Análise empírica sobre retrabalho manual no gerenciamento de dependências com Dependabot

1st Lourenço F. M. D. Montenegro
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brasil
lourencomonteiro@ufmg.br

2nd André Cavalcante Hora
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brasil
andrehora@dcc.ufmg.br

***Abstract*—Modern software systems rely heavily on third-party libraries, making automated dependency management tools increasingly important. Dependabot is widely adopted to create pull requests (PRs) that update vulnerable or outdated dependencies; however, many of these PRs are closed without merge, and it is unclear to what extent their changes are later reproduced manually by developers. This paper reports an empirical study on manual rework following rejected Dependabot updates. We mined Java repositories hosted on GitHub, selected by activity and maturity thresholds, and collected Dependabot pull requests closed without merge. For each pull request, was extracted type of update (major, minor, or patch) and number of dependencies updated, and use PyDriller to analyze commits to the main branch within time windows of 30, 60, and 90 days after PR closure. A commit that updates the same dependency is classified as manual rework. Our results show that 10.05% of closed Dependabot PRs are followed by manual rework on the same dependencies, with an average delay of about 15 days. Patch updates exhibit a higher rework rate than minor or major updates, and PRs touching more dependencies tend to suffer more rework. In contrast, repository size/age show no clear correlation with mnual rework, while repositories with 4 until 10 contributors present low rework rate. These findings provide evidence on when automated dependency updates are effectively leveraged, when they are overridden by developers, and which project profiles are more prone to manual rework.**

***Index Terms*—dependencies, dependabot, manual rework, pull request**

I. Introdução

Sistemas de software atuais tem a tendência de a depender gradativamente mais de bibliotecas e frameworks. Tais dependências fornecem funcionalidades essenciais e auxiliam as aplicações a evoluir em um ritmo mais acelerado[1]. Esse fato, naturalmente, exige que um gerenciamento eficientes desses componentes terceiros seja realizado, de forma que estejam sempre atualizados e protegidos de brechas de segurança. A literatura sobre o gerenciamento de dependências destaca o desenvolvimento de tecnologias e ferramentas para automatização de tarefas de atualização de dependências. Soluções de gerenciamento automatizadas tem sido gradativamente mais adotadas, para facilitar o trabalho dos desenvolvedores no ciclo de desenvolvimento. Uma dessas ferramentas é o Dependabot, que cria pull requests automatizados para atualizar dependências desatualizadas ou com vulnerabilidades de segurança.

Estudos relacionados ao gerencimento automatizado de dependências utilizando ferramentas como o Dependabot exploram o comportamento dos desenvolvedores ao interagir com as pull requests geradas pelo bot. Mais especificamente, a literatura analisa a forma como os desenvolvedores lidam quando o bot abre pull requests de atualizações de segurança, quando as propostas de atualização são aceitas, ignoradas ou rejeitadas. Essa análise tem como principal objetivo entender qual é o impacto do uso do Dependabot na atualização das dependências. Esses estudos mostram o crescente papel da ferramenta Dependabot no processo de manutenção e gerenciamento de dependências, reforçando a relevância de compreender, de forma empírica, como as atualizações automáticas são tratadas na prática pelos desenvolvedores.

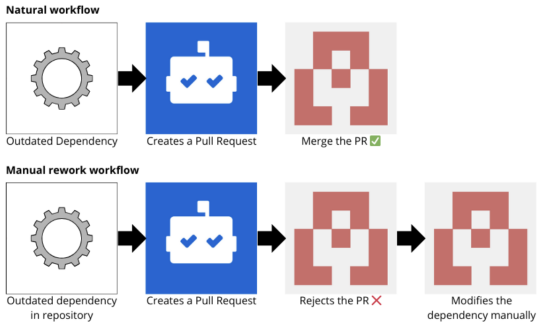
Apesar disso, ainda existe uma lacuna de investigação sobre se as dependências sugeridas pelo Dependabot para atualização são posteriormente atualizadas por desenvolvedores por meio de commits manuais após o encerramento das pull requests sem merge. O retrabalho manual foi considerado quando um pull request aberto pelo Dependabot é fechado sem merge e posteriormente a mesma dependência é atualizada por um desenvolvedor manualmente. O retrabalho manual pode ser interpretado como ineficiência no processo desses mecanismos de automação, uma vez que o procedimento acaba exigindo repetição manual. Nesta pesquisa, foram conduzidos experimentos iniciais envolvendo múltiplas linguagens de programação, nos quais foram encontradas dificuldades, como diferentes formas de tratar e identificar dependências para declaração. Por isso, esta pesquisa foca em projetos Java hospedados no GitHub.

