Computers & Security 164 (2026) 104842

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TC 11 Briefing Papers

Insecure by design? A human-centric security perspective on AI-assisted software development

Magdalena Glas [a,*], Christoph Nirschl [a], Bar Lanyado [b], Johan van Niekerk [c]

[a] University of Regensburg Universitätsstr. 31, Regensburg, 93047, Germany
[b] Lasso, Beit Shamai St 10, Tel Aviv, 6701838, Israel
[c] Noroff University College Tordenskjolds gate 9, Kristiansand, 4612, Norway

ARTICLE INFO

Keywords:
Artificial intelligence
Software development
AI-assistance
Security
Coding

ABSTRACT

Generative artificial intelligence (AI) tools are increasingly used in software development, improving the efficiency of software developers. However, this adoption introduces notable security challenges. AI/generated code is not secure by default, as it is often based on large-scale training data that includes open-source code of varying quality and trustworthiness. Developers using these tools may be unaware of the associated risks or may place excessive trust in the security of the output. This briefing paper outlines the key security risks associated with generative AI and offers human-centered strategies for mitigation. Since these risks arise not only from how generative AI models are built but also from how humans interact with them, we adopt a human-centric perspective. To this end, we provide recommendations for individuals, organizations, and educators to help harness the potential of generative AI in software development while effectively managing the associated security risks.

1. Introduction

Developers increasingly use AI assistants like ChatGPT or GitHub Copilot in software development for tasks like generating, debugging, and documenting code. These assistants have great potential to make software development faster (Moradi Dakhel et al., 2023; GitHub, 2024), less tiring (Klemmer et al., 2024; Kazemitabaar et al., 2023), and more attainable to novices (Perry et al., 2023). By offloading routine coding tasks to generative AI, developers can focus more on higher-level design and problem-solving. Since the AI can turn natural language prompts into executable code, developers may no longer need prior programming knowledge or an understanding of syntax to produce working solutions. However, the potential that AI assistants offer comes with inherent security risks. AI/generated code may include insecure practices or vulnerabilities that users might overlook, especially if they rely too heavily on the tool without critically evaluating its output. Seasoned developers are generally more aware of these risks, as they have the knowledge and experience to spot insecure code or question the suggestions provided by AI tools (Klemmer et al., 2024). In contrast, novices and less experienced programmers might lack the skills or confidence to assess the security implications of AI/generated code. Ultimately, both seasoned developers and beginners need to consider security when using AI assistants for coding. The goal is to enable everyone to benefit from the advantages AI brings to software development while still ensuring that the code they produce is secure.

The capabilities of AI coding assistants like ChatGPT are grounded in the technological evolution of transformer-based Large Language Models (LLMs), first introduced by Vaswani et al. in 2017 (Vaswani et al., 2017). The transformer architecture replaced earlier sequential models such as LSTMs with a parallelized attention mechanism, allowing models to scale dramatically in both size and performance (Han et al., 2021). OpenAI's GPT series exemplifies this progression, growing from GPT-1 to the much larger GPT-4 and GPT-4o, with increasing ability to perform few-shot and zero-shot learning (Brown et al., 2020). These models are pre-trained on vast corpora of text and code, such as public GitHub repositories, technical documentation, and Q&A forums, allowing them to learn statistical associations between language tokens, including programming constructs.

While the current generation of transformer-based LLMs marks a major leap, early AI models, such as Hidden Markov Models, were capable of generating sequential data, although they were limited in scope (Wang et al., 2024). Significant progress in AI, particularly in Natural Language Processing (NLP), was marked by the advent of deep learning models like RNNs, LSTMs, and GRUs, enabling the generation of longer text content (Wang et al., 2024). Recently, the landscape of software development has been dramatically transformed by the rapid

* Corresponding author.
E-mail addresses: magdalena.glas@informatik.uni-regensburg.de (M. Glas), christoph.nirschl@ur.de (C. Nirschl), blanyado@gmail.com (B. Lanyado), johan.vanniekerk@noroff.no (J. van Niekerk).

https://doi.org/10.1016/j.cose.2026.104842
Received 19 January 2026; Accepted 20 January 2026
Available online 23 January 2026

advances in AI models. These models, often based on the attention-based transformer architecture, became widely known and available through pre-trained models like *OpenAI's GPT series*, *Codex*, *Google's PaLM*, *Meta's LLaMA*, and *Anthropic's Claude*.

The AI assistants built upon these underlying machine learning models have emerged as powerful tools intended to aid in the software development lifecycle and improve developer productivity. Prominent examples include *GitHub Copilot*, *ChatGPT*, *Google Gemini*, and *Microsoft Copilot*. To create code, the underlying models rely on large datasets of publicly available code, sourced from platforms like GitHub. They operate by taking text prompts, such as function definitions or natural language descriptions, and generating corresponding code outputs. AI assistants constitute a novel code inclusion path, distinct from traditional mechanisms like importing or forking dependencies. As AI assistants become increasingly integrated into software development workflows, they offer a range of benefits across the entire development process. Developers are using these tools to support a variety of tasks throughout the software development lifecycle, including code generation, testing, debugging, review, and documentation. By automating repetitive and time-consuming activities, AI allows developers to focus more on complex, higher-level challenges such as architecture and design. This shift not only accelerates development timelines (Barke et al., 2023), but also contributes to improved code quality and fewer errors (GitHub, 2023, 2024). Reflecting these advantages, adoption has become widespread. Recent surveys show that well over 90% of professional developers use AI-assisted tools in their workflows, with the majority recognizing clear benefits, demonstrating a strong and growing level of trust in AI-/generated code (GitHub, 2023, 2024).

Furthermore, AI assistants lower the entry barrier for novice programmers. As AI assistants can generate code directly from natural language, users do not necessarily need to know the syntax of a programming language. As such, individuals who may lack formal training in programming languages are suddenly able to create functioning code (Peng et al., 2023; Cui et al., 2025; Haindl and Weinberger, 2024). Beyond code generation, AI assistants also provide contextual suggestions, clarify programming concepts, and flag potential errors, enabling novices to learn as they code and gradually build their skills through guided feedback. Over time, many novices report lasting improvements in their skills, indicating a durable, domain-specific learning effect. This aligns with broader research suggesting that regular use of AI assistants can support learning and skill development in programming contexts (GitHub, 2023; Liang et al., 2024).

