IBIMA Publishing
Journal of Software & Systems Development
https://ibimapublishing.com/articles/JSSD/2026/397496/
Vol. 2026 (2026), Article ID 397496, 7 pages, ISSEN: 2166-0824
https://doi.org/10.5171/2026.397496

Research Article

Software Ecosystem Growth Dynamics: Dependencies, Complexity, Vulnerabilities, and the Influence of AI Tools

Artur ZIEBA-KOZARZEWSKI

KRYPTON Polska Sp. z o.o., Warsaw, Poland
a.zieb@krypton-polska.com,
https://orcid.org/0000-0003-4696-4983

Received date:13 January 2026; Accepted date: 8 June 2026; published date: 14 July 2026

Abstract

The growing complexity of software development, driven by the extensive use of external dependencies and AI-based development tools, has become an important but still underexplored factor affecting software quality, maintainability, and security. As large language models become increasingly embedded in everyday development workflows, they may also shape decisions that once required more deliberate technical judgement, including whether a new dependency should be introduced in the first place. Although existing research has examined the quality of AI-generated code and the effectiveness of LLM-supported programming, much less attention has been given to the ways in which these tools may influence dependency selection or change the structure of dependency networks. This issue becomes more significant as such networks continue to expand and, in turn, increase security exposure. To address this gap, the study investigates changes in dependency counts in relation to the adoption of AI-based development tools and considers how these changes may affect vulnerability risk.

The methodology combines a longitudinal review of dependency metrics from 2021 to 2024 with correlation analysis and a simple probabilistic model used to estimate the likelihood of multiple vulnerabilities in projects with extensive dependency trees.

The results indicate a strong positive correlation ($r \approx 0.995$) between the adoption of AI tools and the growth in dependency counts. They also suggest a high probability—often exceeding 95% in major ecosystems—that an average project will contain at least three vulnerabilities. These findings point to the need for further research on how AI tools affect architectural and dependency-related decisions, as well as for practical strategies to reduce the resulting security risks.

Keywords: artificial intelligence, software, dependency, llm, vulnerability.

Cite this Article as: Artur ZIEBA-KOZARZEWSKI (2026)," Software Ecosystem Growth Dynamics: Dependencies, Complexity, Vulnerabilities, and the Influence of AI Tools ", Journal of Software & Systems Development, Vol. 2026 (2026), Article ID 397496, https://doi.org/10.5171/2026.397496

Introduction

Broadly defined, software today forms the practical basis for the functioning of devices and technologies used in everyday life. Various types of logical structures implementing complex algorithms operate in home consumer devices, means of transport, and critical infrastructure. Their operation, maintenance, and development are the pillars of further civilization and comfortable existence of our society. The development of professional software is a complex, multi-stage manufacturing process. The standard software development cycle usually consists of the following stages: requirements analysis, architecture design, code writing, testing, integration, error diagnostics, and bug fixes. The project development itself can be based on various management methodologies, from classic waterfall to agile ones such as Agile or Scrum. Ultimately, however, the goal of these processes is the same – to obtain software that meets specific customer requirementsi. The above development model is associated with two negative phenomena. The first stems from the complexity of the real world and the need to define the interactions between it and the software. This makes it difficult to describe the specifics of the expected behaviour with the required accuracy and affects the quality of the project's input requirements, and thus the entire project.

The second problem is the growing number of dependencies of the produced material on external, specialized software components.

The following paper focuses on the second issue, i.e., the increase in the number of software dependencies, particularly in connection with the popularization of support tools using artificial intelligence (AI) methods such as large language models (LLMs).

Related Works

Recent developments in generative artificial intelligence systems and their impact on software development have been the subject of numerous studies and analyses. Research has been conducted on both the impact of AI-based support tools on the productivity of developers themselves and on the quality of the material they produce, as well as on the importance of these tools in the context of support during the development process. The issue of requirements analysis prior to the start of software development work has also been considered, as well as the issue of the education process itself in the field of programming.

Software Development Process

The software development process mentioned at the beginning requires an analytical decomposition of the expected product into its constituent parts. This allows for the parallel development of its functionality and simplifies the future maintenance process. The implementation of these functionalities has often already been carried out by someone else and could be used in the material being developed. This is recommended by the widely accepted principle of software development optimization; this opportunity is provided by the development of libraries and frameworks, carried out both within the internal resources of various software development organizations and by the Open-Source Community. This approach also allows for better testing of the modules used. External components under open-source licenses are used in many solutions, which provides a large group of “involuntary” testers.

