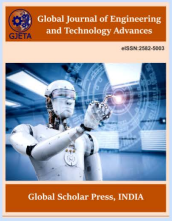
Global Journal of Engineering and Technology Advances
eISSN: 2582-5003
Cross Ref DOI: 10.30574/gjeta
Journal homepage: https://gjeta.com/



(REVIEW ARTICLE)

Check for updates

Survival of software reuse: Node.js packages component selection and dependencies

Tolulope Amos Awoniyi * and Stephen Olusola Maitanmi

Department of Software Engineering, Babcock University, Ilishan-Remo, Ogun State PMB 4003, Nigeria.

Global Journal of Engineering and Technology Advances, 2024, 19(01), 207–211

Publication history: Received on 12 January 2024; revised on 17 April 2024; accepted on 20 April 2024

Article DOI: https://doi.org/10.30574/gjeta.2024.19.1.0015

Abstract

Software reuse is a widely recognized and extensively studied practice in the field of software engineering. It involves using pre-existing software components to enhance productivity and efficiency. Node.js and its package manager enable software reuse by managing dependencies. Dependencies play a crucial role in defining the connections and interactions between various components of a software system. The selection of components and their interactions can greatly impact the functionality, performance, and stability of a software system. Overlooking these factors or making poor choices regarding component selection may lead to incomplete or flawed systems. Various factors that impact the decision-making process for selecting components were discussed, including documentation availability, crowdsourcing initiatives, and GitHub activities.

Keywords: Component; Dependencies; Node.js; Packages; Software reuse

1. Introduction

The practice of software reuse is well-known and extensively documented in the field of software engineering. It involves the utilization of pre-existing software components (1), such as modules, libraries, frameworks, or even complete applications, to develop new software systems. This approach allows developers to save time and effort during development while also enhancing overall software quality and maintainability. In the current software development environment, it is crucial to carefully choose components for reuse to have successful and sustainable software reuse practices. To reap the maximum advantages from software reuse, developers and organizations should take into account factors such as the suitability of components for specific projects.

The compatibility of open-source components differs across key domains (2). For instance, in the systems infrastructure domain, operating systems and databases are often highly compatible and can be easily integrated into different software systems. In the personal productivity domain, there is a wide range of compatible tools for tasks like document editing, email management, and project collaboration. In the software development domain, open-source components are extensively used with interoperability ensured through community-driven standards and best practices.

Open-source components have gained popularity in different domains (3) due to their benefits, such as cost reduction and flexibility. These components are created and maintained by a community of contributors and have diverse applications in operating systems, databases, and software development. In the field of software development specifically, widely used open-source components include programming languages like Python and JavaScript, version control systems such as Git, and development frameworks like Django and React.

Node.js is widely used by developers as a runtime environment for executing JavaScript code on the server-side (4). It provides an effective and streamlined foundation for creating scalable network applications because of its event-driven and non-blocking I/O model. A key benefit of Node.js is its package manager (5), which facilitates software reuse

* Corresponding author: Tolulope Amos Awoniyi

through dependency management. Dependencies are critical in determining relationships and interactions between different components within a software system. Additionally, successful software reuse in large-scale systems depends on factors such as the reliability of reusable components and the strategies employed for their management and implementation.

Managing dependencies in Node Package Manager (npm) presents challenges such as ensuring compatibility and resolving conflicts between different package versions. The security impact of npm dependencies is substantial (6), especially considering that Node.js depends extensively on its package manager, npm, which is a major part of its software environment. As a result, developers frequently rely on many third-party packages, even for fundamental features, leading to a substantial vulnerability. Additionally, the Node.js/npm community's inclination towards numerous small packages adds to the extensive number of dependencies (6).

It is now well established that the software world is rapidly changing. The success of software development projects relies heavily on the careful selection and management of software components and their dependencies. The choice of components and how they interact can significantly impact the functionality, performance, and stability of a software system. Neglecting to consider these factors or making poor decisions in component selection can result in incomplete or faulty systems. A notable example in 2016 is the deletion of a small package with just 11 lines of code from the npm registry caused widespread issues among numerous packages (7). However, current practices for selecting software components often overlook important evaluation criteria such as suitability and completeness. Additionally, there is a scarcity of effective models and measures that can aid in assessing the quality attributes of software components when making selections.

