Aprimorando a obtenção de dados relevantes sobre as práticas de Integração e Entrega Contínua em repositórios de software

Palavras-Chave: Continuous Delivery, Continuous Integration, Mining Software Repositories

Autores:

Geovani Reolon de Sousa – IMECC Unicamp

Prof. Breno Bernard Nicolau de Franca – IC Unicamp

Introdução:

Atualmente, as práticas de integração (CI) e entrega contínua (CD) vêm sendo crescentemente adotadas por organizações com foco no desenvolvimento de software devido a intensa velocidade e flexibilidade para atender as demandas que o mercado exige. Essas práticas são baseadas no princípio de automação das etapas do desenvolvimento de software, no qual desenvolvedores constroem, executam, testam e entregam software de uma maneira automatizada e consistente [1]. Ainda mais, as práticas de CI/CD são utilizadas de acordo com um pipeline, que é um fluxo organizado de etapas para entrega de software no menor tempo possível.

Nesse cenário, a recorrente adoção dos métodos CI/CD gera uma necessidade sobre os desenvolvedores de possuírem um panorama geral sobre os impactos dessas práticas e seu nível de adoção em projetos para que compreendam como essas são utilizadas e suas limitações. Consequentemente, esse conhecimento permite a melhoria da capacidade de desenvolvimento e evolução dos sistemas de software. Desse modo, técnicas de mineração e análise de repositórios de software vêm sendo desenvolvidas para analisar o nível de adoção de práticas CI/CD e como elas impactam na garantia de qualidade e redução do tempo de entrega.

Tendo isso em vista, a plataforma GitHub, uma das mais populares em serviços de gerenciamento e controle de versão, é uma fonte rica de informações para análise de repositórios de software. Pois, ela apresenta uma API (*Application Programming Interface*) REST (REpresentational State Transfer), que disponibiliza diversos dados relevantes sobre as ferramentas e procedimentos relacionados às práticas de CI/CD utilizados dentro dos projetos hospedados, permitindo minerar informações relacionadas a essas práticas. No passado, o GitHub era operado mais como um sistema de controle de versão, por isso processos de construção (*build*) e implantação (*deployment*) acabavam sendo integrados em plataformas externas. Entretanto, no final de 2019, a plataforma lançou um plugin integrado conhecido como *GitHub Actions*, o qual permite administrar fluxos de trabalho (incluindo *pipelines* de CI/CD) e automatizar as etapas de compilação, teste e implantação dentro da plataforma. Desse modo, a chegada do plugin permite que desenvolvedores trabalhem com *pipelines* de CI/CD

dentro do próprio repositório e implementem as práticas de forma simples e integrada no ambiente do GitHub, não precisando mais realizar complexas integrações externas com outros sistemas.

Então, usando a API do *Actions* podemos minerar informações relacionadas à estrutura real do pipeline e suas execuções. Dessa forma, neste projeto apresentamos o CI-Scavenger[1], uma ferramenta destinada à mineração de dados em repositórios de software que caracteriza a utilização de práticas de CI/CD através da coleta de informações referentes à estrutura do pipeline e suas execuções. Desse modo, como uma das principais demandas na área de software analytics é o desenvolvimento de ferramentas de mineração de repositórios de software [2], este projeto pode auxiliar pesquisas que envolvam análise e coleta de dados de repositórios de software livre. Além de caracterizar os efeitos da adoção das práticas citadas no desenvolvimento de sistemas.

Motivação:

Dada a importância das práticas de CI/CD no desenvolvimento de software, a necessidade de ferramentas e técnicas de mineração destinadas a investigar o nível de adoção dessas práticas em repositórios de software se torna um meio relevante e viável para investigar fenômenos associados a essas práticas em larga escala. Assim, iniciativas com esse propósito já existem, como a ferramenta Garimpeiro [2], que permite analisar a adoção de práticas CI/CD em repositórios de software livre hospedados no GitHub. Apesar da capacidade da ferramenta, sua utilização é limitada a apenas repositórios que utilizem plataformas Java e o conjunto de métricas disponibilizadas é limitado no que diz respeito ao escopo das práticas de CI/CD. Desse modo, o Garimpeiro não possibilita identificar a estrutura real do pipeline, apenas inferi-la, e nem analisar os dados das suas execuções.

Assim, o CI-Scavenger pode complementar a ferramenta e produzir análises mais detalhadas, minerando informações que a ferramenta não é capaz de obter. Nessa perspectiva, o desenvolvimento da ferramenta se mostrou relevante para auxiliar possíveis estudos que investiguem o nível de adoção de práticas CI/CD em repositórios de software livre hospedados no GitHub. Como também, possibilitar que desenvolvedores tenham um diagnóstico de como executam essas práticas e permitir a melhoria contínua do desenvolvimento.

