O IMPACTO DA UTILIZAÇÃO DE PACOTES DE TERCEIROS NO DESENVOLVIMENTO COM JAVASCRIPT: UMA REVISÃO SISTEMÁTICA DA LITERATURA

Caique Andrey Fuzaite Nascimento
Giovani Fonseca Ravagnani Disperati
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Resumo

A adoção de bibliotecas de terceiros para o desenvolvimento de projetos de software no ecossistema do Javascript é uma prática comum, a utilização desses pacotes de terceiros permite aos desenvolvedores, acesso a funcionalidades que exigiriam tempo e esforço para serem construídos. Através de gerenciadores de pacotes como NPM ou Yarn, a comunidade pode disponibilizar, sustentar ou até mesmo criar novas funcionalidades para estas bibliotecas. Porém, o que se percebe é que nem sempre os desenvolvedores que utilizam esses pacotes em seus projetos se mantêm atualizados em relação às bibliotecas. Com base nesse fator da utilização dos desenvolvedores, esse projeto tem como intuito identificar, avaliar e sintetizar estudos que abordem os impactos da defasagem de atualizações desses pacotes por parte dos desenvolvedores que as utilizam. Como essa defasagem de atualização afeta a segurança, a confiabilidade, o desempenho e até mesmo a qualidade dos softwares que as utilizam.

Palavras-chave: Defasagem. Pacotes terceiros. Javascript. Confiabilidade. Desempenho.

1. Introdução

Nos últimos anos, o ecossistema JavaScript tem visto um crescimento exponencial de pacotes de terceiros disponíveis por meio de gerenciadores de pacotes como Node Package Manager (NPM) e Yarn. O reúso de código realizado pelos desenvolvedores por meio de pacotes é uma prática comum que, agiliza a construção do software, viabiliza a implementação de funcionalidades, desde pequenos pacotes "triviais" que acabam se tornando parte importante do produto até grandes pacotes que moldam toda arquitetura dos softwares ao redor dessa dependência.

Por outro lado, segundo Chowdhury et al., 2021, à medida que a dependência de pacotes de terceiros se tornou comum, surgem questões críticas relacionadas ao desempenho e à confiabilidade das aplicações que os utilizam.

Um fator importante dessa visão é que, é necessário voltar o olhar para a atualização desses pacotes por parte dos desenvolvedores que os utilizam, a comunidade que disponibiliza as bibliotecas geralmente possui uma rotina de atualizações dessas dependências, versões novas para melhorar a segurança, criação de novas funcionalidades dos pacotes, correções de erros etc.

Tendo em vista esse cenário, caso os desenvolvedores não mantenham os pacotes terceiros atualizados em conjunto as versões disponibilizadas pelos gerenciadores, quais são os impactos que essa defasagem pode acarretar aos softwares?

Neste documento, apresentaremos uma análise resultante de uma revisão sistemática da literatura de estudos que, se concentram na utilização de bibliotecas de terceiros no desenvolvimento com JavaScript, o foco deste estudo é analisar os impactos que a defasagem na atualização dos pacotes consumidos pelos desenvolvedores pode causar, como a segurança, confiabilidade, desempenho e a qualidade do código das aplicações que as utilizam podem ser afetadas.

2. Materiais e Métodos

Nesta seção será abordada a metodologia de pesquisa utilizada para realização deste projeto, quais as abordagens utilizadas para evidenciar e buscar respostas para as perguntas deste trabalho.

Como descrito anteriormente, esta revisão sistemática da literatura busca analisar e evidenciar artigos, estudos de caso, e quaisquer outros materiais que abordem sobre os impactos da utilização de pacotes desatualizados por meio dos desenvolvedores de software.

Algumas fases foram estruturadas junto ao orientador para o desenvolvimento deste trabalho, a saber: definição das perguntas de pesquisa; Criação de strings de busca nas bases bibliográficas; Pesquisa prévia nas bases bibliográficas; Pesquisa inicial de artigos; Elaboração dos resultados e discussão; Escrita das conclusões.

As perguntas de pesquisa foram separadas em três para que a busca por cada uma das características possa ser pesquisada nas bases bibliográficas de forma independente. As perguntas são:

- Qual o impacto da defasagem na atualização de pacotes de terceiros no ecossistema JavaScript e como isso afeta a segurança?
- Qual o impacto da defasagem na atualização de pacotes de terceiros no ecossistema JavaScript e como isso afeta a confiabilidade das aplicações que utilizam esses pacotes?