Despite these advantages, this widespread use introduces new concerns, especially regarding security. Using AI assistants in software development carries the risk of creating a misleading sense of competence, especially in inexperienced users, as successful outcomes may reinforce a misleading perception of their own skills. This, in turn, may lead to an uncritical acceptance of AI/generated suggestions and reduce emphasis on critical quality assurance practices (Wang et al., 2023), such as thorough testing and peer code review. Coupled with growing trust in and (over-)reliance on AI assistants, this can lead to unintentionally introducing security vulnerabilities (Becker et al., 2023). In the following, this briefing paper outlines key security risks in AI-assisted software development and provides recommendations for individuals, organizations, and educators to address these risks.

2. Security risks of AI-assisted software development

We identified three key problems that contribute to security risks in AI-assisted software development. These are (1) the tendency of AI assistants to generate insecure code, (2) developers' uncritical use of such code, and (3) the absence of methods or benchmarks to assess its security. The three problems are outlined in the following.

Problem 1: AI assistants do not necessarily produce secure code. The pre-trained nature of AI models introduces significant limitations when it comes to the security of the created code. The quality, diversity, and representativeness of the training corpus directly affect the model's performance. This training data, often pulled from open-source platforms like GitHub or StackOverflow, includes code of varying quality, and importantly, it is not curated for security (Perry et al., 2023; Fu et al., 2023). As a result, these models can learn and reproduce insecure coding patterns, even if the generated code is fully functional and appears correct at first glance (OWASP Foundation, 2024). Fig. 1 shows an example of such a vulnerable code when advising an AI assistant to generate code that allows users to upload files to a website. The implementation operates correctly. However, from a security standpoint it fails to enforce upload sanitization and restrictions, thereby permitting directory traversal, unintended file overwriting, and the submission of executable (and potentially malicious) files.

The security of AI/generated code depends on multiple factors. One key factor is the specific AI model being used. Different models may produce more or less secure code depending on the quality of the training data and the way the model was fine-tuned (Bhatt et al., 2023; Tihanyi et al., 2024). Second, there are clear differences between programming languages (Ambati et al., 2024; Fu et al., 2023). Languages like Python or JavaScript, may have better community practices around security, while others, especially hardware-related languages like Verilog, are more prone to insecure output (Pearce et al., 2023; Nair et al., 2023). In some cases, the vulnerabilities are not obvious in the code itself but become critical once the code is integrated into a larger system (Sandoval et al., 2023).

Even when the model generates syntactically sound code, the inclusion of external libraries or tools can introduce further vulnerabilities. AI models often suggest the inclusion of packages to solve a given task. However, these suggestions are not always reliable. In some cases, the assistant may hallucinate packages that do not exist at all (Lanyado, 2024). In other cases, it might recommend outdated or vulnerable libraries (Ji et al., 2024; Synk, 2023). These insecure dependencies introduce risks that developers may not detect, especially if they trust the AI output without further validation.

Overall, while AI tools can speed up coding and help with complex tasks, they do not guarantee secure results. Without proper filtering, validation, and review, AI/generated code can quietly introduce security flaws that are hard to detect later in the development process.

Problem 2: Developers often use AI/generated code without full awareness of security risks. Beyond the technical limitations of AI models, a significant risk arises from how developers adopt and use AI-/generated code, often without fully considering potential security implications (Hamer et al., 2024). This behavior continues a long-standing trend in software development, relying on ready-made code snippets from platforms like StackOverflow without fully understanding their implications (Fischer et al., 2017). However, the use of AI assistants increases this risk. Because AI assistants provide tailored, context-specific code on demand, they create a sense of efficiency and reliability that can lead users to assume the code is safe. As long as it compiles and produces the expected result, developers often accept it without considering whether it introduces security issues.

This risk is especially pronounced among less experienced developers. Many are capable of generating working solutions with the help of AI, but lack the security knowledge to assess whether the code is actually secure. In these cases, functionality becomes the primary indicator of quality: if the code works, it is considered "good enough". Security checks are skipped, not necessarily out of carelessness, but because developers may not realize what to look for or do not know that a vulnerability could be present (Ji et al., 2024).

Even experienced developers are not immune to this behavior. Although they may be aware of secure coding practices, they are still at risk of overtrusting AI/generated code. Prior studies have shown that developers often assume the AI "knows better", which can lead to misplaced confidence in its output (Perry et al., 2023). Developers perceive AI/generated code as more secure or higher quality than human-written code, simply because it comes from a seemingly intelligent

