Gerenciamento Automatizado de Dependências: Um Estudo em Larga Escala sobre a Adoção do Dependabot

Gabriel Estevão,[1] **Letícia Lott,**[1] **Luana Fleury,**[1] **Yan Nalon,**[1]
Aline Brito,[2] **Joana Gabriela Souza,**[1] **Laerte Xavier**[1]

[1]Instituto de Ciências Exatas e Informática, Departamento de Engenharia de Software
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte, MG – Brazil

[2]Instituto de Ciências Exatas e Biológicas, Departamento de Computação
Universidade Federal de Ouro Preto (UFOP)
Ouro Preto, MG – Brazil

{gabriel.sobrinho,leticia.carvalho.13780938}@sga.pucminas.br,

{1400015,yan.nalon}@sga.pucminas.br,aline.brito@ufop.edu.br,

{joanasouza,laertexavier}@pucminas.br

Abstract. *Managing and updating external dependencies is a critical activity in software projects. Automated tools such as Dependabot have been widely adopted to support this process. This study investigates practices related to the adoption of Dependabot in GitHub projects. By mining approximately one million pull requests across 500 projects, we analyzed maintainers' behavior toward proposed updates, considering metrics such as notification frequency, acceptance rate, and time to integration. We observed that about 46.4% of the analyzed projects used Dependabot at some point during the studied period. The results also suggest that, although a significant portion of pull requests is accepted (59.5%), barriers to adoption remain. Furthermore, the time required to integrate a dependency update can range from a few days to several months, indicating that projects may remain vulnerable for extended periods.*

Resumo. *A gestão e atualização de dependências externas é uma atividade crítica em projetos de software. Ferramentas automatizadas como o Dependabot têm sido amplamente adotadas para auxiliar nesse processo. Este trabalho investiga práticas relacionadas à aceitação do Dependabot em projetos hospedados no GitHub. A partir da mineração de aproximadamente um milhão de pull requests, distribuídos em 500 projetos, analisou-se o comportamento dos mantenedores em relação às atualizações propostas, considerando métricas como frequência das notificações, taxa de aceitação e tempo até a integração. Observa-se que cerca de 46,4% dos projetos analisados utilizaram o Dependabot em algum momento ao longo do período estudado. Os resultados sugerem também que, embora uma parcela significativa de pull requests seja aceita (59,5%), ainda existem barreiras à adoção. Além disso, o tempo para integração de uma atualização de dependência pode variar entre poucos dias e vários meses, indicando que os projetos podem permanecer vulneráveis por longos períodos.*

1. Introdução

Bibliotecas e *frameworks* estão em constante evolução, com *releases* sendo disponibilizadas recorrentemente para incorporar novas funcionalidades, bem como para corrigir *bugs* e falhas de segurança [Hora and Valente 2015, Brito et al. 2018]. Entretanto, a literatura recente mostra que uma taxa significativa de bibliotecas e *frameworks* não é atualizada pelos sistemas clientes, mesmo após alertas de segurança [Kula et al. 2018, Garrett et al. 2019]. Em alguns casos, os desenvolvedores desconhecem as falhas reportadas nas versões que estão utilizando nos projetos. Por exemplo, em 2012, uma falha de segurança em uma biblioteca de criptografia chamada OpenSSL afetou milhares de servidores ao redor do mundo [Decan et al. 2018, Durumeric et al. 2014].

Para gerenciar dependências diretas e transitivas, os desenvolvedores podem fazer uso de ferramentas de monitoramento [Rebatchi et al. 2024, Chowdhury et al. 2025a]. Existem diversas ferramentas para realizar esta tarefa, algumas voltadas para linguagens de programação específicas como o Maven[1] e Gradle[2] usados para Java, npm[3] e Yarn[4] para JavaScript e Pip[5] para Python. Para além de ferramentas focadas em uma linguagem específica, pode-se citar o Dependabot, uma ferramenta gratuita de gerenciamento automático de dependências provida pelo GitHub[6], a plataforma de desenvolvimento de software mais usada no mundo[7].

Dada a importância de gerenciar as dependências de projetos, o GitHub desenvolveu o Dependabot que permite monitorar o uso de pacotes externos, alertando sobre defasagens e vulnerabilidades, isto é, dependências desatualizadas que não estão em conformidade com a versão recomendada. Dessa forma, o uso desta ferramenta tem como objetivo mitigar riscos de segurança, mantendo as dependências atualizadas.