The study explores the survival of software reuse in terms of Node.js packages, focusing on component selection and dependency. It aims to identify the key factors that influence component selection in the context of node.js packages and examine the process of managing dependencies in the Node.js ecosystem.

2. Literature review

This section explores the most pertinent research for this study. Previous studies are divided into two primary categories: research focused on component selection and work relevant to package dependencies.

2.1. Software Reuse

The literature on Opportunities for Software Reuse provides a historical overview of software reuse, discussing its development and emerging trends in the field. It also examines the origins of software reuse that emerged during the software crisis of the late 1960s and its subsequent evolution as a formal discipline. This paper identifies four key categories of software reuse: domain analysis, generators, and transformation models; components, languages, and code reuse; and reuse cost models. In addition, they present findings from an industry survey that highlights an increasing inclination toward reusing existing software along with a rise in repository usage to facilitate such reuse (8). (9) proposed a machine learning algorithm that improves performance metrics for testing the reusability of software components, while (10) examined software reuse in Colombia, identifying barriers to adoption and factors influencing success.

Drawing on an extensive range of sources, the authors set out a new concept of software reuse within the context of opportunistic design (11). This approach involves integrating evolving components in a non-systematic haphazard manner, effectively leading to what the authors term “tip-of-the-iceberg” in software systems that primarily rely on third-party components rather than those developed by the primary team. To illustrate this phenomenon, the authors present a case study involving an industrial Internet of Things project where less than 5% of the source code originated from their team, with most coming from open-source components obtained from external sources. Overall, these papers emphasize the significance of software reuse while providing various strategies and perspectives for its implementation and enhancement.

3. Review of related works

In an exploration of npm package selection, a survey examines the factors that influence developers from both user and contributor perspectives. From the user point of view, it delves into aspects such as package usage, while from the contributor perspective, it focuses on contribution guidelines and package building/testing. The study highlights differing priorities between users and contributors, indicating a need for additional metrics to capture these perspectives. Relevant features are identified and analyzed for each viewpoint, with correlations investigated among

these features. Additionally, the paper evaluates the "runnability" (including installation, building, and testing) of both npm packages themselves and their dependencies. Ultimately, the research aims to provide valuable insights for package recommendation systems and encourage consideration of new metrics in measuring package quality (12). A simplification approach (2) enhances the effectiveness of the npm search engine by streamlining its components while maintaining package rankings. The approach involves simplification across various metrics categorized as quality, popularity, and maintenance.

The work of (13) provides an in-depth analysis of the characteristics of popular packages within the npm ecosystem, The study incorporates both qualitative and quantitative methods, including surveys with 118 JavaScript developers and regression analysis on a dataset comprising 2,527 npm packages. The findings highlight various factors that contribute to the selection of highly-popular packages, such as comprehensive documentation, a large number of stars on GitHub, high download rates, and minimal security vulnerabilities. These insights offer valuable guidance for package developers, ecosystem maintainers, and users in maximizing their usage of npm packages.

A recent study of JavaScript static analysis tools for vulnerability detection in Node.js Packages (14), evaluates the effectiveness of JavaScript vulnerability detection tools for analyzing Node.js packages. The authors curated a dataset containing 957 vulnerabilities in Node.js code, which was used to test nine different static code analysis tools. The results revealed that many significant vulnerabilities from the Open Worldwide Application Security Project (OWASP) Top-10 list went undetected by all tested tools. The most effective tools only identified up to 57.6% of the vulnerabilities with a precision rate as low as 0.11%. These findings highlight the need for improved detection methods and provide valuable insights for future research in this field. The publicly available curated dataset can serve as a benchmark for evaluating vulnerability detection techniques in Node.js applications.

One study (15) investigates the role of updating dependencies and explores who is responsible for these updates. Through analyzing 89,393 npm packages, the authors examine repository data to determine the individuals or authors responsible for updating libraries and whether the distribution of responsibility impacts which libraries are updated. The results indicate that 64.24% of packages are maintained by a single dependency author responsible for updating their dependencies. Furthermore, a relationship exists between the number of dependency authors and changes in dependencies, implying that having more responsible developers raises the probability of updating these external components. Finally, npm packages with only one dependency author update different libraries compared to those with multiple dependency authors.

