Vibe Coding Applications for Industrial Safety

Michael Dorin[1], Juan M. Machuca De Pina[2]

[1]mike.dorin@stthomas.edu, [2]jmachuca@ulima.edu.pe

[1]School of Software Engineering and Data Science, University of St. Thomas, St. Paul, MN, USA
[2]Carrera de Ingeniería Industrial, Universidad de Lima, Perú

Received: August 15, 2025 / Accepted: September 19, 2025 / Published: 5 June, 2026
doi: https://doi.org/10.26439/ciii2025.8653

ABSTRACT—This paper explores the practicality of vibe coding for industrial engineering applications. Specifically, it investigates whether vibe coding can be utilized to create a safety-related application. In industrial environments, many occupations involve physical labor or the operation of potentially hazardous machinery, raising significant concerns regarding worker health and safety. Although various tools and applications exist to address these risks, they are often overly generic and may not adequately meet the specific requirements of individual organizations. The advent of artificial intelligence (AI) and machine learning has transformed this landscape. This study demonstrates how vibe coding can be used to develop a simple Android-based application that leverages built-in sensors—such as sound, light, and accelerometers—to monitor workers' physical activities and track environmental conditions for accident prevention. It is expected that this work will encourage companies engaged in physical labor to develop customized applications aimed at improving worker safety. Beyond this specific prototype, the proposed approach illustrates how AI-driven coding can bridge the gap between generic safety solutions and tailored workplace applications.

Index Terms—Application, artificial intelligence, industrial, safety, vibe coding.

I. Introduction

This research explores the practicality of applying the emerging programming paradigm known as vibe coding to industrial engineering applications [1]. This topic is essential because industrial institutions have specific needs that are not fully addressed by standard commercial off-the-shelf (COTS) software [2]. Given that worker safety is a critical priority across all industries, this research examines whether vibe coding can support the rapid development of customized safety-related applications tailored to specific operational environments.

Underscoring the importance of safety-related software applications, major academic databases such as Scopus, Web of Science, the Association for Computing Machinery (ACM), and Google Scholar have indexed thousands of publications on worker safety since 2024. In particular, Google Scholar reports more than 20.000 articles addressing vibration exposure, over 40.000 studies focused on sound-related risks, and more than 18.000 publications examining bright working conditions [3]. As demonstrated by Nouri et al., this literature search reveals that large language models (LLMs) are being used for safety analysis [4]. The literature review further indicates that numerous technological applications in occupational safety employ virtual reality or simulation-based training to reduce errors and prevent accidents [5]. Although these tools are beneficial during training stages, they cannot monitor the environment during actual operational activities. Zhang et al. have presented an insightful study on a smartphone-based vibration monitoring system [6]. Adverse working conditions can negatively affect worker performance and alertness throughout the workday. Exposure to certain conditions, such as vibrations, is known to cause, at the very least, discomfort or musculoskeletal issues [7]. In contrast, the presence of such technologies contributes to a safer work environment that supports operational excellence and sustainability [8]. Although this study does not specifically focus on worker safety, it demonstrates the potential of mobile devices for vibration management tasks. Feldbusch et al. presented a study on using a network of smartphones to monitor structural vibrations [9]. The authors also effectively demonstrate its applicability to safety-related applications.

Vibe coding is a technique in which software developers use LLMs to generate source code for a desired application. Interaction with LLMs is conducted through what is

commonly referred to as a prompt, which is an instruction or question provided by the user to elicit a specific response [10]. In essence, prompts are the mechanism through which users interact with LLMs. In pure vibe coding, software engineers provide a prompt describing the desired software and rely on the LLM to generate output that works as expected [11]. This application of artificial intelligence enhances the software development experience by automating programming minutiae, allowing developers to focus on more creative aspects of the application under development. While output quality depends on multiple factors, a crucial element is providing appropriate context within the prompt [12]. In the context of AI-driven vibe coding for safety applications, it can be assumed that domain expertise in safety is as valuable as programming skills.