Especificamente, o Dependabot monitora continuamente os repositórios que declaram as dependências externas do projeto, como por exemplo, o arquivo `package.json`. Quando uma vulnerabilidade é identificada, a ferramenta submete um *pull request* com as atualizações, alertando também os mantenedores do projeto via e-mail [Birari 2024].

Entretanto, apesar dos esforços de plataformas como o GitHub para auxiliar no processo de evolução e migração de dependências, ainda persistem desafios e dúvidas quanto à adoção e às práticas relacionadas a essas soluções [Jafari et al. 2022, Alfadel et al. 2021, Mens and Decan 2024]. Este estudo teve como objeto de estudo avaliar o contexto de uso de uma ferramenta de gerenciamento de dependências nativa de uma plataforma largamente usada pelo mundo, com o objetivo de realizar uma caracterização da adoção desta ferramenta. Para tanto, neste trabalho, investigam-se quatro questões de pesquisa:

- QP_1*:* Qual é a proporção de projetos que adotam o Dependabot?
- QP_2*:* Qual é a taxa de aceitação de *pull requests* abertos pelo Dependabot?

[1]https://maven.apache.org
[2]https://gradle.org
[3]https://www.npmjs.com
[4]https://yarnpkg.com
[5]https://pypi.org/project/pip
[6]docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts
[7]https://github.com/?locale=pt-br

- *QP*$_3$*:* Qual é a frequência de vulnerabilidades indentificadas pelo Dependabot por linguagem de programação?
- *QP*$_4$*:* Qual é o tempo de espera para resolução de *pull requests* abertos pelo Dependabot?

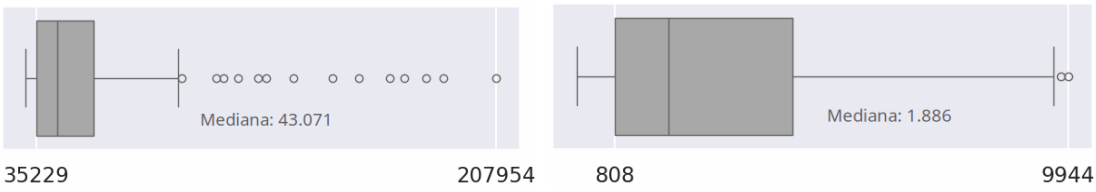
O restante deste artigo está organizado da seguinte forma. A Seção 2 descreve a metodologia adotada, detalhando os procedimentos utilizados na mineração e análise de *pull requests* gerados pelo Dependabot. Os resultados são apresentados na Seção 3 e discutidos na Seção 4. A Seção 5 apresenta os trabalhos relacionados. As ameaças à validade do estudo são discutidas na Seção 6. Por fim, a Seção 7 encerra o artigo com as considerações finais e sugestões para pesquisas futuras.

2. Metodologia

2.1. Seleção dos Repositórios

Neste trabalho, analisou-se 500 projetos GitHub populares, ordenados pelo número de estrelas, visto que é uma métrica frequentemente utilizada para indicar a relevância dos sistemas [Borges et al. 2016, Silva and Valente 2018].

A Figura 1 mostra a distribuição de estrelas e *pull requests* dos repositórios. O número de estrelas varia entre aproximadamente 31 mil e 207 mil, enquanto o número de *pull requests* varia entre 36 e 9.944 ocorrências. O conjunto de dados inclui sistemas de domínios distintos, como por exemplo `spring boot`, um *framework* Java para desenvolvimento de aplicações Spring; e `prisma`, um *Object-Relational Mapping* (ORM) para Node.js.



(a) *Estrelas* (b) *Pull Requests*

Figura 1. Distribuição de estrelas e *pull requests* por projeto

2.2. Extração de Métricas

Para cada um dos 500 projetos, foram coletados atributos de até 10 mil *pull requests* por projeto, resultando em um conjunto de dados composto por 909.879 instâncias.

