generation tools suggestions can lead to copy-paste development practices where fixes are applied without any real comprehension of their impact [62]. In practice, this over-reliance can introduce serious risks: practitioners have documented cases of insecure systems built through vibe coding, including applications that lacked authentication, authorization, or contained hardcoded secrets [55]. Beyond individual projects, such practices contribute to the rise of "shadow IT", where employees outside formal development teams build software without oversight or governance [76]. As some experts caution, the danger is not when AI-generated code fails outright, but when it appears to work while embedding subtle vulnerabilities and technical debt [48]. Taken together, these accounts suggest that while vibe coding can lower the barrier to entry, it can also transfer significant responsibility to vibe coders who may not yet possess the necessary skills to manage it effectively.

Recommendation for Organizations and Practitioners: *Match tasks to skill and provide scaffolds (guided debugging, safe templates, and escalation paths), so newcomers learn to diagnose issues rather than outsource all QA to the AI.* This is particularly important to avoid the novice developer trap, where failures can lock beginners into reprompt–paste loops, accepting fragile behavior when they cannot restore alignment between intent (their goal) and implementation (what the AI-generated code actually does).

Recommendation for Educators: Our findings indicate that vibe coding is often characterised by rapid progress, minimal code inspection, lightweight QA, and difficulty debugging when code fails. In the context of education, these behaviours may intersect with the development of foundational skills such as critical code reading, testing, and systematic debugging. Software engineering educators may benefit from designing assessments that prompt students to critically examine AI-generated code, rather than relying on them uncritically. Such assessments could encourage students to articulate their reasoning, verify correctness, and demonstrate an understanding of fundamental software engineering concepts when using AI code generation tools.

5.2 Future Work and Open Research Questions

Further research is needed on how vibe coders' practices shift with experience from non-software developers to novices to professionals, so that AI code generation tools can adapt feedback, explanations, and safeguards to the vibe coder's expertise. Further empirical evidence is needed to understand if these new practices will lead to new defect and vulnerability patterns unique compared to both conventional AI-assisted (used by experienced developers) and human coding [8, 22], to inform automated quality signals and assurance features in next-generation AI code generation tools.

It is also unclear which code review practices (e.g., run-and-see checks, automated tests, AI-assisted reviews) actually work under vibe coding conditions. Research in this area can inform the design of practical, built-in QA workflows that keep pace with rapid prototyping. Understanding how code review and QA strategies should differ for non-software developers, novices, and professionals can enable tools to enhance the quality of the generated code, providing guardrails for newcomers while supporting advanced workflows for experts. Future research should explore how vibe coding practices can be integrated into training for non-software developers, so that educational interventions complement tool design and enhance baseline QA competence. While vibe coding is useful for rapid exploration and prototyping, our findings suggest it should be used cautiously for tasks requiring reliability or maintainability. In such cases, traditional AI-assisted programming (with deliberate review and testing) remains more appropriate. Overall, the coding style should match the risk level of the task.

6 Threats to Validity

Internal Validity: To address possible *search bias* in our GLR, we first designed a comprehensive search strategy. We employed a detailed search strategy with iterative pilot searches to refine the search terms (see Section 3.2). This approach broadened coverage and reduced search string bias, but given the diffuse and evolving nature of grey literature, we cannot claim to have captured all relevant sources. We also defined a set of *inclusion and exclusion criteria* (see Section 3.3) and applied them consistently during the data filtration to minimize *selection bias*. To minimize *quality bias*, we applied Garousi et al.'s five-dimension quality checklist [14] (Section 3.4). This reduces the likelihood that anecdotal or promotional content disproportionately influenced the findings, although there is still some risk of quality bias.

