



ENISA Technical Advisory on AI-assisted software development

Fields marked with * are mandatory.

ENISA Technical Advisory on AI-assisted software development

Thank you for taking part in this open consultation. We are seeking feedback on our [draft Technical Advisory on AI-assisted software development](#). Your input will help ensure that the content is practical, relevant, and reflective of real-world practices.

This survey invites you to comment on key aspects of the advisory, including its overall approach, coverage of risks and threats, and the clarity and usefulness of its recommendations. We are also interested in whether the advisory is sufficiently concise and where content could be clarified, refined, expanded or removed.

Your feedback, whether based on hands-on experience, security expertise or general familiarity with AI-assisted software development, is highly valued. The results of this consultation will help inform the final revision of the Technical Advisory.

The consultation will remain open until the **15th of October 2026**.

Thank you in advance for your valuable contributions.

Data Protection

Authors may voluntarily provide their name, surname, email, and affiliation as part of this survey. If the author gives explicit consent, their name, surname, and affiliation may be included in the acknowledgements of the

final report; however, inclusion is not guaranteed.

Data collected within this survey will be deleted 3 months after the processing of the submissions.

☒ I have read and understood the [Data Protection Notice](#).

Respondent info

Name and surname (optional)

Tobie Langel

Current job title (optional)

Principal

Affiliation (optional)

UnlockOpen

☒ If names are included in the acknowledgements, I am happy for my name and affiliation to be included. I understand that inclusion is not guaranteed.

Please provide any relevant context about your work, organisation, or experience that may help us better understand your feedback.

I run a consulting business that helps organisations scale open technologies so they can have real-world impact. With that impact comes responsibility, and I also help organisations understand and address their regulatory and cybersecurity obligations.

As part of this work, I helped the open-source ecosystem organise around the Open Regulatory Compliance Working Group of the Eclipse Foundation, which I bootstrapped and served as technical lead for. Through the Eclipse Foundation, I am also a member of the European Commission's Cyber Resilience Act (CRA) Expert Group, where I have made substantial contributions to the Commission's implementation guidance. I also participate in joint meetings between the CRA Expert Group and the European AI Board.

In parallel, I am co-chair of the Cross Project Council of the OpenJS Foundation and serve on the Technical Steering Committee of Lodash, one of the most widely depended-on software libraries in the world. I have also worked closely with CISA on threats affecting OpenJS and open source projects.

Finally, I am also a practitioner and am currently developing an AI product using AI-augmented development workflows, with the aim of serving regulated markets and meeting demanding security, assurance and compliance requirements from the outset.

1) Overall, how useful do you find the draft Technical Advisory on AI-assisted software development?

Please explain your rating (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

The advisory is useful and well structured, particularly as guidance for AI-assisted development within a conventional software development lifecycle (SDLC). I strongly support ENISA addressing this topic.

My main concern is that the document does not yet fully account for how increasing autonomy and development velocity change the security model itself. I see four issues in particular:

1. Human review capacity does not scale with machine-generated output. As development throughput increases, a process can remain formally “human in the loop” while becoming progressively less meaningful in practice. Reviewers face approval fatigue, lack sufficient time or context to understand changes, and risk rubber-stamping work they cannot realistically inspect in depth. Human judgement, responsibility and accountability remain essential, but development and assurance need to be resourced proportionally, with human attention applied where it provides the most value.
2. Assurance needs to become risk-driven and increasingly engineered. The depth and type of assurance should be proportionate both to the risk profile of the product itself and to the risk introduced by the specific change. A change to authentication, authorization, cryptography, I/O boundaries or security configuration should not be treated the same as a low-impact documentation or UI change, and the same change may warrant different controls depending on the criticality and exposure of the product in which it occurs. Review depth, automated checks, deterministic gates and human intervention should therefore be driven by both product-level risk and change-level risk. As software production accelerates, assurance capacity must accelerate with it, with human judgement increasingly focused on control-system design, policy, exceptions and high-risk changes while routine verification is automated.
3. Agents require strong containment because they can fail or be compromised. Agents can misunderstand instructions, make mistakes, take unintended actions, be manipulated through prompt or memory poisoning, or be compromised through their tools, dependencies or surrounding environment. Sandboxing, least-privilege credentials, constrained capabilities, strong identity boundaries, and explicit ingress and egress controls therefore become fundamental security controls for limiting blast radius.
4. The distinction between development, build and production increasingly collapses. Where an agentic system continuously modifies, builds, verifies and deploys a running product, the development and build environment becomes part of the product’s effective cybersecurity boundary and software-supply-chain attack surface. Compromise of the software-production system can translate directly into compromise of the product. This also raises an important regulatory question: where that system retains continuous authority over the product, should product-security assurance explicitly account for the development and build environment as part of the effective security perimeter?