O retrabalho manual traz impacto significativo porque funciona como um indicador de um desperdício de tempo de desenvolvimento e de esforço. O Dependabot identifica uma dependência desatualizada, abre um pull request para atualizar a dependência, porém um desenvolvedor precisa refazer a mesma atualização manualmente. Esse retrabalho resulta em desperdício de tempo que poderia ser direcionado para outras tarefas do ciclo de desenvolvimento. Ademais, um alto fluxo de atualizações automáticas não aproveitadas pode atrapalhar o fluxo de trabalho e sobrecarregar desenvolvedores, dificultando a identificação das mudanças que realmente precisam ser feitas.

No contexto de avaliação de ferramentas, estudos anteriores costumam usar métricas como taxa de merge e tempo até o merge, além de razões para aceitação ou rejeição de PRs, para medir a efetividade do Dependabot[1][3]. Porém, quando uma PR é fechada e a mesma dependência é atualizada manualmente mais tarde, o status da pull request fechada não captura o papel desempenhado pelo bot nesse cenário de forma adequada.

Isso gera uma lacuna na literatura que impede a compreensão de quando a automação de atualizações realmente agrega valor ao processo de manutenção de software e de quando leva a esforço duplicado, levando a desperdício de tempo dos desenvolvedores. Quantificando empiricamente com que frequência ocorre retrabalho manual após pull requests do Dependabot fechadas sem merge e relacionar esse fenômeno a características das PRs e dos repositórios, este trabalho busca fornecer evidências que ajudem pesquisadores e profissionais a ajustar, adotar e evoluir ferramentas de atualização automática de dependências.

Diante do cenário descrito, este trabalho investiga os PRs do Dependabot fechadas sem merge. Inicialmente, foi preciso compreender em que medida esses pull requests resultam em retrabalho manual e como esse retrabalho se relaciona com características das pull requests e dos repositórios. O principal objetivo é responder as seguintes questões: em que medida pull requests do Dependabot rejeitadas pelos desenvolvedores resultam em retrabalho manual, ou seja, em commits por desenvolvedores que atualizam as mesmas dependências posteriormente? E quais fatores de projeto ou de PR, como tamanho e idade do repositório, número de contribuidores, tipo de atualização e número de dependências modificadas, estão associados à ocorrência de retrabalho manual?



II. Referencial Teórico

A. Gerenciamento de dependências e vulnerabilidades em software

A utilização de dependências e bibliotecas traz importantes ganhos quando se trata de produtividade e qualidade do software. O principal deles é permitir que os desenvolvedores possam trabalhar mais tempo em regras de negócio e delegar tarefas manuais a bots ou tarefas automáticas [3]. Contudo, essa dependência de ferramentas externas pode expor projetos a uma superfície de ataque e a um impacto potencial de vulnerabilidades [3]. Dessa maneira, manter as dependências atualizadas se torna uma atividade de grande relevância no processo de desenvolvimento de software.

Ao mesmo tempo, o crescente aumento de escala das aplicações tornaram inviável o gerenciamento totalmente manual de dependências. Os estudos em referência mostram que desenvolvedores têm dificuldade em acompanhar vulnerabilidades recém-divulgadas e aplicar atualizações em tempo hábil. Esse fato corrobora com o crescimento do débito técnico: nome dado ao atraso entre a versão de uma dependência utilizada em um projeto e a versão mais recente disponível no ecossistema [1]. Esse atraso aumenta o risco de exposição a vulnerabilidades conhecidas e dificulta a manutenção contínua da base de código.