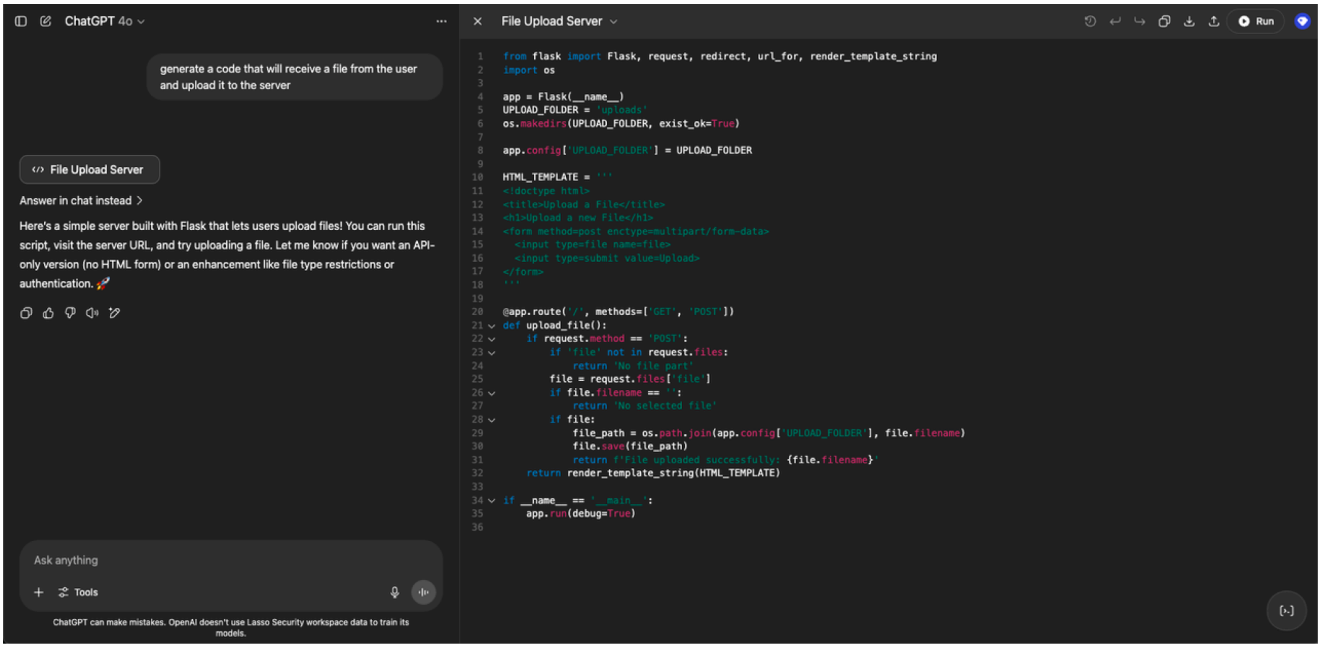

Fig. 1. Code generated by ChatGPT (GPT model 4o) that implements a file upload functionality. While the code is functioning correctly, it is prone to several security vulnerabilities. It allows 1) file traversal, 2) overwriting of other files and 3) unvalidated uploads of potentially malicious executables due to insufficient file-type restrictions and sanitization.

system (Synk, 2023; Perry et al., 2023). This perception reduces the likelihood of questioning the results.

On top of that, developers also tend to overestimate their ability to review and evaluate AI/generated suggestions. In interviews, developers reported high confidence in their capacity to catch mistakes or security flaws in AI output, yet their actual performance in identifying risks did not match this confidence (Klemmer et al., 2024). This gap between perceived and actual oversight contributes to the silent spread of insecure code, even in teams that are otherwise security-aware.

In short, the ease and speed of using AI/generated code can overshadow security concerns, both for novices and experienced developers. Without targeted interventions or changes in development practices, insecure code may continue to be introduced simply because it appears to work.

Problem 3: Lack of methods and benchmarks to assess security of AI/generated code. Even if developers become more aware of security concerns, a critical barrier remains: there are few effective methods to systematically evaluate whether AI/generated code is actually secure. Traditional vulnerability handling processes, such as manual code reviews and static analysis, face major challenges when applied to AI-/generated code (Kaniewski et al., 2024). These tools and workflows were built around the assumption that code is written by humans and follows common coding patterns. However, AI/generated code can introduce vulnerabilities that are semantically identical to known issues but look syntactically different, making them harder to detect using conventional tools (Kaniewski et al., 2024). As a result, known weaknesses may go undetected because they evade the pattern-matching logic of traditional tools.

In addition to these technical challenges, the volume and speed of AI/generated code can overwhelm manual review processes. Security analysts and reviewers may struggle to keep up with the output of AI tools, especially in fast-paced development environments (Cotroneo et al., 2025). This further increases the risk that insecure code slips through without proper validation.

Another limitation lies in the evaluation itself. While there are many benchmarks for assessing the performance of AI/generated code, there are few widely accepted benchmarks focused specifically on security (Ji et al., 2024). Some research efforts have proposed new ways to test AI tools for vulnerability introduction, but these security-focused benchmarks have not yet seen broad adoption within the AI and machine learning communities (Ji et al., 2024; Pearce et al., 2025). As a result, it is difficult to systematically compare or improve the security of different models and tools. What further complicates this evaluation is that LLMs remain fundamentally non-deterministic. The same prompt can lead to different outputs depending on random sampling parameters, making it difficult to reproduce or benchmark model behavior consistently. This unpredictability poses a significant challenge for security validation, where deterministic behavior is essential to detect, compare, and mitigate vulnerabilities. LLMs generate code by predicting a *reasonable continuation* of the text so far (Wolfram, 2023), using probabilistic sampling to determine the next most likely word or token [(Wolfram, 2023), p.9]. The level of randomness is influenced by the *temperature* parameter, which controls the likelihood of selecting less probable words or structures (Wolfram, 2023). As a result, even small variations in generation parameters, or simply repeating the same prompt, can yield different outputs. While this flexibility can be useful in creative tasks, it creates friction for secure software development, where consistency and reproducibility are critical. Security tools and review processes depend on being able to trace and compare versions, but this becomes harder when outputs shift unpredictably. Worse, these models often appear confident in their answers, even when they are wrong (Han et al., 2021), and they lack awareness of what they do not know (Vaswani et al., 2017).

Beyond technical issues, many organizations have not yet established clear policies or responsibilities when it comes to AI-assisted development. There is often no formal process for handling security risks introduced by AI tools, and responsibilities for reviewing or auditing AI/generated code remain unclear (Klemmer et al., 2024). Even when security tools are available, they are not always adapted to detect patterns specific to AI/generated code (Pearce et al., 2025). This leaves developers without adequate support to evaluate or improve the security of the code they are using.

3. Practical recommendations

Together, these gaps, technical, procedural, and organizational, make it difficult to reliably detect and manage vulnerabilities in AI-/generated code. The non-deterministic behavior of AI models and the complexity of modern software infrastructures make these challenges not solvable through technical solutions alone. Human awareness, oversight, and accountability remain essential components of any secure development process. This paper focuses on the human perspective of secure AI-assisted software development, aiming to support developers, organizations, and educators in integrating AI tools responsibly and effectively into their workflows. To this end, the following section presents actionable recommendations for promoting secure and mindful use of generative AI in coding practice.