- Quais os impactos da utilização de pacotes terceiros na qualidade do código e no desempenho das aplicações utilizando Javascript?

Para auxiliar na busca dos artigos científicos, baseando-se nas perguntas de pesquisa as strings de busca utilizadas foram:

- Outdated third-party packages Javascript
- Third-party packages Javascript
- Pacotes terceiros Javascript
- NPM outdated packages
- Impactos pacotes terceiros Javascript
- Client packages Javascript

O motivo da utilização dessas strings, é que após pesquisas iniciais foi possível identificar um padrão nos artigos relacionados ao tema. Na grande maioria dos trabalhos, as palavras em inglês “Outdated” que em português pode ser traduzida para desatualizada, foram bastante utilizadas. Assim como os termos “third-party” e “client package”, que em Português significam “de terceiros” e “pacotes de clientes” respectivamente, foram diversamente aplicados para representar as bibliotecas de terceiros consumidas pelos desenvolvedores.

Através das strings de busca, essas foram as bases bibliográficas que resultaram em artigos que se encaixavam inicialmente ao objetivo deste trabalho:

- Capes periódicos
- IEEE-Xplore
- Google Acadêmico
- ACM Digital Library

Inicialmente um levantamento dos artigos foi feito nas bases bibliográficas, a escolha dos artigos foi feita após uma leitura do resumo e conclusões de cada um dos 12 artigos encontrados, e que poderiam ser utilizados pela revisão.

Após a escolha inicial dos artigos, e a leitura completa de como foi trabalhado cada um dos artigos, qual o objetivo de cada um e o resultado obtido, um resumo de cada um deles foi feito para que pudesse dar início a revisão sistemática dos artigos que respondem às perguntas de pesquisa.

Após essa análise mais profunda, 4 artigos foram mantidos para esta revisão até o momento, pois a pesquisa realizada nestes trabalhos auxilia a entender e embasar as perguntas de pesquisa deste trabalho. Os artigos selecionados foram:

- An Empirical Study of Dependency Downgrades in the npm Ecosystem (COGO et al., 2021)
- On the Impact of Outdated and Vulnerable Javascript Packages in Docker Images (ZEROUALI et al., 2019)
- Software Reuse and Evolution in JavaScript Applications (TERZI et al., 2022)
- Towards smoother library migrations: A look at vulnerable dependency migrations at function level for npm javascript packages (ZAPATA et al., 2018)

Através desses artigos, foi possível analisar alguns pontos de vista que proporcionaram os resultados deste projeto, que visa disponibilizar uma análise desses materiais para futuras pesquisas.

Após a finalização dos resumos e resultados, será possível analisar se esses artigos foram capazes de responder todas as perguntas de pesquisa estabelecidas. Com a finalização dessas etapas, será viável definir uma análise baseada no que foi encontrado para que essa revisão sistemática da literatura possa auxiliar em futuras pesquisas relacionadas ao tema.

3. Resultados e Discussão

No decorrer da pesquisa, foram feitas várias análises de materiais que abordem as perguntas de pesquisa desta revisão. Os resultados deste estudo fornecem uma análise dos resultados obtidos por alguns pesquisadores do tema.

Segundo Zapata et al. (2018), apontam que estudos recentes mostram que muitos projetos defasados de atualização não chamam de fato a função afetada e estão a salvo da vulnerabilidade. Eles se referem a esses projetos como sendo limpos (ou seja, eles não executam o código afetado nos softwares que os utilizam).

No estudo realizado por Zapata et al. (2018), examinaram manualmente um total de 60 projetos, investigando três casos de vulnerabilidades de alta prioridade para entender como os projetos seguros e inseguros lidam com a migração para versões de dependência mais seguras.

Os resultados do estudo exploratório sugerem que até 73,3% dos clientes desatualizados da amostra estavam de fato a salvo da ameaça de vulnerabilidade (ou seja, não executaram o código vulnerável). Ainda segundo os resultados do estudo fornecem evidência de que muitos dos projetos desatualizados estão livres da vulnerabilidade. Essa percepção é uma indicação de que é necessária mais análise em nível de função para apoiar essa afirmação.