4. Factors that influence component selection

4.1. Documentation

Documentation plays a crucial role in the selection of software packages by users. In addition to well-designed and reusable code, package users consider recent releases and high-quality documentation as important factors. Good documentation is essential for understanding how to use a software system effectively in projects. While some packages have addressed this need by providing unofficial documentation such as blog posts, it is still necessary for packages to prioritize comprehensive and official documentation for user convenience (12).

4.2. Crowdsourcing

Developers frequently rely on the Internet as a source of information when writing code, often placing more trust in anonymous commenters and popularity ratings than their colleagues. In fact, (16) maintains that seeking advice online is a considerably preferred method of finding components compared to asking a colleague for assistance.

4.3. GitHub Activities

The frequency of package releases can indicate how well maintained the package is. A higher number of releases suggests that the package is actively updated and maintained(13). Developers may be discouraged from selecting and using an npm package if it depends on vulnerable dependencies, as these vulnerabilities can raise concerns about the security of the package (17,18).

The frequency of commits in the package repository is an important factor for developers when selecting packages to use. Previous studies have shown that a higher number of commits indicates a more active and well-maintained package, making it favored by developers, another significant factor is the number of downloads, Higher download numbers for a package suggest that it is widely chosen and used (17). The popularity of a package on GitHub is often

determined by the number of projects that utilize it. Packages with a large user base tend to attract more developers to adopt them (18).

5. Dependence management

In the context of software programs or applications, a dependency is an external resource or library that is necessary for proper functioning. Dependencies can be packages or modules that are required by the application to run effectively and without any issues. Similarly in the context of npm dependencies, a dependency refers to a package or module that is necessary for the proper functioning of a Node.js application. These dependencies can be installed and managed using the npm package manager, which provides developers with an efficient way to incorporate and update external libraries in their projects.

Node.js applications rely on the Node Package Manager (npm) to handle their dependencies (19). The npm registry houses over 4 million packages and has experienced a rapid increase in package availability compared to other programming languages (20). Node.js applications, dependencies are typically specified in a package.json file using JSON format. This file lists the dependencies and their versioning constraints. The versioning constraint is a convention that dictates which version(s) of a package an application can depend on. It can be either static, specifying a specific version (for example, "P:1.0.0"), or dynamic, defining a range of versions (for example "P:>1.0.0"). Developers commonly use dynamic versioning constraints to install the latest updates and security fixes for packages. In this case, the resolved version corresponds to the most recent installable version that meets the constraint (21).

In the package.json file, Node.js applications can define two categories of dependencies: development and production. Development dependencies are exclusively installed on development environments, meaning any issues that arise from them do not affect production environments. Conversely, production dependencies (also known as runtime dependencies) are installed on both production and development environments (22).

6. Discussion

One of the main findings of this study highlights the importance of carefully choosing software components to ensure long-term durability and adaptability. The Node.js ecosystem offers a wide range of open-source packages, this study emphasizes that developers should consider factors like community support and maintenance frequency when selecting components.

Efficiently managing dependencies is crucial for maintaining the stability and security of software systems. This study emphasizes that failing to properly handle version updates and outdated functionality can introduce considerable risks. To mitigate these risks, developers need to be proactive in their approach to managing Node.js Packages and their corresponding dependencies. By staying attentive and adhering to best practices, developers can effectively safeguard their software from potential issues.

7. Conclusion

The success of software reuse in Node.js relies on developers making informed choices regarding component selection and dependencies, actively managing their dependencies, and keeping up with industry trends. As we navigate the ever-changing landscape of software development, it is crucial to understand the interdependence between careful component selection, effective dependency management practices, and long-term sustainability.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Esther Rajakumari K. Towards A Novel Conceptual Framework for Analyzing Code Clones to Assist in Software Development and Software Reuse. In: 2020 4th International Conference on Intelligent Computing and Control

Systems (ICICCS) [Internet]. IEEE; 2020 [cited 2023 Oct 27]. p. 105–11. Available from: https://ieeexplore.ieee.org/document/9120965/

[2] Abdellatif A, Zeng Y, Elshafei M, Shihab E, Shang W. Simplifying the Search of npm Packages. Inf Softw Technol. 2020 Oct 1;126:106365.

[3] Mikkonen T, Taivalsaari A. Software reuse in the era of opportunistic design. IEEE Softw. 2019 May 1;36(3):105–11.