For these reasons, I think the advisory would benefit from framing AI-assisted development as a continuum of increasing autonomy and velocity, with an explicit shift in the control and assurance architecture as those increase. The current document provides a useful foundation for lower-autonomy use within an existing SDLC,

but higher-autonomy workflows increasingly require different containment, verification, governance and operational controls.

There is a real risk that organisations move quite far along this continuum while retaining assurance models, review processes and security assumptions designed for lower-autonomy, lower-throughput development. In that situation, development velocity and machine autonomy increase, but the surrounding controls do not evolve with them, creating a widening gap between how software is produced and how it is assured.

If this advisory intends to remain scoped primarily to AI augmentation of an existing SDLC, it should explicitly warn against this failure mode and explain when increasing autonomy begins to require a different security and assurance architecture. ENISA should then provide dedicated guidance on how security and assurance architectures need to evolve as AI-assisted development becomes increasingly autonomous.

2) How accurately and clearly does Section 2 represent a typical AI-assisted software development workflow and its main components?

6

Please explain your rating (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

Section 2 provides a clear baseline description of current AI-assisted development. The distinction between models, assistants or agents, tools and integrations, and project context is useful, as is the progression from developer-led assistance to goal-managed autonomous development.

My main concern is that it describes what agents can do without treating the security of the agentic system itself as a first-class architectural concern. An agent that can inspect repositories, modify files, run commands, install dependencies, use credentials or interact with CI/CD and deployment infrastructure is a privileged software component and attack surface. It can make mistakes, misunderstand instructions, take unintended actions, or be manipulated or compromised through prompts, context, tools, dependencies or connected services. Security therefore cannot depend primarily on the agent behaving correctly.

Section 2 should explicitly represent the security boundary around the agent, including:

- sandboxed execution and filesystem, process and network isolation;
- least-privilege, scoped and preferably short-lived identities and credentials;
- capability-scoped tools and explicit ingress, egress and data-flow controls;
- deterministic policy-enforcement points;
- tamper-resistant logging, telemetry and behavioural monitoring;
- containment, revocation, remediation and rollback.

Tools and integrations should also be described as security boundaries, not only as functional capabilities. Access should be capability-scoped, observable and subject to explicit policy, credential and network controls.

Commercial coding agents increasingly provide sandboxing or constrained execution, but the guarantees vary depending on how and where the agent runs. Local, self-hosted and custom harnesses may provide much weaker containment. The advisory should therefore make isolation, credential scoping and capability restrictions explicit rather than assuming an agent implies an effective security boundary.

The draft briefly mentions the agent harness or orchestration layer. This is a natural place to explain its security role more clearly. At higher autonomy, the harness can become the layer that enforces execution boundaries, permissions, credential use, network policy, tool capabilities, verification and observability.

As autonomy increases, the role of these controls changes. In developer-led assistance, the existing SDLC remains the primary control structure. In higher-autonomy workflows, the harness, sandbox, identity model, policy enforcement, verification, telemetry, deployment and rollback mechanisms increasingly become the control plane governing software production.

PROPOSED CHANGE TO FIGURE 1 / SECTION 2.1

I would consider showing two architectural views: one for lower-autonomy AI assistance within a conventional SDLC, and one for a higher-autonomy agentic workflow in which the surrounding engineering system carries more of the control and assurance burden.

Both should show the agent inside an explicit controlled execution environment, with sandboxing, identity, credential and capability boundaries, and with the harness mediating access to tools, data, networks and deployment capabilities. The distinction is not whether these controls exist, but the role they play as autonomy increases.

Supporting examples:

- OpenAI — Harness engineering: <https://openai.com/index/harness-engineering/>
- Anthropic — How we contain Claude across products: <https://www.anthropic.com/engineering/how-we-contain-claude>
- OpenAI — Running Codex safely at OpenAI: <https://openai.com/index/running-codex-safely/>

3) How well do the risks, failure modes and adversarial threats described in Section 3 reflect realistic scenarios in AI-assisted software development?