Entender se a vulnerabilidade afeta ou não o código do cliente, ajudará os desenvolvedores a planejar melhor suas migrações de bibliotecas (ZAPATA et al., 2018).

Ao analisar o estudo realizado por Zerouali et al. (2019), que apresenta uma análise empírica do uso de pacotes JavaScript em imagens oficiais do Docker, baseadas na imagem de Node.js para as distribuições Debian e Alpine Linux. Ao todo 961 imagens únicas foram recuperadas e analisadas contra 1.099 relatórios de segurança extraídos do Snyk.io, um registro bem conhecido de vulnerabilidades para pacotes JavaScript npm.

Para capturar a base de vulnerabilidades a serem investigadas eles separaram da seguinte forma: Cada relatório de segurança contém informações de vulnerabilidade sobre o pacote afetado, o intervalo de versões afetadas, sua gravidade, seu tipo, a data de sua divulgação e a data em que foi publicado no banco de dados (ZEROUALI et al., 2019).

Depois de identificar quais pacotes npm instalados são afetados pelas vulnerabilidades relatadas no Snyk.io, descobrimos que, de 1.099 vulnerabilidades conhecidas, apenas 74 estão afetando os pacotes npm em imagens do Docker. Esse número de vulnerabilidades está afetando 3,7% de todos os pacotes instalados. No entanto, todas as imagens são afetadas por esses pacotes vulneráveis. Descobriram que a maioria das vulnerabilidades (54%) foram divulgadas, sendo elas 40% vulnerabilidades de gravidade média, 35% de gravidade alta e 25% contendo uma gravidade baixa (ZEROUALI et al., 2019).

Por fim Zerouali et al. (2019) concluem que idealmente, as imagens do Docker devem depender da versão mais recente de seus pacotes para se beneficiar das funcionalidades, atualizações de segurança e correções de bugs mais recentes. No entanto, pode ser possível que os mantenedores prefiram dar maior prioridade à compatibilidade com versões anteriores e à estabilidade em vez de atualizar seu software, já que essa última exigiria um esforço considerável.

Quando calculamos o atraso técnico na data da última atualização de cada imagem, descobrimos que a maioria dos pacotes estão atualizados e que o atraso técnico é baixo. Por outro lado, na data de nossa extração de dados, detectamos uma mudança importante, pois o atraso técnico aumenta com uma versão principal. Essa defasagem pode induzir a presença de vulnerabilidades de segurança (ZEROUALI et al., 2019).

O estudo realizado por Cogo et al. (2021) analisa downgrades de dependências em ecossistemas de software. Os pacotes em um ecossistema de software geralmente lançam versões com correções de bugs, novas funcionalidades e aprimoramentos de segurança. Portanto, atualizar a versão do provedor é uma tarefa de manutenção importante para os pacotes clientes. Apesar do número de investigações sobre atualizações de dependência, segundo esse artigo, faltam estudos sobre downgrades de dependência em ecossistemas de software. O downgrade indica que a versão adotada de um pacote do provedor não é adequada para o pacote do cliente em um determinado momento (COGO et al., 2021).

Apesar de ser uma indicação natural de problemas, de acordo com Cogo et al. (2021) os downgrades podem ser uma solução simples e rápida para problemas específicos que surgem quando um pacote de provedor é atualizado. Essa técnica aumenta o atraso técnico dos pacotes de clientes.

O atraso técnico representa o grau em que os pacotes de clientes estão perdendo os recursos e as correções de bugs mais recentes de um provedor. Medir o atraso técnico que é introduzido quando um downgrade é realizado deve ajudar os profissionais a entender os efeitos colaterais dessa solução alternativa (COGO et al., 2021).

O trabalho conduzido por Terzi et al. (2022) tem como objetivo principal, analisar a evolução das dependências das bibliotecas de terceiros no contexto dos aplicativos desenvolvidos com Javascript. A pesquisa conduzida por eles investigou 20 versões de 86 projetos com Javascript hospedados no GitHub.