Although vibe coding is a relatively recent phenomenon, researchers have been actively engaged and have published numerous papers on this topic as well. Google Scholar already suggests that more than 10.000 papers have been published [3]. Several authors discuss how vibe coding represents a paradigm shift, where developers focus on refining AI-generated code rather than writing it manually. This is substantially exemplified in an essay by Kreutzbender, which describes his personal experience with the concept [13]. Moore and Tatonetti describe how Vibe coding accelerates application development, reduces software creation cycle times, and reduces the reliance on highly specialized programming skills [14]. Benedetti, a self-described developer with over 40 years of experience, reports a twofold increase in productivity using Vibe coding [15]. Shukla et al. describe the use of LLMs in the pipeline from prototyping to production [16]. While there is considerable optimism regarding vibe coding, some concerns have also been reported. Some critics argue that vibe coding often produces code with insufficient structure, impairing maintainability, and may lead developers to spend more time fixing AI-generated issues than they save. Additional concerns include potential security risks associated with vibe coding, as AI tools may not consistently adhere to best security practices [17].

A number of papers examined the development of mobile applications for worker safety and the monitoring of vibrations. However, these studies do not address the integration of safety applications with vibe coding. This gap is significant: although safety applications and vibe coding have been investigated independently, no research has yet combined them to produce deployable prototypes. This study addresses that gap. Moreover, the concerns raised in previous works regarding vibe coding remain valid and must be considered. Accordingly, this article investigates the application of pure vibe coding to develop a safety application, demonstrating that such applications can be rapidly created for diverse environments.

Several LLMs are capable of performing vibe coding. A useful reference for available tools is the comprehensive list provided by DigitalOcean, which also outlines the strengths of each model [18]. A summary of this list is presented in Table I. Note that the DigitalOcean list is not exhaustive, as additional tools exist. This research explores vibe coding using Grok from xAI, based on the author's experience with the tool. Grok was selected because it is publicly available, features a conversational interface that facilitates rapid prototyping, and has demonstrated strong performance in software development tasks [19]. A Davis report also supports the choice, highlighting Grok AI's performance in software development and its ability to generate efficient code for complex scenarios [20].

TABLE I
LLMS AND TOOLS SUITABLE FOR VIBE CODING

Model/Tool	Key strengths
Gemini 2.5 Pro	Multimodal, large context
Claude (Sonnet/Opus/3.7/4)	High quality, long context
GPT-4/GPT-4º	Versatile, multimodal
Llama 3/CodeLlama	Open-source, local
DeepSeek Coder	Coding-optimized
GitHub Copilot	IDE integration
Cursor	Conversational editor
Claude Code (CLI)	Terminal agent
Warp Terminal	NL-to-code Shell
Lovable.com	Rapid prototyping

In this research, Grok is used to develop a full-stack application consisting of a Flask back-end and an Android front-end. The results are highly satisfactory. Organizations can adopt the template defined here to create custom safety-related applications.

II. Methodology

The steps outlined here are not overly complex; programmers are not required to possess advanced engineering skills. Basic programming knowledge and some understanding of software development are sufficient. As the saying goes, "A goal without a plan is just a wish" [21]. This holds true for vibe coding, where a foundational understanding of software development is necessary.

Although several software life cycle models exist, one that stands out for this project is the spiral life cycle. An essential aspect of the spiral life cycle model is risk management, making it a suitable choice for developing a safety-based application. Within the spiral model, the project progresses through multiple iterations, known as spirals, rather than reaching a final product in a single pass. Each spiral consists of several phases: planning, building, evaluating results, and planning the next iteration. If, at

the end of a spiral, risks remain unresolved or there is no clear strategy to address them, the spiral life cycle model requires terminating the project [22]. Spirals can be viewed as iterations of conventional life cycle models, with result evaluation and planning separating each. For instance, each spiral may represent an implementation of the waterfall life cycle model [23].

This study integrates the waterfall model within the spiral framework, as illustrated in Fig. 1. Accordingly, the research commenced with a standard waterfall software development life cycle, in which requirements are defined first, followed by code implementation, and finally testing [24]. These characteristics make the waterfall model well suited for integration with the spiral life cycle. Planning begins with requirements gathering, including the definition of user stories for this project. Each iteration through the spiral produces an improved version of the product.