B. Ferramentas automatizadas e bots de atualização de dependências

Para resolver esse problema, desenvolvedores têm investido tempo para configuração e desenvolvimento de ferramentas que auxiliam na automação do gerenciamento de dependências. Essas ferramentas são construídas para monitorar repositórios, compararando as versões de dependências utilizadas com as versões mais novas disponíveis. Quando as ferramentas encontram uma atualização relevante, geram pull requests para propor a alteração, evitando brechas de segurança e dependências desatualizadas [2][3].

O Dependabot é uma das ferramentas de gerenciamento de dependências mais adotados no GitHub. O Dependabot monitora arquivos de configuração de dependências e a base GitHub Security Advisories, identificando dependências desatualizadas ou vulneráveis e abrindo PRs para atualizá-las [1][3]. A utilização dessa ferramenta permite que o esforço dos desenvolvedores esteja focado em revisar, testar e decidir sobre o merge dessas PRs. Em teoria, esse fluxo deveria reduzir o débito técnico e acelerar a aplicação de correções de segurança[1][2].

C. Dependabot e atualizações de dependências em projetos GitHub

Três estudos em específico serviram como base e inspiração para realização deste trabalho. Dentre eles, ”Automating dependency updates in practice: An exploratory study on github dependabot”[1] analisam projetos JavaScript ativos no GitHub com o Dependabot habilitado. Dentre os repositórios coletados, pull requests de segurança criados entre 2017 e 2020 foram coletados e o estudo buscou medir: 1- Qual a taxa e o tempo de aceitação dos pull-requests do Dependabot. 2- Quais são os motivos pelos quais pull requests não são aceitos? 3- Quais fatores influenciam o tempo de merge de um pull request?

O segundo estudo, ”Dependabot and security pull requests: large empirical study. Empirical Software Engineering”[2], avalia como a adoção do Dependabot evoului ao longo do tempo e quais fatores sustentam ou dificultam sua utilização contínua.

Por fim, o terceiro estudo, ”On the use of dependabot security pull requests”[3], avalia se ferramentas como o De-

pendabot reduzem vulnerabilidades e melhoram a qualidade do software ou apenas aumentam a carga de manutenção.

III. Contribuição

A. Coleta de repositórios

Primeiramente, para realização do estudo foi necessário definir de quais repositórios as informações de commits e pull requests seriam coletadas. Quais lingugagens ou frameworks seriam considerados? Essa é uma decisão importante pois diferentes linguagens, repositórios ou frameworks podem trazer diferentes insights sobre as perguntas que queremos responder na pesquisa. Inicialmente a premissa era coletar repositórios em Java, JavaScript e Python. Entretanto nas primeiras etapas da execução do projeto foi visto que identificar o retrabalho em repositórios que utilizavam tecnologias diferentes poderia trazer ruídos e maior dificuldade de identificação de retrabalho. Portanto, para validar uma hipótese inicial, a linguagem Java foi escolhida pela forma em que os repositórios que utilizam a linguagem como linguagem principal gerenciam dependências ter um padrão mais consolidado para identificar as dependências que foram atualizadas e, consequentemente, identificar o retrabalho.

Após definir claramente o escopo do estudo, foi preciso coletar os repositórios para começar a parte de mineração dos dados. Dessa forma, a ferramenta GHS[4] foi utilizada para seleção dos repositórios no GitHub. Essa ferramenta permite selecionar diversos filtros diferentes e retorna repositórios no GitHub que atendem aos filtros selecionados, como número de estrelas, número de commits, contribuidores, branches, etc. Portanto, outra decisão que seria importante para prosseguimento do estudo era a escolha desses filtros. O objetivo era identificar repositórios que representassem cenários reais de desenvolvimento de software, no qual problemas enfrentados por médias e grandes aplicações pudessem ser identificados, para entender o cenário de pull requests fechados com retrabalho manual e entender os verdadeiros motivos para que esse fenômeno acontecesse. Dessa maneira, foram selecionados os filtros:

- Linguagem: Java
- Número mínimo de commits: 100
- Número mínimo de pull requests: 100
- Número mínimo de estrelas: 100
- Criado no máximo até 01/01/2024
- Não incluir forks de outros repositórios

O número mínimo de commits, estrelas e pull requests foi escolhido afim de capturar repositórios que tivessem o tamanho suficiente para que o fenômeno fosse encontrado e aplicações reais fossem identificadas. Além disso, a data de criação máxima foi escolhida para que os repositórios escolhidos não fossem recém-criados e refletissem processo de engenharia de software natural. Por fim, não foram incluídos forks de outros repositórios para que uma diversidade maior de código fosse abrangida.

Após a coleta dos repositórios, foi necessário realizar uma segunda filtragem: identificar quais projetos utilizavam o Dependabot. Para isso um script em Python foi criado para clonar os repositórios e verificar a existência do arquivo de configuração do Dependabot (dentro da pasta .github o arquivo dependabot.yml ou dependabot.yaml). Dos 4.806 repositórios coletados na primeira etapa (utilizando a ferramenta GHS), em 1.244 repositórios foram encontrados o arquivo de configuração do Dependabot. Ou seja, aproximadamente 25,8% dos repositórios Java coletados utilizavam o Dependabot.

B. Coleta de Pull Requests

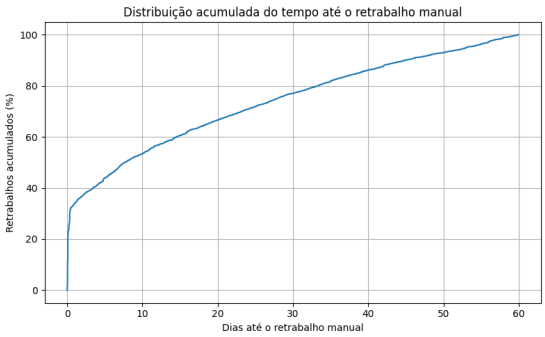
Com os repositórios que utilizavam o Dependabot coletados, a próxima etapa foi coletar as pull requests fechadas sem merge de cada um dos repositórios. Para isso foi utilizada a API do GitHub, na qual, um token foi gerado para utilização da API. Um arquivo JSON foi utilizado para armazenar a URL de cada um dos repositórios. Todo o desenvolvimento do estudo foi feito utilizando notebooks Python. Um loop foi utilizado para iterar pelos repositórios, no qual em cada iteração era feita uma requisição a API do GitHub coletando os seguintes dados de cada um dos pull requests fechados: repository, pr_number, state, closed_at, merged_at, title, html_url, author, head_sha, base_ref, created_at, updated_at, labels, dependency_updates (lista de dependency, from_version, to_version). Para identificação de que o pull request era autorado pelo Dependabot o atributo ”author” foi verificado para que fosse igual a ”dependabot[bot]”.

Após a coleta e armazenamento dos pull requests a estrutura para analisar o retrabalho manual estava praticamente pronta.

C. Definição de retrabalho manual

Para coletar resultados e interpretar os pontos dessa pesquisa seria necessário ter uma definição de como seria categorizado o retrabalho manual. O que queremos identificar é se o Dependabot fez uma sugestão de mudança de dependência, abrindo um pull request, essa sugestão foi rejeitada, ou seja, o pull request foi fechado e não mergeado, e, por fim, em uma janela de tempo foi feita alguma alteração manual na dependência. Foi considerada a ”mesma dependência” quando o commit modifica a versão do mesmo artefato identificado em arquivos de configuração.

Para identificação do retrabalho foram testadas diferentes quantidades de dias: 30, 60 e 90 dias. A diferença entre a janela de 60 e 90 dias era muito baixa, escolhendo uma janela de 90 dias pouco retrabalho foi identificado a mais e a hipótese de que mais que 60 dias para alteração da dependência poderia não configurar retrabalho manual. Por outro lado, a diferença do retrabalho entre 30 e 60 dias foi grande, como mostra o gráfico a seguir:



Mais de 20% do retrabalho medido foi identificado após o trigésimo dia após o fechamento do pull request. Portanto a janela de tempo mais adequada escolhida foi de 60 dias. Dessa forma, concluiu-se que intervalos superiores a 60 dias podiam refletir decisões de atualização em um contexto diferente do pull request do Dependabot, tornando-o menos plausível de serem interpretados como retrabalho direto.