With regards to *data extraction bias*, to avoid emphasizing certain quotes or themes, we used a standardized extraction template and applied Braun and Clarke [4] structured thematic analysis steps (see Section 3.7). Coding was conducted primarily by the first author, with discussions and consensus building among the other authors. Pilot extractions and iterative refinement helped calibrate interpretation in findings. However, thematic coding inevitably contains subjective judgment, so our findings should be understood as patterns drawn from the data rather than precise measurements.
External Validity: Our study is based on 101 grey literature sources, which reflect the experiences of practitioners who chose to share their stories online. This introduces *self-selection bias*, those represented may not reflect all vibe coders. We mitigated this by ensuring diversity in high-quality source types (blogs, forums, and media articles) and user roles (novices and experienced users) from various industries. However, the findings cannot be considered generalizable to all vibe coders. Instead, they should be interpreted as indicative of common patterns in reported experiences rather than as representative of the entire population of vibe coders. Frequencies and percentages illustrate patterns in the analyzed sources but do not imply statistical significance or generalizability beyond the reviewed grey literature.
Construct Validity: To establish a consistent data extraction basis for our GLR, we utilized a standard data extraction template to ensure uniformity. We refined our data extraction methods after several pilot tests. Reviews and discussions with other co-authors were carried out for the extracted data process, resolving discrepancies through consensus. Behavioral unit extraction was conducted mainly by the first author. To increase reliability, a subset of units was cross-validated by another co-author, minimizing the likelihood of inaccurate conclusions. This reduced, but did not eliminate, the risk of subjective bias and inaccurate conclusions.

7 Conclusion

This study provides the first empirical investigation of how vibe coders actually engage in vibe coding, especially outside formal development settings. By systematically analyzing 101 grey literature sources containing 518 firsthand behavioral units, we uncovered why vibe coders engage in vibe coding, what they experience while doing so, how they perceive the quality of AI-generated code, and what practices they apply to review or test the code.

Our findings reveal a speed–quality trade-off in vibe coding; vibe coders, particularly those without software development experience, are enabled to create usable applications quickly, which often comes at the expense of verification and maintainability limitations. The widespread use of delegated QA to AI, uncritical trust, and skipped testing suggests that vibe coding outputs can be vulnerable. Moreover, the emergence of *vulnerable developers*(who are capable of building but unable to debug) highlights the risks of democratizing software creation without corresponding investments in quality practices. For tool designers, our results suggest including QA feedback, visualizing code quality indicators, and providing inline explanations to mitigate the impact of uncritical trust. For teams adopting AI-assisted development, cautious use of vibe coding in production systems is recommended. Adhering to organizations' guardrails that require review processes and maintaining debugging skills. AI code generation tools support will play a bigger role in software development in the future. How we respond to the behavioral patterns identified in this study will determine whether the future improves or degrades software quality. The AI code generation tools exist; the challenge lies in using them wisely.

Acknowledgments

The project is partially supported by a Rutherford Discovery Fellowship, administered by the Royal Society Te Apārangi, New Zealand.

References

[1] Jean Adams, Frances C. Hillier-Brown, Helen J. Moore, Amelia A. Lake, Vera Araujo-Soares, Martin White, and Carolyn Summerbell. 2016. Searching and synthesising 'grey literature' and 'grey information' in public health: critical reflections on three case studies. *Systematic Reviews* 5, 1 (2016), 164. https://doi.org/10.1186/s13643-016-0337-y

[2] Richard J. Adams, Paul Smart, and Anne S. Huff. 2016. Searching for grey literature for systematic reviews: challenges and benefits. *Research Synthesis Methods* 7, 3 (2016), 263–281. https://doi.org/10.1002/jrsm.1106

[3] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[4] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77.

[5] Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative research in sport, exercise and health* 11, 4 (2019), 589–597.

[6] Virginia Braun and Victoria Clarke. 2021. Thematic analysis: A practical guide. (2021).

[7] Simon Briscoe and Morwenna Rogers. 2024. An alternative screening approach for Google Search identifies an accurate and manageable number of results for a systematic review (case study). *Health Information & Libraries Journal* 41, 2 (2024), 149–155.