[4] Node.js [Internet]. 2023 [cited 2023 Oct 27]. Available from: https://nodejs.org/en

[5] npm | Home [Internet]. 2023 [cited 2023 Oct 27]. Available from: https://www.npmjs.com/

[6] Ferreira G, Jia L, Sunshine J, Kastner C. Containing malicious package updates in npm with a lightweight permission system. Proceedings - International Conference on Software Engineering. 2021 May 1;1334–46.

[7] Hafner A, Mur A, Bernard J. Node package manager's dependency network robustness. 2021;

[8] Capilla R, Gallina B, Cetina C, Favaro J. Opportunities for software reuse in an uncertain world: From past to emerging trends. Journal of Software: Evolution and Process [Internet]. 2019 Aug 1 [cited 2023 Nov 5];31(8):e2217. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/smr.2217

[9] Sandhu AK, Batth RS. Software reuse analytics using integrated random forest and gradient boosting machine learning algorithm. Softw Pract Exp [Internet]. 2021 Apr 1 [cited 2023 Nov 9];51(4):735–47. Available from: https://onlinelibrary.wiley.com/doi/full/10.1002/spe.2921

[10] Restrepo-Gutiérrez LF, Suescún-Monsalve E, Mazo R, Vallejo-Correa PA, Correa D. "Snapshot" of the State of Software Reuse in Colombia. Revista Científica [Internet]. 2022 May 1 [cited 2023 Nov 9];44(2):242–56. Available from: https://revistas.udistrital.edu.co/index.php/revcie/article/view/18389

[11] Mikkonen T, Taivalsaari A. Software reuse in the era of opportunistic design. IEEE Softw. 2019 May 1;36(3):105–11.

[12] Chinthanet B, Reid B, Treude C, Wagner M, Kula RG, Ishio T, et al. What makes a good Node.js package? Investigating Users, Contributors, and Runnability. 2021 Jun 23 [cited 2023 Oct 30]; Available from: http://arxiv.org/abs/2106.12239

[13] Mujahid S, Abdalkareem R, Shihab E. What are the characteristics of highly-selected packages? A case study on the npm ecosystem. Journal of Systems and Software. 2023 Apr 1;198.

[14] Brito T, Ferreira M, Monteiro M, Lopes P, Barros M, Santos JF, et al. Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages. IEEE Trans Reliab [Internet]. 2023 [cited 2023 Oct 31]; Available from: https://github.com/VulcaN-Study/Supplementary-Material

[15] Maeprasart V, Ikegami A, Kula RG, Matsumoto K. Which Dependency was Updated? Exploring Who Changes Dependencies in npm packages. Proceedings - 22nd IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2021-Fall. 2021;258–61.

[16] Mäkitalo N, Taivalsaari A, Kiviluoto A, Mikkonen T, Capilla R. On opportunistic software reuse. Computing [Internet]. 2020 Nov 1 [cited 2023 Nov 23];102(11):2385–408. Available from: https://link.springer.com/article/10.1007/s00607-020-00833-6

[17] Abdellatif A, Zeng Y, Elshafei M, Shihab E, Shang W. Simplifying the Search of npm Packages. Inf Softw Technol. 2020 Oct 1;126:106365.

[18] Abdalkareem R, Oda V, Mujahid S, Shihab E. On the impact of using trivial packages: an empirical case study on npm and PyPI. Empir Softw Eng [Internet]. 2020 Mar 1 [cited 2023 Nov 23];25(2):1168–204. Available from: https://link.springer.com/article/10.1007/s10664-019-09792-9

[19] registry | npm Docs [Internet]. 2023 [cited 2023 Nov 5]. Available from: https://docs.npmjs.com/cli/v10/using-npm/registry

[20] npm - Libraries.io [Internet]. 2023 [cited 2023 Nov 5]. Available from: https://libraries.io/NPM

[21] Cogo FR, Oliva GA, Hassan AE. An Empirical Study of Dependency Downgrades in the npm Ecosystem. IEEE Transactions on Software Engineering. 2021 Nov 1;47(11):2457–70.

[22] Alfadel M, Elias Costa D, Mokhallalati M, Shihab E, Member S, Adams B. On the Threat of npm Vulnerable Dependencies in Node.js Applications. 2020 Sep 18 [cited 2023 Nov 5]; Available from: http://arxiv.org/abs/2009.09019