Terzi et al. (2022) estabeleceram duas perguntas de pesquisa para serem respondidas, a primeira delas visa examinar o número de bibliotecas de dependências que foram adicionadas, removidas ou sofreram algum tipo de manutenção em versões diferentes. A principal informação que essa questão pode responder é que ela pode indicar se as dependências daquela biblioteca estão estáveis, durante o ciclo de vida desse projeto, ou se as escolhas dos mantenedores da biblioteca são frequentemente consideradas. O que pode garantir maior confiabilidade para os desenvolvedores que queiram utilizar dessas bibliotecas.

A segunda pergunta de pesquisa de Terzi et al. (2022) busca analisar se há uma tendência de atualização das dependências de bibliotecas. Para Terzi et al. (2022), é necessário explorar até que ponto as atualizações de bibliotecas de terceiros são atualizadas. Ainda segundo eles, dependências de bibliotecas desatualizadas podem envolver riscos potenciais relacionados à incompatibilidade de API, ameaças à segurança, correções de bugs e etc. Portanto, a resposta dessa pergunta ajudará os desenvolvedores de Javascript a organizar e programar futuras atualizações de bibliotecas de terceiros com base em evidências.

Ao final de sua pesquisa, Terzi et al. (2022) levanta algumas discussões captadas por suas métricas, o número de dependências de biblioteca permanece relativamente estável apresentando um pequeno aumento durante a evolução dos aplicativos. Em geral, os desenvolvedores de Javascript reutilizam um número pré-definido de bibliotecas que implementam funcionalidades específicas. Novas bibliotecas podem ser adicionadas durante a evolução do projeto, mas esse aumento permanece estável, pois eles não observam picos repentinos na intensidade da reutilização.

Essa descoberta vai de encontro com o trabalho de Zerouali et al. (2019), que concluíram que no nível do pacote, o número de dependências raramente mudou. Eles apontam que as dependências são adicionadas ou removidas principalmente nas versões principais, mas eles não determinaram se há uma alteração no número total de dependências (TERZI et al., 2022).

O resultado deste projeto mostra que os desenvolvedores, em geral, preservam a estabilidade organizacional dos aplicativos em termos de dependências de terceiros, mantendo sob controle o esforço de manutenção necessário para gerenciar as dependências (Terzi et al., 2022).

Por fim, Terzi et al. (2022) conclui que na maioria dos casos, os desenvolvedores preferem manter as dependências da biblioteca em versões desatualizadas, provavelmente em uma tentativa de reduzir o risco de incompatibilidades que uma nova versão pode causar.

4. Considerações Finais

Com base nos resultados e discussões apresentados pelos estudos de Zapata et al. (2018), Zerouali et al. (2019), Cogo et al. (2021) e Terzi et al. (2022), é possível extrair diversas conclusões significativas sobre a gestão de dependências em projetos de software com Javascript, e seus impactos na segurança, confiabilidade, qualidade do código e desempenho das aplicações.

Primeiramente, a análise de Zapata et al. (2018) revela que uma proporção considerável de projetos desatualizados pode, na verdade, estar livre de vulnerabilidades, sugerindo a necessidade de análises mais detalhadas em nível de função para embasar decisões de migração de bibliotecas. Por outro lado, o estudo de Zerouali et al. (2019) destaca que, embora uma quantidade significativa de vulnerabilidades afete pacotes em imagens do Docker, a maioria dos pacotes está atualizada, mas a defasagem técnica aumenta com versões mais antigas, potencialmente introduzindo vulnerabilidades de segurança.

Além disso, a pesquisa de Cogo et al. (2021) ressalta a importância dos estudos sobre downgrades de dependências, demonstrando que, embora os downgrades possam ser uma solução rápida para problemas específicos com o software, eles aumentam o atraso técnico dos pacotes de clientes, afetando a capacidade de aproveitar recursos e correções de bugs mais recentes. Por fim, o estudo de Terzi et al. (2022) destaca que os desenvolvedores tendem a preservar a estabilidade dos aplicativos em termos de pacotes de terceiros, mantendo sob controle o esforço de manutenção necessário para gerenciar as dependências, mesmo que isso signifique manter versões desatualizadas para mitigar riscos de incompatibilidade. Quanto aos impactos da utilização de pacotes de terceiros na qualidade do código e no desempenho das aplicações, no entanto, essa prática pode comprometer a qualidade do código e o desempenho, uma vez que versões desatualizadas podem conter falhas de segurança e não aproveitar melhorias de desempenho e otimizações de código.