Para a caracterização do uso da ferramenta nos repositórios, coletou-se dados relacionados à aceitação, autoria, modificações e tecnologias, providos pela *Application Programing Interface* (API) oficial do GitHub (`State`, `CreatedAt`, `ClosedAt`, `UpdatedAt`, `PublishedAt`, `MergedAt`, `LastEditedAt`, `RepositoryId`, `PrimaryLanguage` e `Author`). Em seguida, filtrou-se *pull requests* que foram criados pelo Dependabot, resultando em um conjunto de 40.292 *pull requests* (PR) distintos (aproximadamente 4,43% dos PR).

Utilizou-se a infraestrutura da *Amazon Web Services* para a mineração e processamento dos dados. O cálculo das métricas foi realizado com destaque para o *BigQuery*, da *Google Cloud Platform*, que possibilitou análises rápidas e eficientes.

3. Resultados

Nesta seção são apresentados os resultados considerando as quatro questões de pesquisa.

3.1. (QP1) Qual é a proporção de projetos que adotam o Dependabot?

Dentre os 500 projetos analisados, 232 (46,4%) receberam notificações do Dependabot ao menos uma vez ao longo do tempo. A Figura 2 mostra a distribuição de *pull requests* de autoria do Dependabot. Como se pode observar, os valores variam entre 13 e 3.321 *pull requests* por projeto, com mediana de 43. Apontando que há uma participação relativamente baixa de PRs feitas pelo Dependabot na maioria dos repositórios.

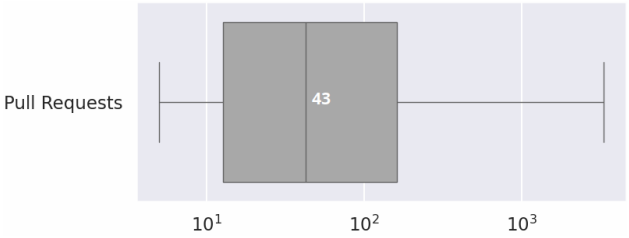


Figura 2. Distribuição de *pull requests* criados pelo Dependabot por projeto

O projeto `nest` — que representa um popular *framework* para JavaScript — é o que apresenta o maior número de alertas do Dependabot (3.321 ocorrências), sendo que cerca de 69% dessas contribuições foram aceitas (2.276 ocorrências com status *merged*).

A Figura 3 mostra um exemplo de *pull request* do Dependabot.[8] Neste caso, o *pull request* atualiza a versão de uma biblioteca gRPC do Node.js. O *pull request* foi aprovado pelos mantenedores em poucas horas, em junho de 2023.

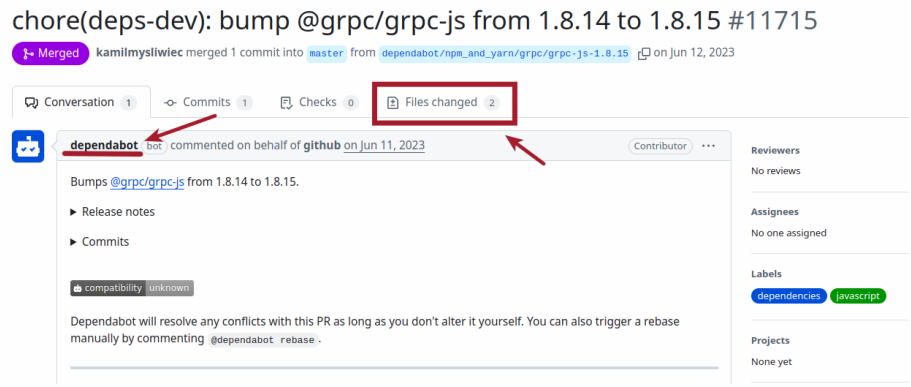


Figura 3. Exemplo de *pull request* submetido pelo Dependabot que foi aceito pelo desenvolvedor

Resumo: Dentre os 500 projetos analisados, 232 (46,4%) receberam pelo menos um *pull request* do Dependabot.

[8] `https://github.com/nestjs/nest/pull/11715`

3.2. (QP2) Qual é a taxa de aceitação de *pull requests* abertos pelo Dependabot?

Dentre os 909.879 *pull requests* analisados, 40.292 (4,4%) foram de autoria do Dependabot. A Tabela 1 mostra a distribuição de *pull requests* por *status*. Observa-se que mais de 50% das atualizações de dependências sugeridas pelo Dependabot são aceitas pelos desenvolvedores e subsequentemente integradas ao código fonte.