3.1. Recommendations for individuals

We emphasize that every individual using AI-assisted coding tools must approach AI/generated code with security in mind. This includes thoughtful prompt design and careful handling of outputs, both of which rely on a broader awareness of AI-related security risks, as illustrated in Fig. 2. In the following, we describe these recommendations in detail.

3.1.1. Awareness in AI-assisted coding

As AI-assisted coding continues to evolve, its applications and use cases are rapidly shifting. Current deployments already reveal security weaknesses that vary by task and context (Majdinasab et al., 2023; Tony et al., 2025). In the absence of universally applicable best practices, secure and responsible use requires a combination of awareness, continuous learning, and critical judgment. We recommend the following considerations to support informed and responsible use of AI-assisted coding tools:

Understand the limitations of AI tools. Responsible and secure interaction furthermore requires a solid understanding of these tools, their limitations, and how to apply them mindfully, especially in contexts involving sensitive information (Security, 0000). AI models trained primarily on smaller, self-contained code snippets may perform well on isolated functions or simple algorithmic tasks but struggle with maintaining coherence across larger codebases, managing interdependent modules, or understanding architectural intent. These tasks typically require reasoning over multiple files, maintaining global state, or preserving logic over long sequences, capabilities that exceed the effective context window or pattern-matching ability of many current models (Bubeck et al., 2023).

Stay informed about evolving threats and best practices.

Despite the novel character of AI tools, established best practices for secure software development still remain relevant. However, with ongoing development and shifting applications and risks, today's secure practice recommendations may become outdated over time (Pearce et al., 2025). Users should therefore remain adaptable and anticipate changes in both security recommendations and tools capabilities. Following current security resources, such as the NIST AI security framework (Tabassi, 2023) or the OWASP Top 10 for LLM Applications (OWASP Foundation, 2024), and remaining informed about newly identified AI-specific threats ensures responsible and secure utilization (Klemmer et al., 2024).

3.1.2. Secure prompting and input handling

To reduce the likelihood of introducing vulnerabilities, developers need to craft prompts thoughtfully and handle inputs with care. While AI assistants can generate code that appears functional, their behavior is highly dependent on the prompts they receive and the context they are given. Secure prompting involves not only asking the right questions but also structuring those questions in a way that promotes secure and predictable code generation. The following practices can help improve the quality and security of AI/generated code:

Avoid vague or unspecific prompts and use explicit security instructions. Vague requests, such as *"optimize our database"* (Pawel and Arkadiusz, 2025), not only leave great room for (mis)-interpretation but also increase the likelihood of introducing security vulnerabilities. Prompts that focus on small, reviewable tasks (Klemmer et al., 2024), reinforced with clear, detailed, and security-oriented instructions such as *"sanitize user input"* or *"avoid hard-coded secrets"* (Nair et al., 2023; White et al., 2023; Tony et al., 2025), help to explicitly guide the tool to follow secure coding practices. AI models tend to perform significantly better on short, localized tasks than on broad, context-heavy ones (Pearce et al., 2023). Splitting complex tasks into smaller, well-defined prompts is an effective approach to improve output accuracy and allow for easier review.

Provide relevant application context. Providing contextual information about the code's use case can help the AI assistant to generate more accurate and guideline-appropriate output, further improving the quality and security of generated code (Pearce et al., 2025).

Request self-assessment and commentary. Prompting the AI assistant to perform a self-assessment of its output (e.g., *"highlight any SQL injection vulnerabilities"*) can help to identify potential security issues. Instructions to comment on critical lines or code segments help draw attention to aspects that may need further review, such as potential attack vectors. This not only supports code transparency but also encourages the critical follow-up of reviewing and validating the generated output.

Handle input securely and avoid exposure of sensitive data. Prompts may contain sensitive information, which can be exposed if not handled cautiously. To reduce this risk, established best practices should be followed, including the use of placeholders such as `API_KEY` instead of actual secrets or credentials, avoiding hard-coded credentials, using environment variables, and leveraging secrets management tools, and, where available, activating privacy mode to reduce the risk of external data transmission (Pawel and Arkadiusz, 2025).

3.1.3. Reviewing and verifying generated code

Secure prompting and input handling already represent proactive review and verification at the input stage. This approach is equally critical during the subsequent output validation phase. We recommend the following practices to review and validate AI/generated code:

Treat AI/generated code as insecure by default. All AI/generated code should be treated as insecure by default (Perry et al., 2023) and be considered as a draft (Fu et al., 2023) that demands the same quality assurance and security review processes as human-written code (Klemmer et al., 2024). Moreover, given AI-specific risks, such as the lack of full application context (Khoury et al., 2023), task- and context-dependent inconsistencies (Majdinasab et al., 2023; Tony et al., 2025), or the mere presence of vulnerabilities unique to AI/generated code. These factors demand even stricter review and validation methods than typically applied to human-written code. Therefore, when using AI-assisted coding tools, outputs should be critically reviewed for hallucinations and potential security risks (Security, 0000). Variability in model behavior can stem not only from prompt formulation but also from configuration parameters like temperature. A higher temperature increases randomness in output, which may be desirable in creative contexts but poses risks in security-sensitive tasks. Individuals aiming for more predictable and secure results should consider using models with lower temperature settings or enabling features that enforce deterministic outputs when available. Reviewing and verification should include treating suggested plugins, libraries, and packages as potential attack vectors. Before applying, such components must be verified for authorship, actual existence, maintenance status, and overall security (Pawel and Arkadiusz, 2025).

Apply traditional testing, security tools, and practices. In addition to AI-specific strategies, traditional software engineering practices remain highly relevant for AI/generated code. These include maintaining comprehensive test coverage and utilizing established techniques such as static analysis and security scanning, for example, to cross-check dependencies against known vulnerability databases (Khoury et al., 2023).