D. Medição do retrabalho manual

Antes de identificação de retrabalho manual, foi feito um enriquecimento dos dados coletados nos pull requests. Foram identificados se as mudanças feitas no pull request eram major, minor ou patch. As mudanças que não entravam em nenhuma dessas 3 categorias eram marcadas como ”unknown”. Como vários pull requests modificavam mais de uma dependência, a alteração mais relevante foi levada em consideração, por exemplo, se havia uma mudança patch e minor, o pull request era classificado como minor. Essa classificação foi feita observando a descrição do pull request do Dependabot ou, caso essa primeira opção não funcionasse, verificando a diferença no código e ver a versão original da dependência e a versão atualizada.

Para identificação do retrabalho, foi utilizada a biblioteca PyDriller. Essa biblioteca é utilizada para mineração de dados em repositórios para extrair informações de repositórios que utilizam Git, como commits, mudanças e código fonte. Dessa forma, para cada pull request fechado, uma busca era feita para verificar se, em uma janela de 60 dias após o fechamento do pull request, havia alguma alteração de um autor que não era o Dependabot na mesma dependência listada no pull request.

Dessa maneira, de 60.572 pull requests fechados analisados, em 6.086 foi identificado retrabalho manual em até 60 dias. O tempo médio para que a mudança do retrabalho fosse feito foi de 15.68 dias. Após a coleta dessa métrica, um estudo de identificação de padrões e correlações foi iniciado para entender quais eram os possíveis motivos que influenciavam no aumento ou diminuição do fenômeno de retrabalho manual.

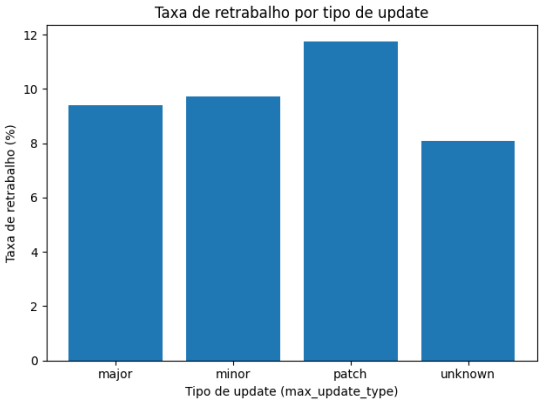
O retrabalho foi medido em nível de PR: para cada PR fechada, verificamos apenas se existia ao menos um commit de retrabalho no intervalo de 60 dias (independente do número de commits).

E. Análise exploratória de retrabalho manual

As análises foram conduzidas de forma descritiva, agrupando PRs por faixas de valores (por exemplo, número de dependências, número de contribuidores) e calculando a taxa de retrabalho em cada grupo.

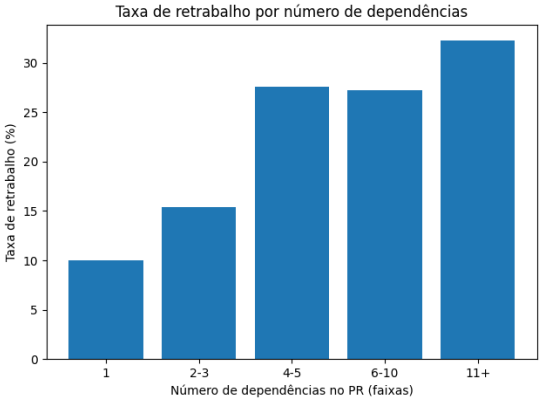
Com o retrabalho coletado e armazenado, foram traçadas análises exploratórias para identificar fatores que influenciam no aumento ou diminuição do retrabalho. Dentre os fatores escolhidos para análise foram escolhidos escopos de característica de: repositório, pull request e dependência.