7

Please explain your rating (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

Section 3 is strong overall. The separation between non-adversarial failure modes, adversarial threats, and governance and assurance risks is useful, and many of the scenarios are directly relevant to current agentic development systems.

In particular, the advisory appropriately identifies unintended agent actions, manipulated prompts and repository context, malicious or spoofed models, tools and instructions, information disclosure, abuse of agent permissions, uncontrolled autonomy, insufficient review capacity and weak traceability. These are important risks, and I think the advisory is right to treat models, skills, instructions and other contextual inputs as security-relevant.

My main concern is therefore not that the threat scenarios are unrealistic or absent, but that their systemic consequences are understated.

The advisory generally describes individual threats against elements of the AI-assisted workflow. It should go one step further and show how these threats can be chained through the software-production system. A compromised model, agent, instruction source, memory or context source, connected tool, credential, repository or build component may not simply produce a bad suggestion: it may gain a path to source code, dependencies, build configuration, CI/CD infrastructure and ultimately deployment.

As autonomy and standing authority increase, the relevant question becomes not only “what can compromise the agent?” but also “what can a compromised agent reach, and how quickly can the consequences propagate?”

This makes blast radius particularly important. Automation can amplify both accidental and adversarial failures across many files, repositories, environments or deployments at a speed that makes manual intervention increasingly ineffective.

The threat model should therefore make three relationships more explicit:

1. Threats can compose. Prompt injection, poisoned context, compromised tools, excessive permissions and weak verification should not be considered only as isolated scenarios. Their combination can create an end-to-end path from untrusted input to privileged modification of software or infrastructure.
2. Impact depends heavily on authority and containment. The same compromised agent has very different consequences depending on its credentials, network access, tool capabilities, sandboxing, deployment permissions and ability to affect multiple repositories or environments.
3. Compromise of the development system can become compromise of the product. Where the agentic development apparatus retains authority to build or deploy changes, the model, agent framework, context and tool supply chain, credentials, build infrastructure, verification systems and deployment infrastructure become part of the product’s effective software-supply-chain security boundary.

In highly autonomous and continuously deployed environments, the threat model of the software-production system therefore begins to converge with the threat model of the operational product. An attack on the development environment is no longer merely an attack on an internal engineering process; it can become a direct software-supply-chain path into the product.

I would recommend making these propagation paths and the concepts of standing authority, blast radius and containment more explicit in Section 3.

Supporting examples and references

- Anthropic — How we contain Claude across products: <https://www.anthropic.com/engineering/how-we-contain-claude>
- OpenAI — Running Codex safely at OpenAI: <https://openai.com/index/running-codex-safely/>
- OpenSSF — Securing Agentic AI: <https://openssf.org/blog/2026/04/08/openssf-tech-talk-recap-securing-agentic-ai/>

4) How well do the good practices proposed in Section 4 address the relevant risks, failure modes and threats.

Please explain your rating (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

Section 4 contains several sound principles. Identifying security requirements, making them explicit, verifying outputs and retaining evidence are all useful. I also agree that AI-generated code should not receive weaker assurance than equivalent developer-produced code, and that verification should be proportionate to risk.

My main concern is that the section largely assumes the existing SDLC assurance model can simply be applied to AI-assisted development. That works while AI remains an assistant inside a conventional process, but becomes less convincing as development velocity and autonomy increase.

First, equivalent controls do not imply equivalent assurance when throughput changes. Applying the same review and approval practices to code produced much faster does not preserve assurance unless assurance capacity scales too. Human review is finite, and at high throughput a formally “human-in-the-loop” process can degrade into shallow review, approval fatigue or rubber-stamping.

Human judgement remains essential, but it should be applied where it adds the most value: control-system design, architecture, risk acceptance, exceptions, and changes whose impact is high, difficult to verify or difficult to reverse.

Second, assurance should be driven by both product risk and change risk. A change to authentication, authorization, cryptography, I/O boundaries or deployment infrastructure should trigger stronger assurance than a low-impact change. Review depth, deterministic gates, independent verification and mandatory human involvement should be selected according to both dimensions.

Third, deterministic engineering controls become more valuable as development accelerates. Tests, static analysis, dependency policy, type systems, architectural constraints, package allow-lists, reproducible builds, policy-as-code and CI/CD gates provide repeatable and independently enforceable assurance. AI can also reduce the cost of producing some of these controls, particularly tests. Where a check can be expressed reliably as code or policy, it should generally be enforced by the system rather than left solely to repeated human judgement.