[8] Domenico Cotroneo, Cristina Improta, and Pietro Liguori. 2025. Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity. *arXiv preprint arXiv:2508.21634* (2025).

[9] Benj Edwards. 2025. Will the Future of Software Development Run on Vibes? https://arstechnica.com/ai/2025/03/is-vibe-coding-with-ai-gnarly-or-reckless-maybe-some-of-both/. [Accessed on 01/05/2025].

[10] Ahmed Fawzy, Amjed Tahir, and Kelly Blincoe. 2025. *Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook - a Grey Literature Review (Replication package)*. https://doi.org/10.5281/zenodo.17188019

[11] Molly Q Feldman and Carolyn Jane Anderson. 2024. Non-expert programmers in the generative AI future. In *Proceedings of the 3rd Annual Meeting of the Symposium on Human-Computer Interaction for Work*. 1–19.

[12] Samuel Ferino, Rashina Hoda, John Grundy, and Christoph Treude. 2025. Junior Software Developers' Perspectives on Adopting LLMs for Software Engineering: a Systematic Literature Review. *arXiv preprint arXiv:2503.07556* (2025).

[13] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, Jiaxin Yu, and Jinfu Chen. 2025. Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study. *ACM Transactions on Software Engineering and Methodology* (2025).

[14] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and software technology* 106 (2019), 101–121.

[15] Philipp Haindl and Gerald Weinberger. 2024. Does ChatGPT help novice programmers write better code? Results from static code analysis. *IEEE Access* (2024).

[16] Fernando Kamei, Igor Wiese, Crescencio Lima, Ivanilton Polato, Vilmar Nepomuceno, Waldemar Ferreira, Márcio Ribeiro, Carolline Pena, Bruno Cartaxo, Gustavo Pinto, and Sérgio Soares. 2021. Grey Literature in Software Engineering: A critical review. *Information and Software Technology* 138 (2021), 106609. https://doi.org/10.1016/j.infsof.2021.106609

[17] Fernando Kenji Kamei. 2019. The use of grey literature review as evidence for practitioners. *ACM SIGSOFT Software Engineering Notes* 44, 3 (2019), 23–23.

[18] Andrej Karpathy. 2025. There's a new kind of coding I call "vibe coding". https://x.com/karpathy/status/1886192184808149383. [Accessed on 01/05/2025].

[19] Syed Mohammad Kashif, Peng Liang, and Amjed Tahir. 2025. On Developers' Self-Declaration of AI-Generated Code: An Analysis of Practices. *arXiv preprint arXiv:2504.16485* (2025).

[20] Hans-Alexander Kruse, Tim Puhlfurb, and Walid Maalej. 2024. Can Developers Prompt? A Controlled Experiment for Code Documentation Generation. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 574–586.

[21] Mohammad Amin Kuhail, Sujith Samuel Mathew, Ashraf Khalil, Jose Berengueres, and Syed Jawad Hussain Shah. 2024. "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming* 235 (2024), 103111.

[22] Sherlock A Licorish, Ansh Bajpai, Chetan Arora, Fanyu Wang, and Kla Tantithamthavorn. 2025. Comparing Human and LLM Generated Code: The Jury is Still Out! *arXiv preprint arXiv:2501.16857* (2025).

[23] Vahid Majdinasab, Michael Joshua Bishop, Shawn Rasheed, Arghavan Moradidakhel, Amjed Tahir, and Foutse Khomh. 2024. Assessing the security of github copilot's generated code-a targeted replication study. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 435–444.

[24] Ivan Mehta. 2025. A quarter of YC's startups have AI-generated codebases. https://techcrunch.com/2025/03/06/a-quarter-of-startups-in-ycs-current-cohort-have-codebases-that-are-almost-entirely-ai-generated. [Accessed on 01/05/2025].

[25] John Naughton. 2025. Now you don't even need code to be a programmer. https://www.theguardian.com/technology/2025/mar/16/ai-software-coding-programmer-expertise-jobs-threat. [Accessed on 01/05/2025].