Inicialmente, o tipo de update foi comparado com a taxa de retrabalho correspondente e foi registrado o seguinte resultado:



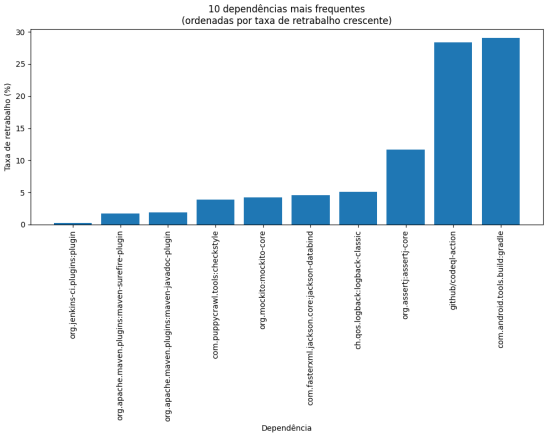
As mudanças do tipo ”patch” registraram a maior taxa de retrabalho, um pouco menos de 12%, enquanto as mudanças ”major”, ”minor” e ”unknown” tiveram uma taxa de retrabalho abaixo de 10%.

Por outro lado, relacionando a taxa de retrabalho ao número de dependências que o pull request atualiza, houve um resultado interessante. Quanto maior o número de dependências maior foi o retrabalho, de acordo com o agrupamento feito no gráfico abaixo:

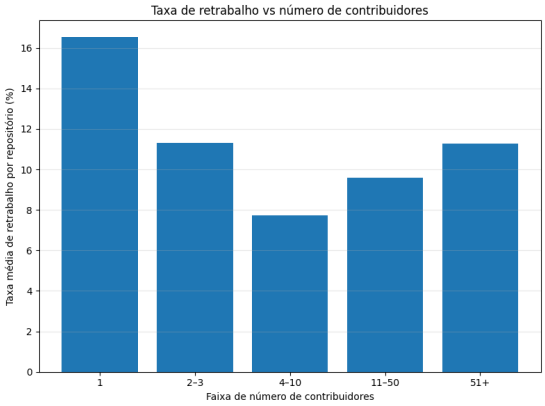


Os pull requests que atualizavam 11 ou mais dependências tiveram uma taxa de retrabalho de mais de 30%.

Também foram levantadas as 10 dependências mais frequentes para tentar encontrar algum padrão de tipo de dependência que gerava maior taxa de retrabalho:

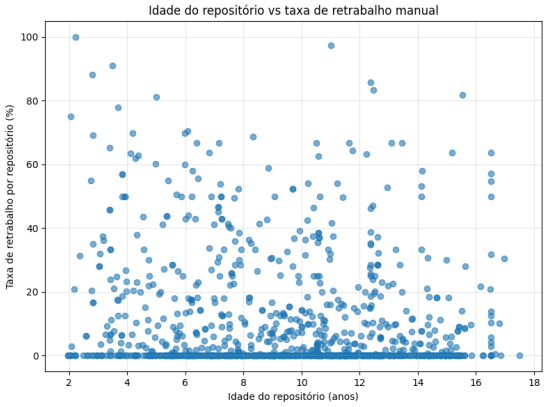


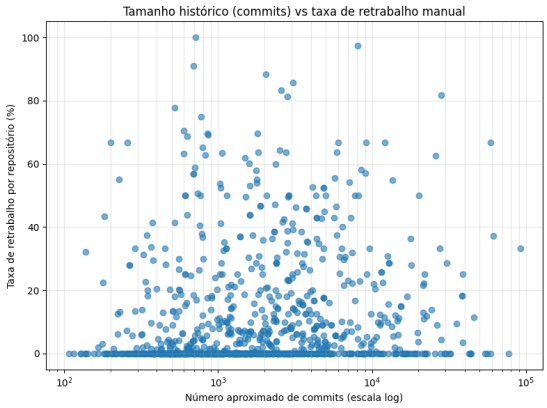
Passando para métricas relacionadas ao repositório, foi medido a média de retrabalho por repositório e medida a taxa de retrabalho por número de contribuidores de um projeto:



Foi perceptível que repositórios com apenas 1 contribuidor mostrou uma taxa de retrabalho bem maior se comparada com múltiplos contribuidores, na qual a faixa de 4-10 contribuidores apresentou a menor taxa dentre os grupos avaliados.