Fourth, telemetry should be part of assurance, not only evidence after the fact. At higher autonomy, organisations need visibility into what agents attempted, which tools and credentials they used, what checks ran or failed, what was overridden and what reached deployment. This supports auditability, anomaly detection, incident response, control validation and continuous improvement. Evidence should increasingly be generated directly from the events that produced and verified a change.

Finally, Figure 2 risks implying that increasing autonomy is simply the existing lifecycle with more AI inserted into it. At higher autonomy, assurance increasingly becomes an engineered system around software production itself: policy enforcement, testing, security scanning, independent review, telemetry, deployment controls, rollback and human escalation operating continuously.

The aim should not be to preserve the same amount of human review per change as throughput rises, but to preserve or improve assurance by combining risk-based human judgement with deterministic and automated controls whose capacity scales with software production.

The same applies to defense. As AI increases the speed of vulnerability discovery and exploitation, defensive assurance cannot remain predominantly periodic or human-paced. Organisations will increasingly need

continuous capabilities such as automated vulnerability discovery, agentic red teaming, adversarial testing, remediation, rapid fix validation and rollback or containment. If software production and offensive capability accelerate, defensive verification and remediation must accelerate as well.

Supporting examples:

- Vercel — DeepSec: <https://vercel.com/blog/introducing-deepsec-find-and-fix-vulnerabilities-in-your-code-base>
- OpenAI — Codex Security: <https://openai.com/index/codex-security-now-in-research-preview/>
- GitHub — Agentic Autofix: <https://github.blog/changelog/2026-07-10-agentic-autofix-for-code-scanning-alerts-in-public-preview/>

5) How useful is the illustrative example in Annex A in demonstrating how the proposed approach can be applied in practice?

3

Please explain your rating (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

I understand the intention of Annex A: to show how security expectations can be made explicit to an assistant or agent. The annex is also appropriately cautious that skills do not guarantee secure output and do not replace engineering judgement, review, vulnerability management or accountability.

My concern is more fundamental: I would not use skills as the primary worked example for secure AI-assisted development.

Skills are useful as one mechanism for communicating organizational expectations to an agent. They can help make requirements explicit, improve consistency, and guide the agent toward preferred practices. But they still locate security primarily in the behavior of the agent producing the code.

For increasingly agentic development, many of the most important security properties should instead be embodied in the surrounding control environment and remain effective independently of whether the agent correctly understands or follows an instruction.

For example:

- “Only use approved dependencies” is useful guidance to the agent. A package proxy or network policy that makes unapproved sources inaccessible is an enforceable control.
- “Run the required security checks” is useful guidance. A build or deployment pipeline that cannot proceed unless independently verifiable checks pass is an enforceable control.
- “Do not modify this sensitive area without review” is useful guidance. A repository or policy rule that automatically triggers stronger verification or mandatory human review is an enforceable control.

This distinction matters particularly because agents are probabilistic and can fail or be compromised. Instructions can be misunderstood, omitted, overridden by conflicting context, affected by prompt or memory poisoning, or deliberately manipulated. A security property that exists only as an instruction is therefore inherently dependent on agent behavior. A security property enforced independently by the surrounding system can continue to hold even when the agent fails.

There is also a durability issue with making skills the main example. Model and agent capabilities are improving rapidly, including their ability to produce more secure code, reason about security requirements, invoke appropriate checks, and use built-in or provider-supplied skills and security capabilities. Guidance centred on manually encoding secure behavior into skills may therefore age quickly.

The more durable guidance is to define the security properties that must hold regardless of which model or agent is used, and then enforce or independently verify those properties through the surrounding software-production system.

I would therefore mention skills only as one illustrative mechanism for communicating expectations to an agent, rather than making them the primary practical example.

A stronger worked example would demonstrate a complete control path around an agentic change. For example, for dependency introduction:

1. the agent proposes a dependency;
2. policy determines whether the source is permitted;
3. network controls prevent use of unapproved registries;
4. credentials are scoped to the required action;
5. provenance, vulnerability and integrity checks run independently;
6. automated tests and static analysis execute;
7. product-level and change-level risk determine whether stronger verification or human review is required;
8. deployment is blocked if mandatory checks fail; and
9. telemetry and attestations record what happened, which controls ran, and what was accepted or overridden.

Skills can still appear in such an example as a way of helping the agent understand the intent behind those controls. But the main lesson should be that the security of the system does not depend on the agent following the skill correctly.