A. Step 1: Define User Stories

As described in the Introduction, an effective prompt is critical for a vibe coding LLM to generate reliable source code. Explicit prompts are essential because they provide clear, detailed instructions to the AI, ensuring that the generated code aligns as closely as possible with the intended outcome. Accordingly, defining high-quality user stories is a key step in the process.

In this study, the authors initially defined four user stories without the aid of AI, following Ken Beck's user story format: As a [type of user], I would like to [do something] so that [benefit] [25]. The initial user stories are as follows:

- As a company safety supervisor, I would like to keep track of the vibration experienced by the operators of the machinery of the employees to ensure their health and safety.
- As a company safety supervisor, I would like to keep track of the audio levels experienced by the employees so that appropriate hearing protection can be provided.
- As a company safety supervisor, I would like historical data to be retained so that, in the event of an injury, we can determine how to prevent similar accidents in the future.
- As a safety supervisor, I would like reports displaying audio and vibration data so that trends can be effectively monitored.

B. Step 2: SMART User Stories

Within the waterfall life cycle model, once requirements have been defined and approved, the analysis phase begins [22]. To ensure the best possible prompt, the first step in the vibe coding process is to refine the user stories. To achieve this, the selected LLM, Grok, was tasked with converting the written stories into SMART user stories.

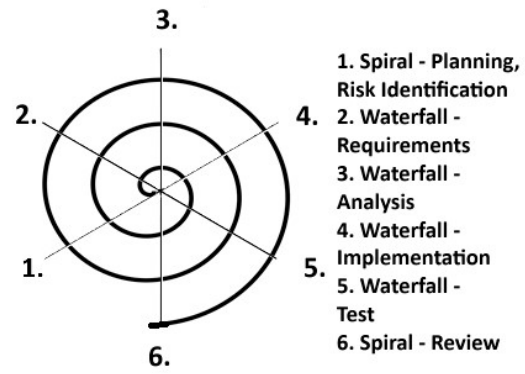

Fig. 1. Hybrid approach blending risk management (spiral) with clear deliverables (waterfall).

TABLE II
FUNCTION-LEVEL METRICS

NLOC	CCN	Function name	File
23	3	onCreate	MainActivity.kt
9	1	(anonymous)	MainActivity.kt
7	3	onResponse	MainActivity.kt
3	1	Run	SensorService.kt
4	1	Run	SensorService.kt
NLOC	CCN	Function name	File
17	2	onStartCommand	SensorService.kt
7	1	onDestroy	SensorService.kt
12	3	onSensorChanged	SensorService.kt
1	1	onAccuracyChanged	SensorService.kt
6	1	startAudioRecording	SensorService.kt
14	3	computeAndSendAudio	SensorService.kt
17	2	sendData	SensorService.kt
7	2	createNotificationChannel	SensorService.kt
10	1	onBind	SensorService.kt

TABLE III
ANDROID METHOD SUMMARY METRICS

NLOC	Avg. NLOC	Avg. CCN	Functions	File
76	13.0	2.3	3	MainActivity.kt
127	8.9	1.6	11	SensorService.kt

SMART is an acronym that guides the creation of well-defined user stories, standing for Specific, Measurable, Achievable, Relevant, and Time-bound. Applying the SMART criteria ensures user stories are precise, emphasizing well-defined outcomes and objectives. In a standard software engineering environment, SMART user stories

enhance team collaboration by providing a clearer understanding of software requirements. For this application, this refinement step ensures that both human and AI share precise expectations, reducing ambiguity and improving the quality of the generated application. Consequently, in vibe coding, the goal is for both the human and the LLM to have a precise understanding of what should be created. Ultimately, SMART user stories contribute to the development of a high-quality prompt. Accordingly, continuing the vibe coding process, human-authored user stories were converted into SMART user stories. The SMART user stories generated by Grok are presented below.