Pull requests	**Ocorrência**	**%**
Abertos	800	2,0
Aceitos	23.980	59,5
Rejeitados	15.512	38,5
Total	40.292	100

Tabela 1. Frequência de *pull requests* criados pelo Dependabot por status

A Figura 4 mostra a distribuição do percentual de aceitação por projeto, em relação ao número de *pull requests* recebidos. Os valores variam entre 22% e 100%, com uma mediana de 58,43% das recomendações sendo aceitas pelos mantenedores. Para 75% da amostra, cerca de 82% do número de *pull requests* abertos pelo Dependabot foram aceitos. Existem também projetos em que todos os *pull requests* submetidos pelo Dependabot foram aceitos, como por exemplo, o *framework* `autogen` da Microsoft, resultando em uma taxa de aprovação de 100%.

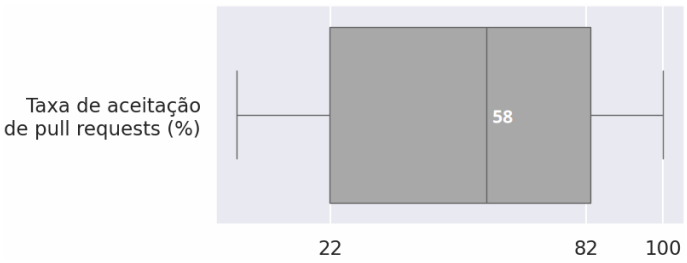


Figura 4. Distribuição da taxa de aceitação de *pull requests* do Dependabot por projeto

Resumo: A taxa de aceitação de *pull requests* submetidos pelo Dependabot varia entre 22% e 100%, com mediana de 58,43% por projeto.

3.3. (QP3) Qual é a frequência de vulnerabilidades identificadas pelo Dependabot por linguagem de programação?

A Figura 5 mostra a frequência de *pull requests* do Dependabot por linguagens de programação. Nota-se a prevalência das linguagens JavaScript e TypeScript, que correspondem às linguagens principais de cerca de 56% dos projetos para os quais o Dependabot submeteu *pull requests* (22.586 ocorrências). Em seguida, encontram-se Go (6.926, 17%) e Python (3.434, 9%). Outras linguanges de programação populares—como Java, C# e C—representam aproximadamente 24% da taxa de *pull requests*. Vale ressaltar que há ferramentas específicas para cada linguagem e que são muito adotadas em suas comunidades de desenvolvedores, como a Maven para Java, o que pode influenciar na não adoção do Dependabot.

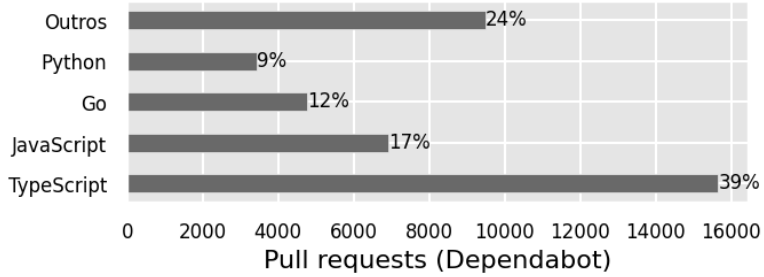


Figura 5. Distribuição de *pull requests* submetidos pelo Dependabot por linguagem de programação

A Figura 7 mostra um exemplo de atualização submetida pelo Dependabot no repositório do `parcel`, projeto que compreende um empacotador JavaScript popular.[9] A mudança envolve uma migração entre *major releases* de uma dependência denominada `simple-git` (uma interface para execução de comandos *git*). O *pull request* foi rejeitado pelos desenvolvedores do projeto, cerca de uma semana após a notificação, em 2023. Não foi possível analisar características de PRs mais aceitas ou recusadas pelos times de desenvolvimento.