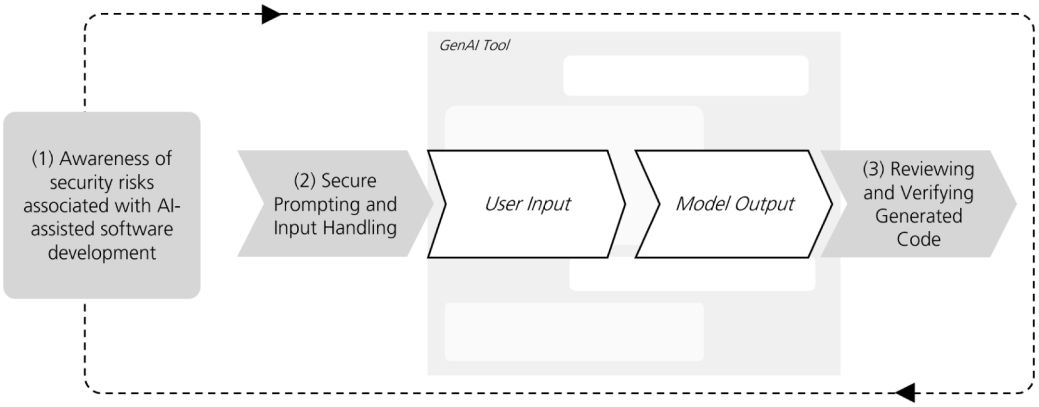

Fig. 2. Key stages of security-aware use of generative AI in software development. Secure usage requires (1) awareness of AI-related security risks, (2) careful prompting and input handling, and (3) critical review of the generated code.

Tools like Bandit or CodeQL can also help detect security issues that AI models might miss (Tony et al., 2025), while unit tests can be used to further identify potential vulnerabilities (Khoury et al., 2023).

Use feedback loops to improve generated code. Finding and fixing those vulnerabilities can be integrated into an automated, AI-assisted workflow by iteratively feeding identified issues back into the model for refinement (Khoury et al., 2023).

Execute code in isolated or secure environments. Lastly, in cases of uncertainty, running code in a sandboxed or secure environment limits potential impact and provides a safety net, reinforcing the need for cautious use.

3.2. Recommendations for organizations

While individual responsibility is essential, it is not sufficient, organizations must also establish structures, policies, and processes to ensure the secure and responsible use of AI/generated code.

3.2.1. Define security-aware AI governance requirements.

Their widespread use introduces new security risks that traditional governance processes are not yet equipped to handle. To address this, organizations need to acknowledge that AI assistants are already widely used in software development and that this trend is unlikely to reverse (Ji et al., 2024). To this end, rather than ignoring or banning their use, organizations should focus on governing their adoption in a security-aware way:

Policies for use cases. A key area of security-related AI governance concerns the use cases for which AI/generated code is considered appropriate, and what level of oversight is required in each context. Rather than banning AI in specific phases of development, organizations should assess where AI/generated output can be adopted with minimal risk, such as for scaffolding, or test generation, and where additional checks and validations are essential (Fu et al., 2023). For example, using AI in prototyping or internal tooling may require only routine peer review, while applying AI/generated code in security-critical components, production systems, or customer-facing features should trigger more rigorous quality assurance, including automated testing, threat modeling, and human validation as outlined in Section 3.1.

Policies for choice of AI assistants. Another area of governance involves setting clear requirements for the capabilities of AI assistants used in software development. Organizations should define which security and transparency features these tools must support. At a minimum, this includes the ability to log prompts and responses, store metadata for auditing purposes, and trace the origin of generated code. Another helpful feature can be the automatic annotation of AI/generated code to ensure accountability and facilitate code reviews. To implement these requirements, some AI assistants offer enterprise features that natively support such functionality (e.g., *Amazon CodeWhisperer or GitHub Copilot Enterprise*). Where such tools are not feasible due to financial or organizational constraints, general-purpose AI assistants (e.g., *ChatGPT* or *Gemini*) may still be used, provided additional safeguards are implemented. In these cases, organizations can route traffic through internal gateways that apply filters, label AI/generated code, and log interactions to meet the defined governance standards.

Policies for user groups. The third governance dimension concerns the roles and responsibilities of the people using AI assistants. When AI assistants are applied in critical or sensitive environments, organizations should implement role-based restrictions to limit risk. For example, only senior developers might be permitted to use AI-assisted code generation in production environments, while junior developers may be limited to non-production contexts. This ensures that the use of AI remains accountable and appropriately controlled based on experience and context.

3.2.2. Creating an AI-aware secure development operations process

In addition, organizations must ensure that secure AI coding practices are embedded into everyday development workflows. This means operationalizing the principles outlined above, not just as guidelines, but as enforceable practices integrated into the full secure development operations (SecDevOps) lifecycle (Mohan and Othmane, 2016) as illustrated in Fig. 3:

Plan, Develop, and Build. Security must be embedded early, starting in the planning phase. Teams should define clear policies for using AI-assisted coding, including which tools are permitted, what usage contexts are acceptable, and how to document AI involvement in the codebase. During development, developers must be trained to critically assess AI/generated suggestions and use secure defaults. Tooling should be introduced early in the build process, ideally at commit or merge stages, to detect common AI-related issues such as hallucinated dependencies or insecure code snippets before they propagate downstream. Integrated development environments (IDEs) can also support this by including plugins that warn about common AI-related vulnerabilities in real time.

Test. AI/generated code must be treated with heightened scrutiny during testing. All AI-assisted contributions should undergo both manual review and automated testing, with a strong focus on security (Klemmer et al., 2024). Dedicated tools (e.g., *Bandit* or *CodeQL*) can detect vulnerabilities that are difficult to catch through manual inspection alone. To scale this, organizations should introduce AI-specific validation steps, such as scanning for known flawed code patterns, hallucinated dependencies, or license issues (Kaniewski et al., 2024).