- As a company safety supervisor, I want to monitor vibration levels experienced by machinery operators in real time using sensors that measure vibrations in g-forces with 0.1 g accuracy, integrated with existing machinery, relevant to preventing musculoskeletal disorders, and updated every 5 min during shifts, so that health and safety are ensured by triggering alerts when levels exceed OSHA thresholds.
- As a company safety supervisor, I want to monitor audio levels experienced by employees using decibel meters with 1 dB accuracy via wearable devices synced to a central dashboard, relevant to complying with noise exposure regulations, and monitored continuously during work hours, so that appropriate hearing protection is provided when levels exceed 85 dB for more than 8 h.
- As a company safety supervisor, I want to store historical data on vibrations, audio levels, and related incidents in a secure database for at least 5 years using cloud storage with automated backups, relevant to root cause analysis for injury prevention, and accessible within 24 h of a request, so that if injuries occur, data-driven insights help protect employees from future accidents.
- As a company safety supervisor, I want customizable reports that display audio and vibration trends over selectable time periods (e.g., daily, weekly, monthly), generated through a reporting tool integrated with the data system and presented as charts and graphs, relevant to proactive safety monitoring, and delivered on a scheduled basis via email or dashboard (e.g., weekly), so that emerging trends can be identified and addressed in a timely manner.

C. Step 3: Generate the Code

In the standard waterfall life-cycle model, coding follows the analysis phase. This step also naturally aligns with the vibe coding approach.

The SMART user stories were implemented through Grok's public user interface to generate a new prompt. This prompt requests the development of a full-stack application comprising a Flask-based back end [26] and a native Android front end [27]. The Grok user interface is shown in Fig. 2.

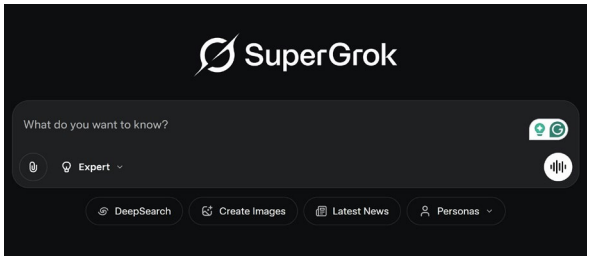

Fig. 2. Grok user interface.

For the Android application, Grok generated Kotlin source code consisting of two main classes and a limited set of methods. Similarly, the Flask application, implemented in Python, was concise and lightweight.

D. Step 4: Testing

The final step prior to deployment in the waterfall model is testing. This phase also aligns with the evaluation stage of the spiral model. During testing, the generated application was deployed on an Android device, and the Python-based back end was launched on a server.

III. Results

At the conclusion of each spiral, the process includes evaluating the results and planning the subsequent iteration. Grok generated two projects: an Android application implemented in Kotlin and a Python-based Flask application. The Android application comprised two classes only—MainActivity and SensorService—with the latter responsible for device sensor monitoring. These components are illustrated in Fig. 3. The Android application was built without syntax errors and successfully launched, as illustrated in Fig. 4. Although the application initially executed without errors, the Android manifest required modification to enable foreground sound monitoring. In addition, the manifest was configured to permit clear-text data transmission for testing purposes, which incidentally highlights a potential security concern. Grok was also tasked with recommending the appropriate manifest modifications.

Upon startup, the Android application displays an interface that allows the entry of an employee identifier and provides a control to initiate monitoring. Once data are available, an additional option is presented to retrieve a report. The monitoring control is not specific to sound or vibration; instead, both sensors are activated simultaneously when the control is triggered. This monitoring supports the requirements established in the user stories; the Android application can monitor sound and vibration.

The ability to retrieve historical data demonstrates support for the reporting requirement. These results illustrate that an operational prototype can be developed within a single development cycle, a task that would typically require significantly more time, effort, and expertise.

Before deployment, two key issues must be addressed: the absence of a stop button and the report presentation format. While continuous monitoring is essential, a stop button is required to allow users to pause data collection and easily retrieve reports. In addition, the current report display on the Android device is too small to be legible. Addressing these issues will improve usability and provide a more effective user experience.

