O. Nahornyi, V. Gerasymov, T. Vorova

# EFFECTIVE DEPENDENCY MANAGEMENT IN SCALABLE PROGRAMMING PROJECTS

In the dynamic landscape of software development, managing dependencies is a critical aspect, particularly as the projects scale in complexity and size. This necessitates the effective strategies to understand, control, and optimize the inter-dependencies between various components. By examining industry best practices, established principles, and practical tools, we aim to provide insights into fostering a robust and scalable codebase.

As programming projects grow, the structure of dependencies between various components becomes increasingly challenging to visualize and comprehend. New modules, libraries, and external dependencies can create the intricate networks of interactions, not only complicating the understanding of the current project state but also hindering changes or additions of new functionalities [1].

The use of different versions of libraries within a project can lead to unpredictable version conflicts. For example, if Module A uses version 1.0 of Library X, and Module B uses version 2.0, compatibility issues may arise, impacting the correctness of project building, deployment, and even functionality.

With the expansion of project scope and the addition of new components, maintaining a dependency structure that facilitates effective project scalability becomes challenging. The absence of clear rules and standards may render the project difficult to maintain, impeding the process of making changes [3].

The possible solutions to the above-mentioned problems might be as follows: a) use of dependency management systems, b) automated dependency analysis, c) standardization and documentation, d) application of design patterns and architectural solutions.

*a) Use of dependency management systems:*

dependency management tools such as Maven, Gradle, or npm provide mechanisms for automating dependency management. They automate the processes of downloading, installing, and updating libraries, ensuring uniformity in versions and preventing conflicts [1].

*b) Automated dependency analysis:*

utilizing tools for dependency analysis, such as Dependency Structure Matrix (DSM) or visualization tools, allows developers to better understand the relationships between project components. This enhances the perception of the project structure and facilitates decision-making regarding modifications [1].

*c) Standardization and documentation:*

implementing strict coding and dependency documentation standards helps to establish a unified approach within the development team. Documenting dependencies in clear and understandable formats makes the project more accessible to the new team members and streamlines the maintenance process [2].

*d) Application of design patterns and architectural solutions:*

utilizing design patterns, such as Dependency Injection, contributes to creating the loosely coupled components, making the project more flexible and ensuring an easiness of making changes. Applying the architectural principles like SOLID also aids in dependency management and facilitates an easiness of maintenance [4].

**REFERENCES**


1. Eric Evans. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.
2. Martin Fowler. (2014). Dependency Injection. [Electronic resource].– Access mode: https://martinfowler.com/articles/injection.html
3. Robert C. Martin. (2002). Principles of Object-Oriented Design. [Electronic resource].– Access mode: https://web.archive.org/web/20140213162847/http://butunclebob.com/ArticleSUncleBob.PrinciplesOfOod
4. Steve McConnell. (2004). Code Complete: A Practical Handbook of Software Construction. MicrosoftPress.



N. Nomerchuk, T. Prokofiev, O. Hurko


## OPTIMIZATION OF EXPONENTIAL FUNCTION COMPUTATION IN JAVA USING LINEAR INTERPOLATION METHOD

Exponential functions used by standard mathematical libraries have high precision but concurrently demonstrate considerable computational overhead. To optimize performance across various scenarios, such as neural network implementations