Por outro lado, foi também medido a taxa de retrabalho comparada com a idade do repositório e o tamanho em número de commits do repositório. Ambas medições não mostraram resultados conclusivos, mostrando que essas características, a princípio, não influenciam na taxa de retrabalho de pull requests fechados do Dependabot.





IV. Conclusões e Trabalhos Futuros

A. Conclusões

Diante do cenário de crescente adoção de bots de atualização de dependências, este trabalho teve como objetivo investigar em que medida pull requests (PRs) do Dependabot fechadas sem *merge* resultam em retrabalho manual e como esse retrabalho se relaciona com características das PRs e dos repositórios em projetos Java hospedados no GitHub. Para isso, foi definido um recorte empírico que considera apenas projetos cuja linguagem principal é Java e que apresentam um nível mínimo de maturidade.

Para responder à primeira questão de pesquisa, foi necessário operacionalizar o conceito de retrabalho manual. Neste estudo, o retrabalho manual foi definido como: o caso em que uma pull request do Dependabot é fechada sem merge e ocorre pelo menos um commit realizado por um desenvolvedor humano que atualiza a mesma dependência em até 60 dias após o fechamento. Dessa forma, foram identificadas 6.086 PRs com retrabalho manual dentre as 60.572 PRs fechadas analisadas, o que corresponde a aproximadamente 10,05% dos casos. O tempo médio entre o fechamento da PR e o commit de retrabalho foi de 15,68 dias. Esses resultados indicam que uma fração não desprezível das PRs do Dependabot fechadas sem merge acaba tendo suas atualizações reimplementadas manualmente, o que sugere que o status ”closed” não implica, necessariamente, rejeição definitiva da atualização proposta.

Comparando características das PRs, os resultados mostraram que o tipo de atualização e o número de dependências modificadas estão associados a diferentes taxas de retrabalho. As PRs foram classificadas em *major*, *minor*, *patch* ou *unknown*. As atualizações *patch* foram as que tiveram a maior taxa de retrabalho registrada, enquanto pull requests *major*, *minor* ou *unknown* registraram menor taxa de retrabalho manual. Uma possível interpretação para esse resultado é que as atualizações *patch* por serem, em tese, menos arriscadas, podem ser mais facilmente reproduzidas manualmente em um momento mais adequado. Ademais, foi observado que pull requests que atualizam um maior número de dependências tem a tendência a registrar taxas de retrabalho significativamente mais altas: pull requests que atualizam 11 ou mais dependências atingiram taxas maiores que 30%. Por outro lado, a análise das 10 dependências

mais frequentes não revelou padrão claro de tipo específico de dependência associado a maiores taxas de retrabalho, sugerindo que o fenômeno está mais ligado à "forma" da PR (tamanho, tipo de update) do que à natureza individual de uma biblioteca específica.

Por outro lado, nas características de repositório, foram monitoradas métricas como número de contribuidores, idade do repositório e tamanho medido em número de commits. A taxa média de retrabalho por repositório relacionado ao número de contribuidores mostrou que os projetos com apenas um único contribuidor apresentam taxa de retrabalho significativamente maior. Enquanto isso, projetos com dois ou mais contribuidores apresentaram taxa menor de retrabalho. Mais especificamente, a faixa de 4 a 10 contribuidores registrou a menor taxa de retrabalho, fato que pode indicar que equipes desse porte possuem capacidade superior de triagem e revisão mais estruturada para lidar com PRs automatizadas. Por fim, análises que comparam taxa de retrabalho à idade do repositório e ao número total de commits não mostraram padrões confiáveis, sugerindo que características de tempo e idade do repositório não produzem influência clara sobre a probabilidade de uma pull request do Dependabot fechar sem merge e dar origem a retrabalho manual.