The Flask application generated by Grok was deployed and executed on a Windows system without major issues. The application exposes a RESTful interface for storing audio and vibration data, as well as for retrieving report information. The RESTful endpoints were tested using a PowerShell script based on Invoke-WebRequest commands with the -Method POST option to submit audio and vibration data.

Invoke-WebRequest commands with the -Method GET option were used to retrieve report data. Initial verification and validation testing revealed no apparent errors in the Flask application.

The benefit of the spiral life cycle model is that it addresses risks. The introduction defined two risks. The first risk was that poor-quality code would be generated. Tables IV and Table V show the lines of code per function, as well as the cyclomatic complexity (CCN). While discussing software metrics is not the primary focus of this paper, none of the generated functions exhibit a CCN greater than seven. Cyclomatic complexity values in the range of 10–15 are generally considered problematic and typically require code restructuring [28]. Furthermore, the overall architecture, as illustrated in the class diagrams, does not exhibit excessive code volume or undue complexity.

The second identified risk relates to security. It can be argued that Grok did not generate any code addressing protection or security mechanisms. As the saying goes, computers do not do what is intended, but rather what they are instructed to do; this principle applies equally to vibe coding. In this case, no security requirements were specified in the original user stories, nor was there any indication that security considerations were expected. Consequently, Grok generated code strictly based on the requirements provided.

With this in mind, as part of the planning for the second spiral, Grok was prompted to generate three additional SMART user stories. These include one addressing authentication, one focused on data security, and another covering the missing stop button in the Android user interface. These results are shown below.

- As a company safety supervisor, I want to implement robust encryption for all stored and transmitted

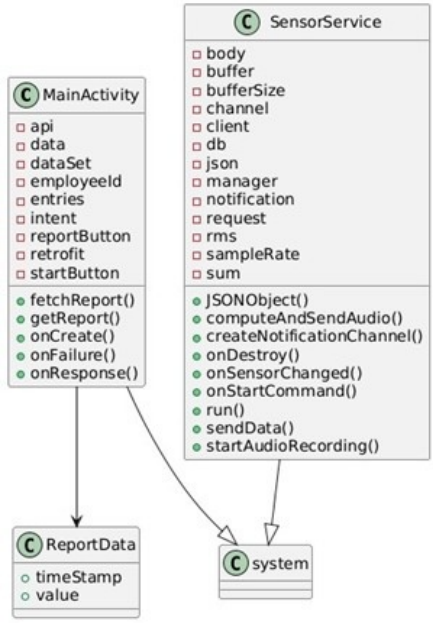

Fig. 3. Android application class diagram.

Fig. 4. Android application showing main menu.

TABLE IV
FUNCTION-LEVEL METRICS

NLOC	CCN	Function name	File
11	2	add vibration	app.py
11	2	add audio	app.py
19	7	get report	app.py

vibration and audio-level data using the AES-256 standard. This can be achieved through integration with secure cloud services such as AWS Key Management Service (KMS), ensuring the protection of sensitive employee health information against unauthorized access. Compliance with data protection regulations, including the Health Insurance Portability and Accountability Act (HIPAA), will be verified through annual penetration testing, thereby safeguarding employee privacy.

- As a company safety supervisor, I want to enforce multifactor authentication (MFA) for all application user logins using methods such as authenticator applications or SMS-based verification. This will be implemented through integration with identity providers such as Okta or Azure Active Directory (Azure AD) to prevent unauthorized access to sensitive health and safety data. MFA will be required for every user session, with automatic logout enforced after 15 min of inactivity, thereby ensuring that only authorized personnel can access the system and that employee data remains protected against potential security breaches.
- As a company safety supervisor, I want to provide a prominent stop/start control at the top of the user interface to manually enable or disable real-time monitoring of vibration and audio levels. This functionality shall be implemented through intuitive user interface controls integrated into the application dashboard, allowing flexible operation during work shifts or maintenance activities. The control shall respond within 1 s of user activation, ensuring that monitoring can be paused or resumed as needed without disrupting operations or data integrity.