Em suma, esses estudos oferecem insights valiosos para os desenvolvedores, pontos que podem ser destacados são que é necessária uma abordagem equilibrada na utilização de dependências, é importante considerar a manutenibilidade desses pacotes, planejar possíveis atualizações fornecidas pelos mantenedores das bibliotecas, mas também traçar planos de ações caso os distribuidores de determinado pacote deixem de mantê-lo. Por fim, é sempre válido uma análise caso a caso, para que essa manutenção seja feita da forma menos danosa possível, tendo em vista que não necessariamente ela seja algo obrigatório.

5. Referências

Chowdhury, M. A. R., Abdalkareem, R., Shihab, E., and Adams, B. (2021). On the untriviality of trivial packages: An empirical study of npm javascript packages. IEEE Transactions on Software Engineering, 48(8):2695–2708.

Cogo, F. R., Oliva, G. A., and Hassan, A. E. (2021). An empirical study of dependency downgrades in the npm ecosystem. IEEE Transactions on Software Engineering, 47(11):2457–2470.

Terzi, A., Christou, O., Bibi, S., and Angelidis, P. (2022). Software reuse and evolution in javascript applications. In 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA),pages 263–269. IEEE.

Zapata, R. E., Kula, R. G., Chinthanet, B., Ishio, T., Matsumoto, K., and Ihara, A. (2018). Towards smoother library migrations: A look at vulnerable dependency migrations at function

level for npm javascript packages. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 559–563. IEEE.

Zerouali, A., Cosentino, V., Mens, T., Robles, G., and Gonzalez-Barahona, J. M. (2019). On the impact of outdated and vulnerable javascript packages in docker images. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 619–623. IEEE.

ANEXOS

Artigos	**Métricas**
Cogo et al. (2021)	Obtiveram o arquivo de package.json de 461.548 pacotes do registro npm no período de 20 de dezembro de 2010 à 1º de julho de 2017. Depois de agrupar os downgrades de acordo com as releases dos clientes, extraíram uma amostra estatisticamente representativa com 95% de nível de confiança e 5% de intervalo de confiança, dessas releases foram (371 casos de 10.967).
Terzi et al. (2022)	Base: 86 projetos do Github. Análise: Tipos de dependências métricas: - Estatísticas descritivas, como mínima, máxima, mediana e desvio padrão; - Box-plots e Line Chart, para visualizar a distribuição dos valores de determinadas métricas; - Teste de tendência Man-Kendall para teste de hipóteses.
Zapata et al., (2018)	Examinou um total de 60 projetos, investigando três casos de vulnerabilidades de alta prioridade para entender como os projetos seguros e inseguros lidam com a migração para versões de dependência mais seguras. O estudo sugere que até 73,3% dos clientes desatualizados da amostra estavam de fato a salvo da ameaça de vulnerabilidade (ou seja, não executaram o código vulnerável).
Zerouali et al., (2019)	Extraíram as imagens disponíveis (ou seja, as mais recentes e marcadas) de 124 repositórios oficiais por meio do Docker. Inspecionaram remotamente usando o skopeo, uma ferramenta para coletar informações de registros de imagens remotas. 961 imagens exclusivas foram recuperadas e analisadas em relação a 1.099 relatórios de segurança extraídos do Snyk.io, um conhecido registro de vulnerabilidades para pacotes npm JavaScript.

Tabela 1 - Fonte: o Autor.

	Artigo			
Tipos de análise	Cogo et al. (2021)	Terzi et al. (2022)	Zapata et al., (2018)	Zerouali et al., (2019)
Confiabilidade	X	X	X	
Desempenho	X			
Segurança		X	X	X

Tabela 2 - Fonte: o Autor.

	Plataforma			
	Google Acadêmico	IEEE Xplore	ACM Digital Library	Capes Periódicos
String de busca	**Artigos Encontrados**			
Outdated third-party packages Javascript	16.100	3	622	4
Third party packages Javascript	17.000	42	4118	16

Pacotes terceiros Javascript	4740	0	34	1
NPM outdated packages	13	2	191	5
Impactos pacotes terceiros Javascript	3060	0	23	0
Client packages Javascript	2120	104	4664	39

Tabela 3 - Fonte: o Autor.