Em resumo, os resultados obtidos mostram que (i) em projetos Java cerca de 10% das PRs do Dependabot fechadas sem merger recebem atualizações manuais em até 60 dias por um desenvolvedor, com um intervalo médio de pouco mais de duas semanas; (ii) pull requests com atualizações *patch* e pull requests que afetam um alto número de dependências estão associadas a taxas de retrabalho mais elevadas; e, por fim, (iii) o número de contribuidores por repositório parece possuir maior relevância do que idade ou tamanho do repositório para identificação de retrabalho manual. As conclusões detalhadas complementam a literatura sobre o Dependabot ao irem além da análise tradicional de taxas de merge e tempo de resposta, introduzindo o conceito de perspectiva de retrabalho manual como uma dimensão adicional para avaliar a eficácia de bots de atualização de dependências e seus impactos no fluxo de manutenção.

B. Trabalhos futuros

Embora os resultados apresentados forneçam evidências sobre padrões de retrabalho manual em pull requests do Dependabot, existem várias oportunidades para extensão e maior aprofundamento no tema. Em primeiro lugar, a escolha da janela de 60 dias para retrabalho, embora empiricamente justificada neste estudo, pode ser revista em abordagens que considerem janelas dinâmicas ou adaptativas ou específicas por tipo de dependência ou perfil de projeto.

Uma segunda direção promissora é a ampliação do escopo empírico. Este trabalho focou exclusivamente em projetos Java, justamente por conta da maior padronização no gerenciamento de dependências, mas há interesse evidente em retomar a intenção inicial de incluir outras linguagens e ecossistemas, como JavaScript e Python. Outros ambientes trazem desafios próprios na identificação de dependências e na evolução de versões, de modo que estudos comparativos podem revelar se os padrões observados em Java se mantêm ou se são específicos desse ecossistema. Igualmente, trabalhos futuros podem fazer o comparativo do comportamento de retrabalho manual em pull requests geradas por diferentes bots de atualização com o desempenho do Dependabot.

Por fim, um recorte relevante de pesquisa envolve investigar de forma mais aprofundada o destino das PRs rejeitadas que não geram retrabalho manual. É importante entender em que medida PRs fechadas sem retrabalho são posteriormente substituídas por novas PRs do Dependabot que acabam sendo aceitas, em que casos a dependência é removida do projeto e em quantas situações a dependência permanece desatualizada por longos períodos. Abordagens qualitativas, como estudos de caso, análise de comentários em PRs e entrevistas ou questionários com mantenedores, podem complementar as análises quantitativas ao esclarecer os motivos pelos quais equipes optam por fechar PRs automatizadas e, ainda assim, por vezes refazer manualmente a atualização proposta. Avançar nessa direção permitirá não apenas caracterizar "se" o retrabalho manual ocorre, mas também compreender "por que" ele ocorre e como ferramentas como o Dependabot podem ser ajustadas para reduzir esforço duplicado e aumentar a confiança dos desenvolvedores na automação de updates.

REFERENCES

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

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

[3] Alfadel, M., Costa, D. E., Shihab, E., Mkhallalati, M. (2021, May). On the use of dependabot security pull requests. In 2021 IEEE/ACM 18th International conference on mining software repositories (MSR) (pp. 254-265). IEEE.

[4] Dabic, O., Aghajani, E., & Bavota, G. (2021, May). Sampling projects in github for MSR studies. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (pp. 560-564). IEEE.

[5] Mohayeji, H., Agaronian, A., Constantinou, E., Zannone, N., & Serebrenik, A. (2023, May). Investigating the resolution of vulnerable dependencies with dependabot security updates. In 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR) (pp. 234-246). IEEE.

[6] Estevão, G., Lott, L., Fleury, L., Nalon, Y., Brito, A., Souza, J. G., & Xavier, L. (2025, September). Gerenciamento Automatizado de Dependências: Um Estudo em Larga Escala sobre a Adoção do Dependabot. In Workshop de Visualização, Evolução e Manutenção de Software (VEM) (pp. 37-47). SBC.