The complete set of user stories was subsequently consolidated into a new prompt, which was provided to Grok to generate a next-generation version of the code. This updated version incorporated the required user interface elements, as well as enhanced authentication and security features. The following changes were implemented in the new code build.

- Implement AES-256 encryption for vibration and audio data transmission via HTTPS/TLS 1.3, using libraries such as OpenSSL or Bouncy Castle in backend and Android app.
- Integrate key management, encrypting data before cloud storage and decrypting on retrieval.
- Add automatic logout after 15 min of inactivity using idle timers and session logic in Android app/backend, with secure token storage.
- Add prominent start/stop button in Android app UI (e.g., Button in Toolbar/AppBar) with event listeners to toggle vibration/audio monitoring.

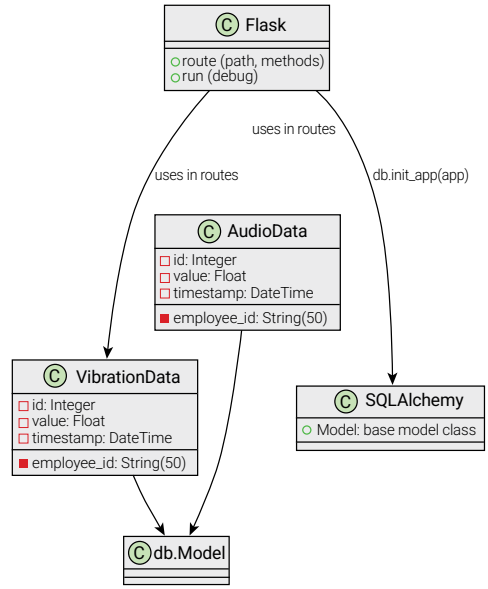

Fig. 5. Flask application class diagram.

TABLE V

FILE-LEVEL SUMMARY METRICS (TWO FILES ANALYZED)

NLOC	Avg. NLOC	Avg. CCN	Function count	File
60	13.7	3.7	3	app.py
13	0.0	0.0	0	models.py

IV. DISCUSSIONS

A. Productivity Benefits

This study demonstrates that vibe coding can be effectively applied to the generation of software applications, including those intended for industrial safety contexts. The Android and Flask applications developed were syntactically correct and closely aligned with the requirements specified by the SMART user stories. This result is significant, as the development of safety-critical applications has traditionally required substantial development effort. The proposed approach leverages AI-driven code generation, indicating that individuals or small teams can achieve high levels of productivity while contributing to operational excellence.

B. Limitations

The successful generation of these applications motivates further research in this area. At a minimum, vibe-coded applications require extensive testing and rigorous review prior to deployment in safety-critical environments. Several

initial implementation issues must also be addressed. For example, the omission of fundamental security features highlights the importance of carefully designed prompts and the need for domain expertise to effectively guide the vibe coding process. The initial results indicate that, while vibe coding significantly increases developer productivity, it does not eliminate the need for thorough verification and validation. Oversight by qualified professionals therefore remains essential.

Future work should focus on expanded testing under realistic operating conditions, the systematic incorporation of user feedback, and the evaluation of alternative LLMs. These efforts should be aligned with applicable safety regulations issued by government agencies, such as the Occupational Safety and Health Administration (OSHA) in the United States [29], as well as with data protection frameworks, including the General Data Protection Regulation (GDPR) of the European Union [30]. In addition, compliance with the HIPAA, which governs the privacy and security of protected health information in the United States [31], should be ensured. Such regulatory alignment is essential to guarantee compliance and to foster trust among workers.

V. Conclusions

The purpose of this investigation was to determine whether vibe coding could be applied to the development of safety-oriented applications in an industrial environment. The study demonstrated that functional prototypes—specifically, an Android monitoring application and a Flask-based backend—can be rapidly generated using a LLM.