At the same time, it is evident that the above-mentioned external software components contain potential vulnerabilities and errors, which have a direct impact on the applications that implement them. In the software development process, it is necessary to be aware not only of the direct dependencies of the software being developed on external components, but also of the interdependencies of the components used, considering the impact of each possible dependency. This results in the creation of complex networks of connections. Previous studies have shown that, in most development environments, the complexity of dependency networks increases over time. In each development environment, this applies to both an increase in the number of packages and an increase in transitive complexity (i.e., a greater number of dependencies between packages). The dependencies of individual packages used may also have additional restrictions on acceptable versions (not less than/not greater than). This structure affects the complexity of solvability of the network of

dependencies between packages by unique instances of packages in a specific version. Determining the exact package versions that meet all the conditions of the dependency network is an NP-complete computational problem. If, in addition, it is necessary to ensure the correct dependencies for multiple programs within a single system, it may turn out that such an arrangement will never be satisfied. If a given package is to be present in the entire system in only one version, the so-called One-Version Rule.

As mentioned above, when developing software, it is recommended to use pre-existing components. An analysis of changes over subsequent years shows that the number of different externally sourced components used as dependencies in software development projects is on the rise, as is the annual number of new releases.

- 2021 - an average of 128 dependencies, an average of 10 releases per year,
- 2022 - an average of 148 dependencies, an average of 10 releases per year,
- 2023 - an average of 148 dependencies, an average of 15 releases per year,
- 2024 - an average of 180 dependencies, an average of 16 releases per year.

In the process of developing new features, programmers use well-known and currently used technologies. Reimplementation and introduction of new libraries into the project takes place when the expected workload for using current solutions seems disproportionately large compared to the benefits obtained.

Moderating Processes

In mature organizations, the process of adding a new library to a project often requires approval from the committee responsible for software security or license compliance. This serves as a moderating factor in the process of excessive growth in the number of external dependencies. It is easier for a software developer to use an existing, approved library, as they do not have to go through the entire process of approving a new component. However, with the increase in the use of specialized assistants based on extensive language models, this condition will no longer be met, and the approval process itself will be automated.

In the first phase, decision delegation will be in practice, meaning that the result of the support system will be approved by a human operator. This will significantly reduce the time needed to moderate the growth of new dependencies in the software and ultimately eliminate the burden on the programmer.

Increased Use of Artificial Intelligence Tools in Software Development

Supporting routine tasks with various automation techniques is a standard feature of professional Integrated Development Environments (IDEs). When large language models were introduced, it became natural to integrate them into these tools. The first solution of this type was introduced in an experimental version in June 2021. In the following years, further growth in the penetration of development environments by support solutions could be observed. Detailed survey data on the use or planned use of AI tools by software developers have been available since 2023. For 2021, only estimates are available for selected tools (Microsoft Copilot); for 2020, a value of less than 5% was assumed (as likely experimental work).

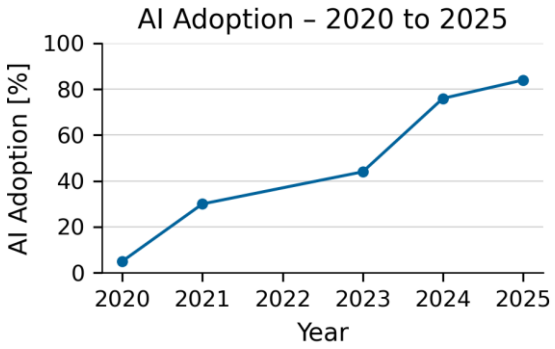

Fig 1. AI Adoption

Based on the above data, it can be clearly stated that, since around mid-2023, the use of AI-based support tools has exceeded 50% and has become the dominant software development technique, Fig 1. This change is most likely permanent and will remain a permanent part of the techniques and tools used for software development.

Growth in the Number of Dependencies

In recent years, with the increasing adoption of AI-assisted tools for software development, there has been a further increase in the number of different dependencies in IT projects (Table 1). The Pearson correlation coefficient for the above random variables is r ≈ 0.995, which should be interpreted as a strong positive correlation. It should be remembered that the sample is small at this point, and positive correlation alone does not imply causality, but the mechanisms described above may provide a basis for further research and observation of the phenomenon.

Table 1: Dependency and AI adoption per year

Year	Average dependency count	AI tools adoption percentage
2021	128	30%[ii]
2022	148	48%[ii]
2023	148	44%
2024	180	74%
2025	-[iii]	86%

In the upcoming years, further growth in the above relationships is to be expected.