I would therefore recommend replacing Annex A's skills-centric worked example with a model-independent example of security enforced through the software-production control environment, while mentioning skills as one supporting mechanism among several. This would make the guidance more robust to improvements in models and agents, and more clearly separate the quality of agent behavior from the security guarantees provided by the system around it.

6) Any additional comments or suggestions? (optional)

Note: If you suggest a change, please indicate the relevant section or subsection, quote or describe the current text, and provide an example of the proposed revised or additional text.

I strongly welcome ENISA addressing AI-assisted software development. ENISA's Secure by Design and Default guidance provides a strong foundation for product security, and this advisory is an important extension of that work.

TWO CONVERGING ACCELERATIONS

I see two converging forces pushing software security toward more automated and increasingly autonomous systems.

The first is development velocity. Agentic development is increasing the speed and volume at which software can be created, modified and deployed. Human review and conventional assurance cannot scale linearly with that increase.

The second is threat velocity. AI is also increasing the speed at which vulnerabilities can be discovered, analyzed, weaponized and exploited, compressing the time defenders have to identify and remediate weaknesses.

These trends point in the same direction. Software production cannot accelerate while assurance and defense remain predominantly periodic and human-paced. Verification, security testing, remediation and response will increasingly need to operate continuously and at machine-relevant speeds.

THE CONTROL AND ASSURANCE MODEL HAS TO CHANGE

In lower-autonomy settings, agents can operate within an existing SDLC whose controls and assurance processes are still largely designed around human-led development. As autonomy and velocity increase, both have to evolve: stronger technical controls are needed to constrain what agents can do, while verification and assurance need to become more continuous, automated and integrated into the surrounding engineering system.

Autonomy and velocity are not binary states; they are trending upward along a continuum. The danger is that organisations can move far along that continuum while retaining controls and assurance mechanisms designed for lower-autonomy, lower-throughput development. Development and build infrastructure can then become an always-on operational system and part of the product's effective software-supply-chain security boundary and attack surface.

If the current advisory is primarily about AI augmentation of an existing SDLC, it should say so explicitly, warn against this failure mode, and explain when increasing autonomy requires a different security and assurance architecture. ENISA should then develop dedicated guidance on how that architecture should evolve.

APPLY SECURE BY DESIGN TO THE DEVELOPMENT SYSTEM ITSELF

This does not require a new security philosophy. Many Secure by Design and Default principles apply equally well to the software-production environment: secure defaults, least privilege, attack-surface reduction, strong identity boundaries, observability, resilience, recovery, provenance and controlled change.

The challenge is to apply those principles recursively: secure the system that builds, verifies and deploys the product with the same discipline expected of the product itself.

Dedicated guidance should focus on durable properties rather than specific frameworks: containment, isolation, standing authority, blast radius, identity and permissions, verifiability, reversibility, provenance, observability, deterministic policy enforcement, independent assurance, rollback and recovery.

Operational cybersecurity disciplines must also extend to development, build and deployment, including telemetry, anomaly detection, incident response, containment and recovery.

DEFENSE HAS TO ACCELERATE TOO

As the threat environment accelerates, defensive assurance will also need to become more automated and continuous: automated vulnerability discovery, agentic red teaming, continuous adversarial testing, remediation and patch generation, rapid fix verification, telemetry-driven detection, and automated containment or rollback.

WE HAVE SEEN THIS PATTERN BEFORE

We have seen a version of this transition before. The research behind the book *Accelerate* showed that higher software-delivery throughput did not require trading away security or stability; high-performing teams were able to move faster while also achieving better security outcomes through automation, testing, observability, feedback loops and disciplined engineering practices.

Agentic development looks like the next iteration of that pattern. Increasing velocity will expose the limits of manual, human-paced assurance, just as continuous delivery exposed manual build and release processes. The difference this time is that threat velocity is increasing too.

Software production, assurance and defense will therefore all need to become more automated and continuous, while human judgement moves toward governance, architecture, risk acceptance, exceptions and the highest-impact decisions. Greater velocity makes good engineering practices more important, not less.

If you would like to share any supporting material (e.g., examples, guidelines, research, or other feedback), please upload it here (optional)

Thank you for providing your feedback and for your support in our endeavors.

Please feel free to get in touch by sending us an email at: product_security@enisa.europa.eu

Contact

[Contact Form](#)