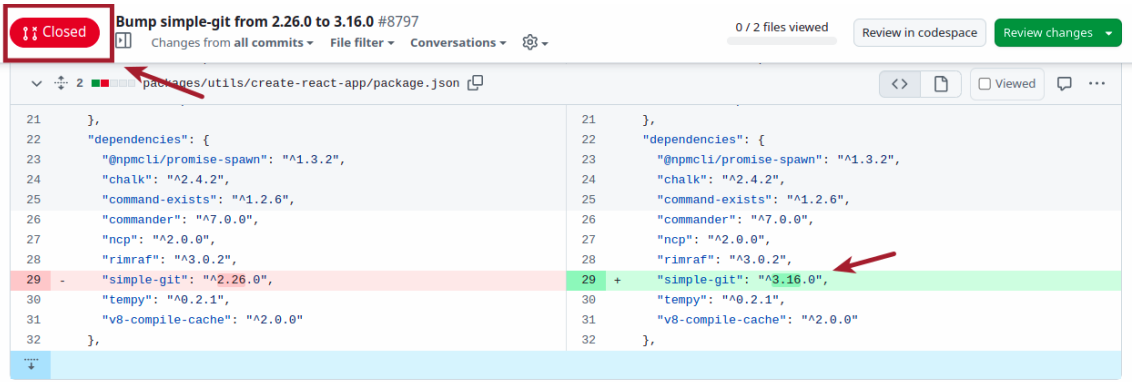


Figura 6. Exemplo de *pull request* rejeitado pelos desenvolvedores no Parcel

Resumo: As três linguagens de programação com maior número de alertas do Dependabot são: TypeScript (39%), JavaScript (17%) e Go (12%).

3.4. (QP4) Qual é o tempo de espera para resolução de *pull requests* abertos pelo Dependabot?

Nesta última questão de pesquisa, investiga-se o tempo entre a abertura de um *pull request* pelo Dependabot e sua aceitação — ou seja, integração ao código fonte — pelos mantenedores do projeto. Dentre os 40.292 *pull requests* analisados, 23.980 (59,5%) foram aceitos pelos desenvolvedores. Por sua vez, 16.294 destes *pull requests* foram aceitos em menos de 24 horas (67,9%) após a sua criação.

Em contraste, existiam 550 contribuições que somente foram aceitas 30 dias após sua submissão. A Figura 7 mostra a distribuição de *pull requests* por tempo de aceitação, com valores variando entre um dia e *merges* realizados após meses.

[9] `https://github.com/parcel-bundler/parcel/pull/8797/files`

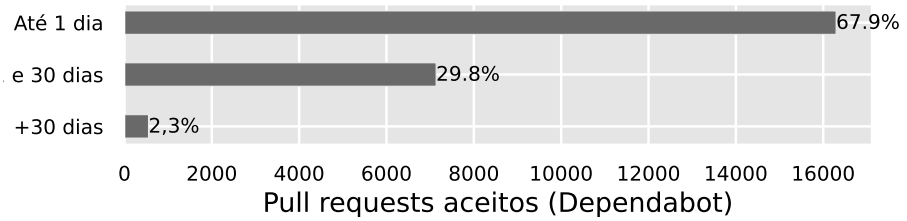


Figura 7. Tempo de aceitação de *pull requests* submetidos pelo Dependabot

O *outlier* do conjunto de dados corresponde a um *pull request* submetido ao projeto `prettier`, que foi aceito por um desenvolvedor após aproximadamente um ano (370 dias).[10] A mudança envolve uma atualização entre *major releases* do pacote `html-tag-names`, biblioteca para manipulação de listas de *tags* válidas em HTML. Entretanto, considerando a visão geral dos dados, a média de aceitação de *pull requests* é de cerca de dois dias (39 horas). Indicando que as PRs fetas pelo Dependabot geralmente são analisadas e aceitas rapidamente pelos desenvolvedores.

Resumo: Considerando os valores medianos, um *pull request* submetido pelo Dependabot é aceito em cerca de dois dias. Entretanto, existem contribuições que levam meses para serem incorporadas aos projetos.