Release, Deploy, Operate, and Monitor. AI-related security risks do not end at deployment. As AI tooling evolves, previously safe outputs may become obsolete or unsafe. Organizations should treat their

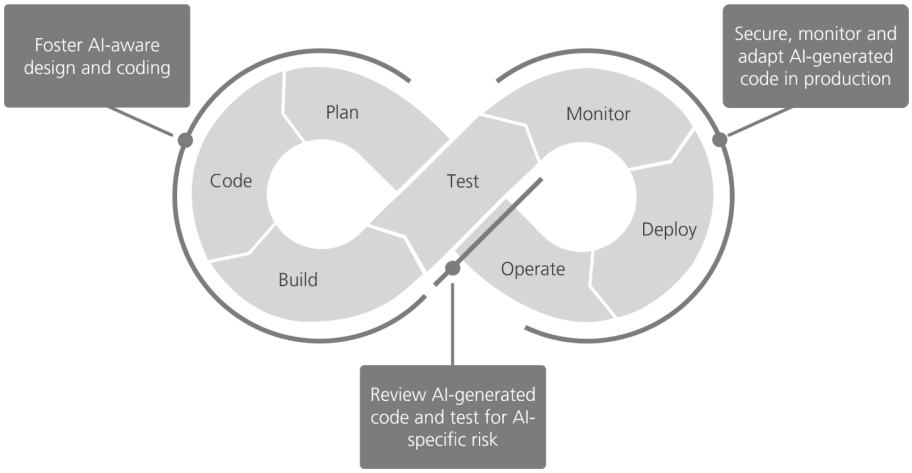

Fig. 3. AI-aware secure software development operations (SecDevOps): AI/generated code should be considered at all stages of the SecDevOps lifecycle.

validation toolchains and policies as living artifacts that must be regularly updated (Pearce et al., 2025). This includes revisiting test coverage, updating static analysis rules, and monitoring AI/generated code in production for anomalies or regressions. Runtime monitoring should flag abnormal behavior potentially stemming from insecure or faulty AI/generated logic. Feedback loops from monitoring can inform future planning and risk mitigation strategies, ensuring continuous improvement across iterations.

3.2.3. *Enable software developers through AI-security training.*

Finally, organizations must ensure that developers are aware of the security risks associated with AI/generated code, particularly in light of the tendency to overtrust AI systems (Klemmer et al., 2024; Synk, 2023). To address this, developers should receive targeted training on AI-specific security flaws as well as on secure prompt engineering and the critical evaluation of AI outputs, as introduced in Section 3.1.

Beyond individual knowledge, organizations should foster a culture of openness around AI usage. Developers should feel encouraged to share experiences, raise concerns, and discuss unexpected behavior or security issues related to AI/generated code.

In the following subsection, we provide concrete guidance on how to design and implement such training programs, whether in organizational contexts or academic curricula, offering specific recommendations for educators and trainers.

3.3. *Recommendations for educators*

Educators and corporate trainers play a critical role in preparing software professionals to navigate the risks and benefits of AI-assisted programming. To ensure that developers are both proficient and security-aware, we recommend the following in both academic institutions and corporate training environments:

Integrate secure use of AI assistants into the curriculum. AI coding assistants like GitHub Copilot and ChatGPT should be introduced in a structured way within programming courses or training modules. Assignments or workshops should explore both the benefits (e.g., productivity, learning support) and limitations (e.g., non-determinism, security risks). Learners and training participants should be required to *critically evaluate AI/generated code*, rather than treat it as authoritative output (Klemmer et al., 2024; Perry et al., 2023; Kazemitabaar et al., 2023; Wang et al., 2023).

Reinforce Secure AI Usage Principles. Educational programs should internalize the secure development practices outlined earlier by integrating them into training objectives and assignments. This includes fostering awareness of how to effectively interact with AI tools and emphasizing the continued importance of verification, oversight, and critical engagement with AI/generated code (rf. Sections 3.1 and 3.2).

Incorporate real-world vulnerability cases. Teaching should include examples of vulnerabilities introduced by AI/generated code, particularly in languages students are familiar with. Walkthroughs of identifying and correcting these flaws are highly effective (Perry et al., 2023; Fu et al., 2023; Sandoval et al., 2023).

Foster critical reflection and ethical awareness. Encourage open discussion around AI trust, developer responsibility, organizational accountability, and over-reliance. Learners and training participants should understand that while AI tools can be powerful, *they do not assume responsibility for security flaws*, humans **must** remain accountable (Klemmer et al., 2024; Perry et al., 2023).

Support open dialogue and experimentation. Rather than banning AI tools, foster safe environments where learners and training participants can experiment and reflect. Comparative analysis between AI- and human-generated solutions can lead to deeper insight and responsible usage. Such analyses could be incorporated as standard parts in periodic code reviews (Liang et al., 2024).

As AI-assisted development becomes mainstream, the need for ongoing education in secure practices spans the entire developer lifecycle, from classroom to corporate setting.

4. Outlook

While AI models excel at solving basic coding tasks, they often falter with complex problems involving intricate systems and higher-level reasoning (Shojaee*† et al., 2025; Wang et al., 2024). Models like Codex can solve simple algorithmic challenges with high success rates (up to 70.2% on HumanEval with sampling), yet these models struggle with long chains of logic, abstract architectures, and exact computation. Real-world development demands a deeper understanding, where current AI assistants frequently produce flawed or fragile code.

These limitations arise partly from how AI models are trained using pattern-matching over large corpora of public code. Simple tasks are well-represented online; complex ones are not. This imbalance mirrors performance gaps in lesser-used languages like Verilog, where sparse training data leads to poorer results. A growing concern is data contamination: as AI/generated content floods the internet, future AI models risk learning from degraded, synthetic data (Burden et al., 2024; ChatGPT, 2025). This recursive loop, known as model collapse, can harm model quality, even with small contamination levels.

To ensure safe, reliable code generation, clean and secure training datasets are urgently needed. This includes filtering out buggy or malicious code, defending against data poisoning, and preserving high-quality human-authored examples. Long-term viability will require

public policies supporting dedicated, protected data spaces to train the next generation of trustworthy AI models. Until we have such protected data spaces, AI-generated code will remain a security concern.