[26] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2025. Asleep at the keyboard? assessing the security of github copilot's code contributions. *Commun. ACM* 68, 2 (2025), 96–105.

[27] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The impact of AI on developer productivity: Evidence from Github Copilot. *arXiv preprint arXiv:2302.06590* (2023).

[28] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's weird that it knows what i want": Usability and interactions with copilot for novice programmers. *ACM transactions on computer-human interaction* 31, 1 (2023), 1–31.

[29] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*. 469–486.

[30] Andreas Scholl and Natalie Kiesler. 2024. How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course. *arXiv preprint arXiv:2407.20792* (2024).

[31] Judy Sheard, Paul Denny, Arto Hellas, Juho Leinonen, Lauri Malmi, and Simon. 2024. Instructor Perceptions of AI Code Generation Tools-A Multi-Institutional Interview Study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1223–1229.

[32] Matthew S. Smith. 2025. Engineers Are Using AI to Code Based on Vibes. https://spectrum.ieee.org/vibe-coding. [Accessed on 01/05/2025].

[33] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. 2018. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software* 146 (2018), 215–232.

[34] Stack Overflow. 2025. Stack Overflow Developer Survey 2025. https://survey.stackoverflow.co/2025/. [Accessed on 11/08/2025].

[35] Amjed Tahir, Shawn Rasheed, Jens Dietrich, Negar Hashemi, and Lu Zhang. 2023. Test flakiness' causes, detection, impact and responses: A multivocal review. *Journal of Systems and Software* 206 (2023), 111837.

[36] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[37] Justin D Weisz, Shraddha Vijay Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai. 2025. Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–13.

[38] Simon Willison. 2025. Two publishers and three authors fail to understand what "vibe coding" means. https://simonwillison.net/2025/May/1/not-vibe-coding/. [Accessed on 04/05/2025].

[39] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.

[40] Yangtian Zi, Luisa Li, Arjun Guha, Carolyn Anderson, and Molly Q Feldman. 2025. "I Would Have Written My Code Differently': Beginners Struggle to Understand LLM-Generated Code. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 1479–1488.

[41] Rina Zviel-Girshin. 2024. The Good and Bad of AI Tools in Novice Programming Education. *Education Sciences* 14, 10 (2024), 1089.

References for Grey Literature

[G42] FedTec 2025. *Vibe Coding: Accelerating Service Delivery in the Government*. FedTec. https://fedtec.com/insights/vibe-coding-accelerating-service-delivery-in-the-government/ [Accessed on 08/07/2025].

[G43] Cloudflare 2025. *What is vibe coding?* Cloudflare. https://www.cloudflare.com/en-gb/learning/ai/ai-vibe-coding/ [Accessed on 07/07/2025].

[G44] Google Cloud 2025. *What is vibe coding?* Google Cloud. https://cloud.google.com/discover/what-is-vibe-coding [Accessed on 07/07/2025].

[G45] Guillaume Beaulieu-Duchesneau. 2025. *Vibe coding or "I let AI write all my code and it was fantastic (until it wasn't)"*. https://ingeno.ca/blog/vibe-coding-or-i-let-ai-write-all-my-code-and-it-was-fantastic-until-it-wasn-t/ [Accessed on 08/07/2025].

[G46] KitFu Coda. 2025. *Vibe-Coding, Storytelling, and LLMs: A Collaborative Approach.* https://ai.plainenglish.io/vibe-coding-storytelling-and-llms-a-collaborative-approach-db5e9a62c8b1 [Accessed on 07/07/2025].

[G47] Namanyay Goel. 2025. *Karpathy's 'Vibe Coding' Movement Considered Harmful.* https://nmn.gl/blog/dangers-vibe-coding [Accessed on 08/07/2025].

[G48] Namanyay Goel. 2025. *Vibe Coding is a Dangerous Fantasy.* https://nmn.gl/blog/vibe-coding-fantasy [Accessed on 08/07/2025].