4. Discussão dos Resultados

Adoção e desuso do Dependabot. Os resultados sugerem que o Dependabot vem sendo amplamente adotado em projetos populares. Por exemplo, 232 projetos receberam ao menos um *pull request* do Dependabot, e aproximadamente 60% das atualizações sugeridas são incorporadas aos projetos. De fato, bibliotecas e *frameworks* estão em constante evolução [Hora and Valente 2015, Bogart et al. 2016]; portanto é comum que novas *releases* sejam publicadas com frequência, para incorporar novas funcionalidades, realizar melhorias, ou corrigir *bugs*. Dessa forma, os resultados reforçam observações sobre a importância de ferramentas e técnicas que auxiliam no monitoramento, escolha, e atualização de dependências [Hora and Valente 2015, Kula et al. 2018, Mens and Decan 2024]. Adicionalmente, observou-se que o tempo de *merge* varia entre os projetos. A maioria dos projetos que usam a ferramenta integram *pull requests* rapidamente (média de 2 dias), enquanto outros rejeitam ou ignoram as sugestões.

Dependabot & *Breaking Changes em APIs.* Durante a inspeção dos resultados, observou-se que diversos *pull requests* submetidos pelo Dependabot foram rejeitados pelos mantenedores, especialmente aqueles que envolviam atualizações entre *major releases*. Esse comportamento pode estar relacionado com o esforço necessário para migração entre grandes versões de bibliotecas. Idealmente, bibliotecas e suas APIs devem ser estáveis e manter a retrocompatibilidade [Brito et al. 2018]. Entretanto, a literatura mostra que as convenções do versionamento semântico nem sempre são respeitadas pelos mantenedores de APIs, tornando o processo de atualização potencialmente arriscado. Os desenvolvedores frequentemente introduzem quebras de contratos (*breaking changes*) que afetam os sistemas clientes [Brito et al. 2018, Xavier et al. 2017, Wu et al. 2010,

[10] `https://github.com/prettier/prettier/pull/10612`

Robbes et al. 2012, McDonnell et al. 2013, Bogart et al. 2016]. Dessa forma, *pull requests* submetidos pelo Dependabot que envolvem *minor releases* também podem gerar impactos significativos nos sistemas clientes. Nesse contexto, a decisão de aceitar ou rejeitar uma atualização automática pode estar relacionada também à confiança dos mantenedores na qualidade e estabilidade da dependência utilizada.

5. Trabalhos Relacionados

Gerenciar as dependências de um projeto é uma tarefa importante na manutenção de *software* [Prana et al. 2021]. Considerando que à medida que o projeto cresce fica mais difícil de gerenciar as dependências, surgiram soluções de automação como o Dependabot. Nesse contexto, Mirhosseini e Parnin (2017) investigaram o impacto de *pull requests* automatizadas para incentivar atualizações de dependências desatualizadas. A análise de 7.470 projetos no GitHub revelou que o envio automático de notificações aumenta a frequência de atualizações, embora a taxa de *merge* seja frequentemente baixa devido à desconfiança dos desenvolvedores. Os resultados deste trabalho reforçam essas observações. Por exemplo, detectou-se que aproximadamente 15 mil *pull requests* (38%) gerados pelo Dependabot foram rejeitados (QP2).

Existem trabalhos que exploram os desafios na gestão de dependências em projetos [Mohayeji et al. 2023, Chowdhury et al. 2025b, Alfadel et al. 2021]. Por exemplo, Chowdhury et al. (2025) concentram-se no ecossistema Maven. Os resultados sugerem que projetos com menos dependências tendem a apresentar maior número de *missed releases* e maior defasagem temporal, enquanto sistemas com maior número de dependências demonstram práticas mais consistentes de atualização. Já Alfadel et al. (2021) investigaram *pull requests* de segurança gerados pelo Dependabot em 2.904 projetos JavaScript. Por meio de um modelo de regressão, os autores identificaram que a principal variável associada à velocidade de aceitação de *pull requests* é a experiência prévia do projeto com o Dependabot. He et al. (2023) destacaram desafios relacionados à aceitação de sugestões de atualização, como o disparo exaustivo de notificações. Para isto, analisaram como a ferramenta contribui para a redução de dívida técnica em 540.665 *pull requests* no GitHub. Mohayeji et al. (2018) analisaram cerca de 4 mil atualizações de segurança em projetos JavaScript, mostrando que os desenvolvedores podem priorizar as correções conforme a gravidade de vulnerabilidade. Entretanto, os estudos se concentraram em cenários e linguagens de programação específicas.