Implications for Security

The increased use of assisted tools leads to increased code complexity. Maintenance work on such material is labour-intensive, resulting in an increase in technical debt and higher code maintenance costs.

Poorly maintained complex code negatively affects the security of the final product. Vulnerabilities in dependencies that are component parts constitute potential points of attack by hostile entities. A greater number of dependencies means a greater number of potential points of interaction that attackers can exploit.

Assuming that p is the average probability of vulnerability occurring in component software, and n is the number of dependencies on external components of a given IT project, it can be predicted that on average there will be V_{avg} vulnerabilities, according to the following formula:

$$V_{avg} = np$$

Breaking the software by an attacker to achieve the desired effect usually requires more than one vulnerability, so a situation of significant risk, i.e., one with no fewer than three vulnerabilities, can be considered. Assuming, for simplicity, that vulnerabilities in component parts are independent of each other, the probability of at least k vulnerabilities occurring in the software can be calculated using the transformed binomial distribution X~Bin(n,p) for k:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Since:

$$P(X \geq k) = 1 - P(X \leq k-1)$$

for k>1 we obtain:

$$P(X \geq k) = 1 - \sum_{l=1}^{k} P(X = l-1)$$

Substituting the currently available data on the number of dependencies (180) and the probability of vulnerabilities occurring in software from different environments, in the above formula, we obtain the following results, Table 2:

Table 2: Probability of vulnerabilities in ecosystems

Environment	p	$P(X \geq 3)$
PyPI	0.42%	4.09%
RubyGems	7.5%	~100%
Cargo	3.91%	97.33%
NPM	6.93%	99.97%

It can therefore be concluded that the probability of at least three vulnerabilities occurring in most environments is significant, exceeding 95%, and only in one environment is it less than 5%.

Conclusions

Analysis of the data presented above clearly indicates an increase in the number of dependencies used by software in recent years, which translates into the number of potential vulnerabilities. The probability of a significant threat occurring, i.e., one in which the number of vulnerabilities exceeds 3, exceeds 95% in most environments.

At this point, the increase in the number of dependencies in software as a result of using AI-powered tools remains a working hypothesis. Currently available data do not allow for clear conclusions to be drawn regarding the impact of this factor on the increase in the number of dependencies in software.

Future Work

It seems reasonable to extend the scope of further research to include an analysis of how software development tools using artificial intelligence decide on the choice of a given package to solve a problem requested by users, and an analysis of the extent to which these tools take into account issues related to the increase in the number of dependencies used, as well as issues related to vulnerabilities present in them.

It is also recommended to develop in-depth reports and analyses on software development using artificial intelligence-assisted techniques, the impact of these techniques on code quality (these analyses are currently underway), the number of dependencies created, and the number of vulnerabilities present in them.

Another reasonable direction for further work is the development of models dedicated to minimizing the negative impact of the increase in dependencies and mitigating the resulting vulnerabilities.