[G49] Shalini Harkar. 2025. *What is Vibe Coding?* https://www.ibm.com/think/topics/vibe-coding [Accessed on 02/07/2025].

[G50] Nick Hodges. 2025. *Vibe coding with Claude Code.* https://www.infoworld.com/article/3853805/vibe-coding-with-claude-code.html [Accessed on 07/07/2025].

[G51] Chetan Jaiswal. 2025. *What is vibe coding? A computer scientist explains what it means to have AI write computer code and what risks that can entail.* https://theconversation.com/what-is-vibe-coding-a-computer-scientist-explains-what-it-means-to-have-ai-write-computer-code-and-what-risks-that-can-entail-257172 [Accessed on 02/07/2025].

[G52] Andrej Karpathy. 2025. *Vibe coding MenuGen.* https://x.com/karpathy/status/1917961248031080455 X (formerly Twitter). [Accessed on 07/07/2025].

[G53] Gene Kim. 2025. *Rearchitecting My Trello Management Tool with Claude Code using Vibe Coding/Architecting (Part 2).* https://itrevolution.com/articles/rearchitecting-my-trello-management-tool-with-claude-code-using-vibe-coding-architecting-part-2/ [Accessed on 08/07/2025].

[G54] Kelsey Libert. 2025. *Vibe coding with AI tools: A marketer's guide.* https://searchengineland.com/vibe-coding-with-ai-tools-a-marketers-guide-455134 [Accessed on 08/07/2025].

[G55] Rami McCarthy. 2025. *Rules Files for Safer Vibe Coding.* https://www.wiz.io/blog/safer-vibe-coding-rules-files [Accessed on 07/07/2025].

[G56] Niall McNulty. 2025. *Vibe Coding: AI-Assisted Coding for Non-Developers.* https://medium.com/@niall.mcnulty/vibe-coding-b79a6d3f0caa [Accessed on 02/07/2025].

[G57] Maxime Najim. 2025. *Vibe coding brought back my love for programming.* https://leaddev.com/culture/vibe-coding-brought-back-love-programming [Accessed on 08/07/2025].

[G58] Gergely Orosz and Elin Nilsson. 2025. *Vibe Coding as a Software Engineer.* https://newsletter.pragmaticengineer.com/p/vibe-coding-as-a-software-engineer [Accessed on 02/07/2025].

[G59] Addy Osmani. 2025. *Speech-to-Code: Vibe Coding with Voice.* https://addyo.substack.com/p/speech-to-code-vibe-coding-with-voice [Accessed on 07/07/2025].

[G60] Addy Osmani. 2025. *Vibe Coding: Revolution or Reckless Abandon?* https://addyo.substack.com/p/vibe-coding-revolution-or-reckless [Accessed on 07/07/2025].

[G61] Matt Palmer. 2025. *What is Vibe Coding?* https://blog.replit.com/what-is-vibe-coding [Accessed on 07/07/2025].

[G62] Katie Parrott. 2025. *I Tried AI Coding Tools. Now I Want to Learn to Code.* https://every.to/working-overtime/i-tried-ai-coding-tools-now-i-want-to-learn-to-code [Accessed on 07/07/2025].

[G63] ProfileTree. 2025. *Vibe Coding: How AI is Transforming Software Development in 2025.* https://profiletree.com/vibe-coding/ [Accessed on 07/07/2025].

[G64] Sujatha R. 2025. *10 Best Vibe Coding Tools: LLM-Powered Code Generators to Try.* https://www.digitalocean.com/resources/articles/vibe-coding-tools [Accessed on 07/07/2025].

[G65] David Ramel. 2025. *AI's Takeover of Software Development Gets a Name: 'Vibe Coding'.* https://visualstudiomagazine.com/articles/2025/03/20/ais-takeover-of-software-development-gets-a-name-vibe-coding.aspx [Accessed on 07/07/2025].