As part of this project, initial user stories were authored by humans. The Grok AI tool was then tasked with transforming them into SMART user stories and generating code for both a Flask-based backend and an Android application. Although some initial tuning was required to obtain a fully functional Android application, Vibe Coding successfully produced code that conformed to the tasks specified in the user stories. Overall, the results clearly indicate that vibe coding significantly increases development productivity. Vibe coding enables small teams—or even individual developers—to create powerful, tailored applications, thereby providing organizations with enhanced tools suited to their specific operational environments. However, despite the significant productivity gains offered by vibe coding, its use should be accompanied by continuous expert human oversight to ensure correctness, safety, and maintainability. Ultimately, vibe coding signifies a significant shift that enables organizations to use their developers more efficiently. More research is needed to enhance the effectiveness of this technique. In addition, the authors of this work envision that similar methods could be applied beyond industrial safety —for example, in specialized healthcare monitoring or logistics—demonstrating the broader generalizability of vibe coding for the development of safety-critical systems.

Acknowledgments

The authors acknowledge the use of AI-based tools—specifically Grammarly for grammar and language refinement, and Grok for user story and code generation.

References

[1] A. Sarkar and I. Drosos, "Vibe coding: Programming through conversation with artificial intelligence," 2025, *arXiv:2506.23253*.

[2] J. Beheshti and J. Dupuis, "Problems with COTS software: A case study," *Proc. Annu. Conf. CAIS Actes du congrès annuel de l'ACSI*, no. 2000, Oct. 2013, doi: https://doi.org/10.29173/cais7

[3] Google. "Google Scholar." Google. Accessed: Aug. 5, 2025. [Online]. Available: https://scholar.google.com

[4] A. Nouri, B. Cabrero-Daniel, F. Torner, H. Sivencrona, and C. Berger, "Welcome your new AI teammate: On safety analysis by leashing large language models," in *Proc. IEEE/ACM 3rd Int. Conf. AI Eng.–Softw. Eng. for AI (CAIN)*, Lisbon, Portugal, Jun. 2024, pp. 172–177, doi: https://doi.org/10.1145/3644815.3644953

[5] O. Flor-Unda *et al.*, "Innovative technologies for occupational health and safety: A scoping review," *Safety*, vol. 9, no. 2, p. 35, May 2023, doi: https://doi.org/10.3390/safety9020035

[6] D. Zhang, J. Tian, and H. Li, "Design and validation of Android smartphone-based wireless structural vibration monitoring system," *Sensors*, vol. 20, no. 17, p. 4799, Aug. 2020, doi: https://doi.org/10.3390/s20174799

[7] B. Halmai, T. P. Holsgrove, S. J. Vine, D. J. Harris, and G. K. Williams, "The after-effects of occupational whole-body vibration on human cognitive, visual, and motor function: A systematic review," *Appl. Ergon.*, vol. 118, p. 104264, Apr. 2024, doi: https://doi.org/10.1016/j.apergo.2024.104264

[8] V. Tripathi and A. Bhadauria, "Leveraging the competency of condition monitoring system for achieving excellence in operations management," *Discov. Sustain.*, vol. 6, p. 452, May 2025, doi: https://doi.org/10.1007/s43621-025-01285-8

[9] A. Feldbusch, H. Sadegh-Azar, and P. Agne, "Vibration analysis using mobile devices (smartphones or tablets)," *Procedia Eng.*, vol. 199, pp. 2790–2795, 2017, doi: https://doi.org/10.1016/j.proeng.2017.09.543

[10] T. B. Brown *et al.*, "Language models are few-shot learners," 2020, *arXiv:2005.14165*.