Declaration of Generative AI Technologies in the Writing Process

During the preparation of this work, the authors used ChatGPT (Model GPT-4o) in order to improve readability and language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

CRediT authorship contribution statement

Magdalena Glas: Conceptualization, Writing – original draft, Writing – review & editing; **Christoph Nirschl:** Writing – original draft; **Bar Lanyado:** Writing – review & editing; **Johan van Niekerk:** Conceptualization, Supervision, Writing – original draft, Writing – review & editing.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Ambati, S.H., Ridley, N., Branca, E., Stakhanova, N., 2024. Navigating (in)security of AI-generated code. In: 2024 IEEE International Conference on Cyber Security and Resilience (CSR). IEEE, pp. 1–8. https://doi.org/10.1109/csr61664.2024.10679468

Barke, S., James, M.B., Polikarpova, N., 2023. Grounded copilot: how programmers interact with code-generating models. Proc. ACM Program. Lang. 7 (OOPSLA1). https://doi.org/10.1145/3586030

Becker, B.A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., Santos, E.A., 2023. Programming is hard - or at least it used to be: educational opportunities and challenges of AI code generation. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. Association for Computing Machinery, New York, NY, USA, p. 500–506. https://doi.org/10.1145/3545945.3569759

Bhatt, M., Chennabasappa, S., Nikolaidis, C., Wan, S., Evtimov, I., Gabi, D., Song, D., Ahmad, F., Aschermann, C., Fontana, L., Frolov, S., Giri, R.P., Kapil, D., Kozyrakis, Y., LeBlanc, D., Milazzo, J., Straumann, A., Synnaeve, G., Vontimitta, V., Whitman, S., Saxe, J., 2023. Purple Llama cyberseceval: a secure coding benchmark for language models. https://doi.org/10.48550/ARXIV.2312.04724

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020. Language models are few-shot learners. arXiv:2005.14165.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Li, Y., Lundberg, S., Nori, H., et al., 2023. Sparks of artificial general intelligence: early experiments with GPT-4. https://arxiv.org/abs/2303.12712.

Burden, J., Chiodo, M., Grosse Ruse-Khan, H., Markschies, L., Müller, D., Seán, Ó.h., Podszun, R., Zech, H., 2024. Legal aspects of access to human-generated data and other essential inputs for AI training. University of Cambridge Faculty of Law Research Paper (35).

ChatGPT Has already polluted the internet so badly that it's hobbling future AI development — futurism.com. https://futurism.com/chatgpt-polluted-ruined-ai-development. [Accessed 19-06-2025].

Cotroneo, D., De Luca, R., Liguori, P., 2025. DeVAIC: a tool for security assessment of AI-generated code. Inf. Softw. Technol. 177, 107572. https://doi.org/10.1016/j.infsof.2024.107572

Cui, Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., Salz, T., 2025. The effects of generative AI on high-skilled work: evidence from three field experiments with software developers. https://ssrn.com/abstract=4945566

Fischer, F., Bottinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., Fahl, S., 2017. Stack overflow considered harmful? The impact of copy&paste on android application security. In: 2017 IEEE Symposium on Security and Privacy (SP). IEEE. https://doi.org/10.1109/sp.2017.31

Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., Chen, J., 2023. Security weaknesses of copilot generated code in github. https://doi.org/10.48550/ARXIV.2310.02059

GitHub, 2023. Survey reveals AI's impact on the developer experience. Accessed: 25-06-2025. https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/.

GitHub, 2024. Survey: the AI wave continues to grow on software development teams. Accessed: 2025-05-06. https://github.blog/news-insights/research/survey-ai-wave-grows/.

Haindl, P., Weinberger, G., 2024. Does ChatGPT help novice programmers write better code? Results from static code analysis, IEEE Access, 12, 114146–114156. https://doi.org/10.1109/access.2024.3445432

Hamer, S., d'Amorim, M., Williams, L., 2024. Just another copy and paste? Comparing the security vulnerabilities of ChatGPT generated code and stackoverflow answers. In: 2024 IEEE Security and Privacy Workshops (SPW). IEEE, pp. 87–94. https://doi.org/10.1109/spw63631.2024.00014

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., Han, W., Huang, M., Jin, Q., Lan, Y., Liu, Y., Liu, Z., Lu, Z., Qiu, X., Song, R., Tang, J., Wen, J.-R., Yuan, J., Zhao, W.X., Zhu, J., 2021. Pre-trained models: past, present and future. arXiv:2106.07139.

Ji, J., Jun, J., Wu, M., Gelles, R., 2024. Cybersecurity Risks of AI-Generated Code. Technical Report. Center for Security and Emerging Technology. https://cset.georgetown.edu/publication/cybersecurity-risks-of-ai-generated-code/.

Kaniewski, S., Holstein, D., Schmidt, F., Heer, T., 2024. Vulnerability handling of AI-generated code - existing solutions and open challenges. In: 2024 Conference on AI, Science, Engineering, and Technology (AIxSET). IEEE, pp. 145–148. https://doi.org/10.1109/aixset62544.2024.00026

Kazemitabaar, M., Chow, J., Ma, C. K.T., Ericson, B.J., Weintrop, D., Grossman, T., 2023. Studying the effect of AI code generators on supporting novice learners in introductory programming. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. ACM, pp. 1–23. https://doi.org/10.1145/3544548.3580919

Khoury, R., Avila, A.R., Brunelle, J., Camara, B.M., 2023. How secure is code generated by ChatGPT? In: 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, pp. 2445–2451. https://doi.org/10.1109/smc53992.2023.10394237

Klemmer, J.H., Horstmann, S.A., Patnaik, N., Ludden, C., Burton, C., Powers, C., Massacci, F., Rahman, A., Votipka, D., Lipford, H.R., Rashid, A., Naiakshina, A., Fahl, S., 2024. Using AI assistants in software development: a qualitative study on security practices and concerns. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, New York, NY, USA, p. 2726–2740. https://doi.org/10.1145/3658644.3690283

Lanyado, B., 2024. Diving deeper into AI package hallucinations. Accessed: 25-06-2025. https://www.lasso.security/blog/ai-package-hallucinations.