[G66] John Ruwitch. 2025. *Anyone can use AI chatbots to 'vibe code.' Could that put programmers out of a job?* https://www.npr.org/2025/05/30/nx-s1-5413387/vibe-coding-ai-software-development [Accessed on 07/07/2025].

[G67] Natassha Selvaraj. 2025. *Building Data Science Projects Using AI: A Vibe Coding Guide.* https://www.kdnuggets.com/building-data-science-projects-using-ai-a-vibe-coding-guide [Accessed on 07/07/2025].

[G68] Pete Sena. 2025. *Cracking the code of vibe coding.* https://uxdesign.cc/cracking-the-code-of-vibe-coding-124b9288e551 [Accessed on 07/07/2025].

[G69] Tyler Shields. 2025. *My first attempt at vibe coding.* https://www.techtarget.com/searchapparchitecture/opinion/My-first-attempt-at-vibe-coding [Accessed on 02/07/2025].

[G70] Richard Speed. 2025. *Forget Vibe Coding, We're All About Vine Coding Nowadays.* https://www.theregister.com/2025/06/13/forget_vibe_coding_were_all/ [Accessed on 07/07/2025].

[G71] Tanium Staff. 2025. *What is Vibe Coding? The Pros, Cons, and Controversies.* https://www.tanium.com/blog/what-is-vibe-coding/ [Accessed on 08/07/2025].

[G72] Chris Stokel-Walker. 2025. *What is vibe coding, should you be doing it, and does it matter?* https://www.newscientist.com/article/2473993-what-is-vibe-coding-should-you-be-doing-it-and-does-it-matter/ [Accessed on 07/07/2025].

[G73] Darryl K. Taft. 2024. *Amazon Q Developer Now Handles Your Entire Code Pipeline.* https://thenewstack.io/amazon-q-developer-now-handles-your-entire-code-pipeline/ [Accessed on 08/07/2025].

[G74] Steven Vaughan-Nichols. 2025. *Bad vibes: How an AI agent coded its way to disaster.* https://www.zdnet.com/article/bad-vibes-how-an-ai-agent-coded-its-way-to-disaster/ [Accessed on 08/07/2025].

[G75] Todd R. Weiss. 2025. *Vibe coding: What automating development means for AppSec.* https://www.reversinglabs.com/blog/vibe-coding-what-autonomous-code-means-for-appsec [Accessed on 08/07/2025].

[G76] Todd R. Weiss. 2025. *What MSPs Need to Know About Vibe Coding, Agentic AI and IT Security.* https://www.channele2e.com/news/what-msps-need-to-know-about-vibe-coding-agentic-ai-and-it-security [Accessed on 08/07/2025].

[G77] Wikipedia contributors. 2025. *Vibe coding.* https://en.wikipedia.org/wiki/Vibe_coding [Accessed on 02/07/2025].

[G78] Rhiannon Williams. 2025. *What is vibe coding, exactly?* https://www.technologyreview.com/2025/04/16/1115135/what-is-vibe-coding-exactly/ [Accessed on 07/07/2025].

[G79] Simon Willison. 2025. *Here's how I use LLMs to help me write code.* https://simonwillison.net/2025/Mar/11/using-llms-for-code/ [Accessed on 07/07/2025].

[G80] Simon Willison. 2025. *Not all AI-assisted programming is vibe coding (but vibe coding rocks).* https://simonwillison.net/2025/Mar/19/vibe-coding/ [Accessed on 02/07/2025].

[G81] Mark Wilson. 2025. *Vibe coding with AI.* https://node4.co.uk/blog/vibe-coding-with-ai/ [Accessed on 08/07/2025].

[G82] Angela Yang. 2025. *Noncoders are using AI to prompt their ideas into reality. They call it 'vibe coding.'.* https://www.nbcnews.com/tech/tech-news/noncoders-ai-prompt-ideas-vibe-coding-rcna205661 [Accessed on 07/07/2025].