References

- Abate, P. et al. (2020) 'Dependency solving is still hard, but we are getting better at it', 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE.
- Arora, C., Grundy, J. and Abdelrazek, M. (2024) 'Advancing requirements engineering through generative AI: Assessing the role of LLMs', in Generative AI for Effective Software Development. Cham: Springer Nature Switzerland, pp. 129–148.
- Besker, T., Martini, A. and Bosch, J. (2018) 'Technical debt cripples software developer productivity: A longitudinal study on developers' daily software development work', in TechDebt. ACM, pp. 105–114.
- Decan, A., Mens, T. and Grosjean, P. (2017) 'An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems', arXiv:1710.04936.
- Dou, S. et al. (2024) 'What's wrong with your code generated by large language models? An extensive study', arXiv:2407.06153.
- Gaffney, J. and Cruickshank, R. (1992) 'A general economics model of software reuse', International Conference on Software Engineering, pp. 327–337.
- Gershgorn, D. (2021) 'GitHub and OpenAI launch a new AI tool that generates its own code', The Verge, 29 June. Internet Archive [accessed 04.01.2026].
- GitHub reports that its AI assistant Copilot was suggesting around 30 % of newly written code in certain programming languages, highlighting early integration of AI into code generation. 'GitHub sees uptick in coders using AI assistant', Axios, 27 Oct 2021. https://www.axios.com/2021/10/27/copilot-artificial-intelligence-coding-github [accessed 04.01.2026].
- Hemmat, A. et al. (2025) 'Research directions for using LLM in software requirement engineering: A systematic review', Frontiers in Computer Science, 7, p. 1519437.
- Jimenez, C.E. et al. (2023) 'SWE-bench: Can Language Models Resolve Real-World GitHub Issues?', arXiv:2310.06770.
- Kim, D.Y.J. (2018) Redefining Computer Science Education: Code-Centric to Natural Language Programming with AI-Based No-Code Platforms. ACM, New York, NY, USA.
- Liu, Z. et al. (2024) 'No need to lift a finger anymore? Assessing the quality of code generation by ChatGPT', IEEE Transactions on Software Engineering, 50(6), pp. 1548–1584.
- Mancinelli, F. et al. (2012) 'Dependency solving: A separate concern in component evolution management', Journal of Systems and Software, 85(10), pp. 2228–2240. https://doi.org/10.1016/j.jss.2012.02.018
- Márquez, A.G., Varela-Vaca, Á.J. and López, M.T.G. (2025) 'A dataset on vulnerabilities affecting dependencies in software package managers', Data in Brief, 62, 111903. https://doi.org/10.1016/j.dib.2025.111903
- Mertens, F. (2023) 'The Use of Artificial Intelligence in Corporate Decision-Making at Board Level: A Preliminary Legal Analysis', Financial Law Institute Working Paper Series 2023-01. https://ssrn.com/abstract=4339413 or http://dx.doi.org/10.2139/ssrn.4339413
- Murphy, R., Winters, T., Sculley, D. and Hennesy, T. (2020) Software Engineering at Google: Lessons Learned from Programming Over Time. Boston: Addison-Wesley Professional.
- Murphy, R., Winters, T., Sculley, D. and Hennesy, T. (2020) Software Engineering at Google: Lessons Learned from Programming Over Time. Boston: Addison-Wesley Professional.
- Ruparelia, N.B. (2010) 'Software development lifecycle models', ACM SIGSOFT Software Engineering Notes, 35(3), pp. 8–13.
- Sonatype (2022) 2022 State of the Software Supply Chain Report. Sonatype. Retrieved from https://www.sonatype.com/hubfs/1-2023%20New%20Site%20Assets/SSCR/8th-Annual-SSCR-digital-0206%20update.pdf [accessed 28.12.2025].
- Sonatype (2023) 2023 State of the Software Supply Chain Report. Sonatype. Retrieved from https://www.sonatype.com/hubfs/SSC/2023%20Sonatype-%209th%20Annual%20State%20of%20the%20Software%20Supply%20Chain-%20Update.pdf [accessed 28.12.2025].
- Sonatype (2024) 2024 State of the Software Supply Chain Report. Sonatype. Retrieved from https://www.sonatype.com/hubfs/SSCR-2024/SSCR_2024-FINAL-10-10-24.pdf [accessed 28.12.2025].

- Stack Overflow (2023) Stack Overflow Developer Survey 2023: AI tools in the development process. Retrieved from https://survey.stackoverflow.co/2023/ [accessed 04.01.2026].
- Stack Overflow (2024) Stack Overflow Developer Survey 2024: AI adoption and developer tool use. Retrieved from https://survey.stackoverflow.co/2024/ai [accessed 04.01.2026].
- Stack Overflow (2025) Stack Overflow Developer Survey 2025: AI adoption and developer tool use. Retrieved from https://survey.stackoverflow.co/2025/ai [accessed 04.01.2026].
- Tanzil, M.H., Uddin, G. and Barcomb, A. (2024) '"How do people decide?": A Model for Software Library Selection', Proceedings of the 2024 IEEE/ACM 17th International Conference on CHASE. ACM, pp. 1–12. https://doi.org/10.1145/3641822.3641865
- Tellnes, J. (2013) Dependencies: No software is an island. MS thesis. The University of Bergen.
- Xu, B., An, L., Thung, F. et al. (2020) 'Why reinventing the wheels? An empirical study on library reuse and re-implementation', Empirical Software Engineering, 25, pp. 755–789. https://doi.org/10.1007/s10664-019-09771-0
- Zadenoori, M.A. et al. (2025) 'Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review', arXiv:2509.11446.
- Zhong, L. and Wang, Z. (2024) 'Can llm replace stack overflow? A study on robustness and reliability of large language model code generation', Proceedings of the AAAI Conference on Artificial Intelligence, 38(19).

[i] The term "customer" should be understood as both the person defining the requirements and the person sponsoring the project.

[ii] Industry data.

[iii] Currently no data available.