Liang, J.T., Yang, C., Myers, B.A., 2024. A large-scale survey on the usability of AI programming assistants: successes and challenges. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3597503.3608128

Majdinasab, V., Bishop, M.J., Rasheed, S., Moradidakhel, A., Tahir, A., Khomh, F., 2023. Assessing the security of github copilot generated code – a targeted replication study. https://arxiv.org/abs/2311.11177.

Mohan, V., Othmane, L.B., 2016. SecDevOps: is it a marketing buzzword? - mapping research on security in devops. In: 2016 11th International Conference on Availability, Reliability and Security (ARES), pp. 542–547. https://doi.org/10.1109/ARES.2016.92

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M.C., Jiang, Z. M.J., 2023. Github copilot AI pair programmer: asset or liability? J. Syst. Softw. 203, 111734. https://doi.org/10.1016/j.jss.2023.111734

Nair, M., Sadhukhan, R., Mukhopadhyay, D., 2023. How Hardened is Your Hardware? Guiding ChatGPT to Generate Secure Hardware Resistant to CWEs. Springer Nature Switzerland. pp. 320–336. https://doi.org/10.1007/978-3-031-34671-2_23

OWASP Foundation, 2024. OWASP Top 10 for LLM Applications – Version 2025. Technical Report. OWASP Foundation. Version 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/.

Pawel, M., Arkadiusz, G., 2025. Secure AI-assisted coding: a definitive guide — pragmaticcoders.com. https://www.pragmaticcoders.com/blog/secure-aiassisted-coding-guide. [Accessed 25-06-2025].

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R., 2025. Asleep at the keyboard? Assessing the security of github copilot's code contributions. Commun. ACM 68 (2), 96–105. https://doi.org/10.1145/3610721

Pearce, H., Tan, B., Ahmad, B., Karri, R., Dolan-Gavitt, B., 2023. Examining zero-shot vulnerability repair with large language models. In: 2023 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 2339–2356. https://doi.org/10.1109/sp46215.2023.10179324

Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M., 2023. The impact of AI on developer productivity: evidence from github copilot. https://doi.org/10.48550/ARXIV.2302.06590

Perry, N., Srivastava, M., Kumar, D., Boneh, D., 2023. Do users write more insecure code with AI assistants? In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.1145/3576915.3623157

Sandoval, G., Pearce, H., Nys, T., Karri, R., Garg, S., Dolan-Gavitt, B., 2023. Lost at C: a user study on the security implications of large language model code assistants. In: 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, pp. 2205–2222. https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval.

Security, F. O. f.I., . Ai coding assistants — bsi.bund.de. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/ANSSI_BSI_AI_Coding_Assistants.pdf?_blob=publicationFile&v=7. [Accessed 25-06-2025].

Shojaee*†, P., Mirzadeh*, I., Alizadeh, K., Horton, M., Bengio, S., Farajtabar, M., 2025. The illusion of thinking: understanding the strengths and limitations of reasoning models via the lens of problem complexity. https://ml-site.cdn-apple.com/papers/the-illusion-of-thinking.pdf.

Synk, 2023. AI Code, Security and Trust: Organizations Must Change Their Approach. Technical Report. Synk. https://go.snyk.io/2023-ai-code-security-report.html.

Tabassi, E., 2023. Artificial Intelligence Risk Management Framework (AI RMF 1.0). Technical Report. National Institute of Standards and Technology (U.S.). https://doi.org/10.6028/nist.ai.100-1

Tihanyi, N., Bisztray, T., Ferrag, M.A., Jain, R., Cordeiro, L.C., 2024. How secure is AI-generated code: a large-scale comparison of large language models. Empirical Softw. Eng. 30 (2). https://doi.org/10.1007/s10664-024-10590-1

Tony, C., Ferreyra, N. E.D., Mutas, M., Dhiff, S., Scandariato, R., 2025. Prompting techniques for secure code generation: a systematic investigation. https://arxiv.org/abs/2407.07064.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Łukasz, Polosukhin, I., 2017. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS). Curran Associates Inc., pp. 5998–6008. https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wang, W., Ning, H., Zhang, G., Liu, L., Wang, Y., 2024. Rocks coding, not development: a human-centric, experimental evaluation of LLM-supported SE tasks. Proc. ACM Softw. Eng. 1 (FSE). https://doi.org/10.1145/3643758

Wang, Y., Zhong, W., Li, L., Mi, F., Zeng, X., Huang, W., Shang, L., Jiang, X., Liu, Q., 2023. Aligning large language models with human: a survey. arXiv:2307.12966.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C., 2023. A prompt pattern catalog to enhance prompt engineering with ChatGPT. https://arxiv.org/abs/2302.11382.

Wolfram, S., 2023. What Is ChatGPT Doing:…and Why Does It Work? Wolfram Media.

Magdalena Glas received her master's and PhD degree from the University of Regensburg with stays abroad at the University College Dublin (Ireland) and the Ionian University (Greece). Since 2025 she has been a PostDoc at the University of Regensburg, Germany and a visiting researcher at NTNU Trondheim (Norway). Her research interests include authentic learning environments in organizational cybersecurity and other people-centric domains of cybersecurity.

Christoph Nirschl received his master's degree in management information systems from the University of Regensburg. Since 2024, he is a PhD Candidate at the Faculty of Informatics and Data Science at the University of Regensburg. His research interests include human-centered cybersecurity, with a focus on user behavior, decision-making, and effective security education.

Bar Lanyado currently leads security research at Lasso (Tel Aviv) bringing over seven years of experience within mobile and web applications, reverse engineering, supply chain attacks, and other security domains. He has shared his knowledge as a speaker at several industry events, including the BlueHat IL 2025 and OWASP Global Conference 2024.

Johan van Niekerk is a Professor of Cyber Security at Noroff University College in Norway. Previously, he was a Professor at the Nelson Mandela University in South Africa where he currently holds the position of an Honorary Professor in the Faculty of Engineering, the Built Environment and IT (EBEIT). His research focuses on the human, cultural, and educational dimensions of information and cyber security, with a particular emphasis on developing effective pedagogical approaches, fostering security-aware cultures, and integrating technology to enhance secure practices and learning outcomes. He is active in several IFIP working groups in both TC11 and TC3.