Kula et al. (2018) oferecem uma visão importante para compreender a gestão de dependências em projetos. Ao longo da investigação foram analisados 4.600 projetos do GitHub, resultando na descoberta de que 81,5% mantinham dependências desatualizadas mesmo após receberem alertas de segurança. Este estudo ajudou a motivar a presente pesquisa, servindo de base para investigar se atualmente com o uso de ferramentas como o Dependabot mudaram este panorama de não atualização de dependências mesmo com a notificação.

Recentemente, Rebatchi et al. (2024) realizaram um estudo empírico em larga escala sobre o uso do Dependabot para gerenciamento de vulnerabilidades em dependências, analisando mais de 9 milhões de *pull requests* em repositórios do GitHub. O trabalho revelou que o Dependabot é responsável por mais de 65% das atualizações automatizadas, com alta taxa de aceitação e aplicação rápida de correções. No entanto,

o estudo também identificou um atraso médio de 512 dias entre a identificação da vulnerabilidade e sua correção, evidenciando limitações na detecção e mitigação precoce de falhas. Esses resultados destacam a relevância dos *bots* no suporte à manutenção de dependências, mas também apontam a necessidade de estratégias complementares para reduzir o tempo de exposição a riscos. Entretanto, o conjunto de dados não se restringe apenas a *pull requests* gerados pelo Dependabot. Os autores analisaram *issues* associadas a *pull requests* voltados à gestão de dependências, criados por pessoas e por *bots*.

6. Ameaças à Validade

O primeiro ponto de ameaça se relaciona à base de dados construída, que se limitou aos 500 repositórios mais populares do GitHub. Portanto, os resultados não podem ser generalizados para outros ecossistemas de desenvolvimento. Adicionalmente, considerou-se no máximo 10 mil *pull requests* por projeto, devido às limitações da API. A escolha pelos 500 repositórios mais populares também traz a limitação de não considerar o funcionamento de repositórios menores ou ainda aspectos relacionados a tipos diferentes de plataformas. Fatores como a política de manutenção de repositórios e a percepção dos desenvolvedores sobre a segurança de código também podem influenciar a adoção e a aceitação de *pull requests* gerados pelo Dependabot. Entretanto, realizou-se um estudo em larga escala, com um conjunto de dados que inclui aproximadamente 1 milhão de *pull requests* e linguagens de programação distintas.

Como segunda ameaça à validade, pode-se citar os filtros aplicados para a identificação de *pull requests* de autoria do Dependabot, bem como a identificação do *status* das contribuições. Para mitigar possíveis erros, elaborou-se um conjunto de *scripts* que se baseia na API oficial do GitHub. Uma amostra dos dados foi inspecionada e os *scripts* foram revisados por mais de um autor.

7. Conclusão

Neste trabalho, realizou-se uma análise em larga escala sobre o uso do Dependabot no GitHub, investigando como os mantenedores lidam com atualizações automatizadas de dependências. Foram minerados aproximadamente um milhão de *pull requests* distribuídos em 500 repositórios, a fim de responder quatro questões de pesquisa relacionadas à adoção da ferramenta. Dentre os principais resultados, destaca-se que:

- cerca de 70% dos projetos analisados utilizaram o Dependabot ao menos uma vez ao longo do período estudado;
- o número de *pull requests* gerados pela ferramenta por projeto variou significativamente, com uma mediana de 43;
- a taxa de *pull requests* aceitos varia entre 22% e 100%, com uma mediana de 58,43%;
- o tempo até o *merge* de uma atualização variou entre poucos dias e vários meses.

Esses resultados sugerem que, embora o Dependabot seja uma ferramenta promissora para o gerenciamento automatizado de dependências, sua adoção efetiva depende de fatores como o tipo de atualização e as decisões dos mantenedores. Como trabalhos futuros, pretende-se identificar as *breaking changes* introduzidas por contribuições do Dependabot, isto é, atualizações em bibliotecas e *frameworks* que não mantêm a retrocompatibilidade. Adicionalmente, consideram-se análises mais aprofundadas para além

da categorização realizando também métodos qualitativos para compreender os desafios na atualização de dependências e desenvolver ferramentas mais adequadas ao contexto de uso.

Disponibilidade de Artefato. Os *scripts* e *dataset* encontram-se publicamente disponíveis em `github.com/alinebrito/vem2025-replication-package-dependabot`

Referências

Alfadel, M., Costa, D. E., Shihab, E., and Mkhallalati, M. (2021). On the use of dependabot security pull requests. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 254–265. IEEE.