[11] A. Smeyatsky, "A comprehensive guide to vibe coding and context engineering: Unlocking AI superpowers by August 2025," LinkedIn Pulse, Accessed: Aug. 14, 2025. [Online]. Available: https://www.linkedin.com/pulse/comprehensive-guide-vibe-coding-context-engineering-ai-smeyatsky-emtnf

[12] S. R. Cavalcante, E. R. Ribeiro, and A. C. Oran, "The impact of AI tools on software development: A case study with GitHub Copilot and other AI assistants," in *Proc. 27th Int. Conf. Enterp. Inf. Syst. (ICEIS)*, Porto, Portugal, Apr. 2025, pp. 245–252, doi: https://doi.org/10.5220/0013294700003929

[13] J. Kreutzbender, "My thoughts and experiences with vibe coding (mid 2025)," Jeremy Kreutzbender, May 2025. [Online]. Available: https://jeremykreutzbender.com/blog/thoughts-and-experiences-vibe-coding-mid-2025

[14] J. H. Moore and N. Tatonetti, "Vibe coding: A new paradigm for biomedical software development," *BioData Min.*, vol. 18, no. 1, Art. no. 46, Jul. 2025, doi: https://doi.org/10.1186/s13040-025-00462-9

[15] M. Benedetti, "Vibe coding as a coding veteran. From 8-bit assembly to English-as-code," Level Up Coding, Aug. 2025. [Online]. Available: https://levelup.gitconnected.com/vibe-coding-as-a-coding-veteran-cd370fe2be50

[16] A. Shukla, J. K. Vijay, U. Sen, and J. Jain, "From prototyping to production: LLM chains carrying the software development pipeline," *Int. J. Emerging Technol. Appl.*, vol. 11, no. 3, pp. 193–200, Jun. 2024. [Online]. Available: https://www.ijetajournal.org/volume-11/issue-3/IJETA-V11I3P36.pdf

[17] M. Gupta, "Don't be a vibe coder. Problems with vibe coding," Medium, Mar, 2025. [Online]. Available: https://medium.com/datascience-in-your-pocket/dont-be-a-vibe-coder-30fa7c525971

[18] DigitalOcean, "10 best vibe coding tools: LLM-powered code generators to try," DigitalOcean, 2025. [Online]. Available: https://www.digitalocean.com/resources/articles/vibe-coding-tools

[19] xAI, "Grok," xAI, 2025. [Online]. Available: https://grok.x.ai

[20] D. Davies, "Code generation and debugging with the Grok 4 API," Weights & Biases, Jul. 2024, Accessed: Aug. 30, 2025. [Online]. Available: https://wandb.ai/onlineinference/genai-research/reports/Code-generation-and-debugging-with-the-Grok-4-API--VmlldzoxMzUzOTUwOA

[21] A. de SaintExupéry, *The Little Prince*, 1st ed. New York, NY, USA: Reynal & Hitchcock, 1943.

[22] R. Sethi, *Software Engineering: Basic Principles and Best Practices*, 1st ed. Cambridge, UK: Cambridge University Press, 2023.

[23] S. K. Pal, "What is spiral model in software engineering?," GeeksforGeeks, Jul. 2025. [Online]. Available: https://www.geeksforgeeks.org/software-engineering/software-engineering-spiral-model/

[24] S. R. Schach, *Classical and ObjectOriented Software Engineering with UML and Java*, 4th ed. Boston, MA, USA: WCB/McGrawHill, 1999.

[25] K. Beck and M. Fowler, *Planning Extreme Programming*, 1st ed. Boston, MA, USA: Addison-Wesley Professional, 2000.

[26] Pallets Projects, "Welcome to Flask — Flask Documentation (3.1.x)," Flask, Accessed: 2025. [Online]. Available: https://flask.palletsprojects.com/

[27] Google, "Google scholar," accessed: 2020-10-20. [Online]. Available: http://scholar.google.com/

[28] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, Dec. 1976, doi: https://doi.org/10.1109/TSE.1976.233837

[29] United States Congress. 91st Congress, 2nd Session. (1970, Dec. 29). *Pub. L. 91596, OSH Act of 1970*. [Online]. Available: https://www.osha.gov/laws-regs/oshact/completeoshact/

[30] European Union. (2016, Apr. 27). *Regulation (EU) 2016/679 of the European Parliament and of the Council, General Data Protection Regulation (GDPR)*. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[31] United States Congress. 104th Congress, 2nd Session. (1996, Aug. 21). *Pub. L. 104191, Health Insurance Portability and Accountability Act of 1996*. [Online]. Available: https://www.govinfo.gov/link/plaw/104/public/191