Birari, S. (2024). Github dependabot: An essential tool for efficient dependency version management. Medium. Accessed: 19 November 2024.

Bogart, C., Kästner, C., Herbsleb, J., and Thung, F. (2016). How to break an API: cost negotiation and community values in three software ecosystems. In *24th International Symposium on the Foundations of Software Engineering (FSE)*, pages 109–120.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In *32nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344.

Brito, A., Xavier, L., Hora, A., and Valente, M. T. (2018). Why and how Java developers break APIs. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1–11.

Chowdhury, B., Rabbi, M. F., Hasan, S. M., and Zibran, M. F. (2025a). Insights into dependency maintenance trends in the maven ecosystem. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pages 280–284. IEEE.

Chowdhury, B., Rabbi, M. F., Mahedy Hasan, S. M., and Zibran, M. F. (2025b). Insights into dependency maintenance trends in the maven ecosystem. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pages 280–284.

Decan, A., Mens, T., and Constantinou, E. (2018). On the impact of security vulnerabilities in the npm package dependency network. In *15th International Conference on Mining Software Repositories (MSR)*, page 181–191.

Durumeric, Z., Li, F., Kasten, J., Amann, J., Beekman, J., Payer, M., Weaver, N., Adrian, D., Paxson, V., Bailey, M., and Halderman, J. A. (2014). The matter of heartbleed. In *2014 Conference on Internet Measurement Conference (IMC)*, page 475–488.

Garrett, K., Ferreira, G., Jia, L., Sunshine, J., and Kästner, C. (2019). Detecting suspicious package updates. In *41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 13–16.

He, R., He, H., Zhang, Y., and Zhou, M. (2023). Automating dependency updates in practice: An exploratory study on github dependabot. *IEEE Transactions on Software Engineering*, 49(8):4004–4022.

Hora, A. and Valente, M. T. (2015). apiwave: Keeping track of api popularity and migration. In *31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 321–323.

Jafari, A. J., Costa, D. E., Abdalkareem, R., Shihab, E., and Tsantalis, N. (2022). Dependency smells in javascript projects. *IEEE Transactions on Software Engineering*, 48(10):3790–3807.

Kula, R., German, D., Ouni, A., et al. (2018). Do developers update their library dependencies? *Empirical Software Engineering*, 23:384–417.

McDonnell, T., Ray, B., and Kim, M. (2013). An empirical study of API stability and adoption in the Android ecosystem. In *29th International Conference on Software Maintenance (ICSM)*, pages 70–79.

Mens, T. and Decan, A. (2024). An overview and catalogue of dependency challenges in open source software package registries. In *Proceedings of the 23rd Belgium–Netherlands Software Evolution Workshop (BENEVOL24)*, CEUR Workshop Proceedings, Namur, Belgium.

Mirhosseini, S. and Parnin, C. (2017). Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 84–94. IEEE.

Mohayeji, H., Agaronian, A., Constantinou, E., Zannone, N., and Serebrenik, A. (2023). Investigating the resolution of vulnerable dependencies with dependabot security updates. In *20th International Conference on Mining Software Repositories (MSR)*, pages 234–246.

Prana, G. A. A., Sharma, A., Shar, L. K., Foo, D., Santosa, A. E., Sharma, A., and Lo, D. (2021). Out of sight, out of mind? how vulnerable dependencies affect open-source projects. *Empirical Software Engineering*, 26:1–34.

Rebatchi, H., Bissyandé, T. F., and Moha, N. (2024). Dependabot and security pull requests: large empirical study. *Empirical Software Engineering*, 29(5):128.

Robbes, R., Lungu, M., and Röthlisberger, D. (2012). How do developers react to API deprecation? the case of a Smalltalk ecosystem. In *20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 56:1–56:11.

Silva, H. and Valente, M. T. (2018). What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129.

Wu, W., Gueheneuc, Y.-G., Antoniol, G., and Kim, M. (2010). AURA: a hybrid approach to identify framework evolution. In *32nd International Conference on Software Engineering (ICSE)*, pages 325–334.

Xavier, L., Brito, A., Hora, A., and Valente, M. T. (2017). Historical and impact analysis of API breaking changes: A large scale study. In *24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 138–147.