Funcionalidades:

A principal aplicação da ferramenta CI-Scavenger é a análise da estrutura dos pipelines definidos no *Actions* e de suas execuções. Para isso, a ferramenta coleta dados por meio de consultas a API do plugin. Além disso, a ferramenta previamente analisa os pipelines presentes no repositório e descarta aqueles que não apresentarem evidências de práticas CI/CD. Essa validação é importante porque algumas organizações implementam fluxos de trabalho sem o propósito de integração e entrega contínua, e apenas automatizam tarefas de manutenção do repositório, como encerrar tarefas antigas ou descartar ramificações abandonadas.

Sobre os pipelines, a ferramenta retorna o nome, estado, data de criação, última atualização e tempo de desenvolvimento do fluxo de trabalho. Assim, possibilitando visualizar quais workflows presentes no repositório estão ativos e sendo utilizados, como também quando foram implementados. Além disso, o tempo de desenvolvimento permite observar com que frequência o pipeline vem sendo atualizado conforme o tempo, visto que calcula a diferença entre a data de criação e última atualização do workflow.

[1] https://github.com/GeovaniR/CI-Scavenger

Para as estatísticas das execuções, a ferramenta calcula a porcentagem de sucesso, a porcentagem de execuções que foram ativados por um conjunto de alterações na ramificação principal (*main branch*) e a porcentagem de execuções que foram ativadas por outras ramificações. Estas duas últimas são importantes, uma vez que Humbley e Farley [1] definem que o fluxo de desenvolvimento não pode se afastar demais da ramificação principal, ao passo que isso caracterizaria uma má prática de integração contínua. Por outro lado, como existem fluxos de trabalho que contém menos execuções do que o informado pelo usuário para análise, também são disponibilizados o número total de execuções que o pipeline apresenta e a quantidade de runs que foram utilizadas para execução da ferramenta. Assim, permitindo visualizar o nível de utilização dos pipelines e quantas execuções foram analisadas para obter as estatísticas calculadas.

A ferramenta ainda obtém dados relacionados à quando as execuções aconteceram e ao seu tempo de execução. Desse modo, são calculados o tempo médio e o desvio padrão do tempo de execução dos fluxos de trabalho, possibilitando analisar o desempenho dos pipelines de CI/CD. Ainda mais, também são calculadas estatísticas sobre o tempo médio e o desvio padrão entre execuções do pipeline. Assim como dados sobre a periodicidade das execuções, apresentando sua frequência entre os meses. Dessa forma, conseguimos analisar se os fluxos de trabalho estão sendo utilizados regularmente, ou só são ativados ocasionalmente.

Por fim, são apresentados dados sobre a média e o desvio padrão da quantidade de *jobs* dos pipelines durante suas execuções. Por definição, os *jobs* são as etapas de automação dentro do fluxo de trabalho, ou seja, o conjunto de práticas que automaticamente serão executadas quando o pipeline for ativado. Desse modo, analisar os jobs permite compreender se os pipelines costumam ser modificados em relação ao conjunto de ações utilizados. Como também, visualizar a quantidade de práticas de automação que os pipelines empregam. Logo, as estatísticas disponibilizadas pela ferramenta relacionadas ao pipeline auxiliam a estudar o nível de adoção das práticas de CI/CD em repositórios de software livre. Apesar disso, é necessário considerar que apenas os dados fornecidos pela API do *GitHub Actions* são recuperados, ou seja, pipelines externos estão fora de escopo.

Métodos:

Em relação ao desenvolvimento, a ferramenta foi construída através da linguagem Python, utilizando principalmente a biblioteca "*requests*" para executar chamadas automatizadas à API do *GitHub Actions*. O processo de desenvolvimento seguiu um ciclo iterativo, onde a cada iteração novas capacidades foram inseridas. Desse modo, a extração dos dados é feita de acordo com a Figura 1, recuperando primeiro informações relacionadas às execuções do pipeline e depois sobre sua estrutura. Inicialmente, são definidos os caminhos que serão utilizados para realizar as chamadas a API através dos parâmetros de entrada. Em seguida, o fluxo de trabalho é analisado para validar se ele é destinado a práticas de CI/CD, e caso confirmado, os dados relacionados às suas execuções e jobs são coletados para o cálculo de estatísticas. Logo após, a ferramenta recupera informações sobre a estrutura do pipeline. Por fim, as informações são organizadas e formatadas em um arquivo de saída para o usuário, juntamente com um arquivo de log das requisições realizadas à API do *GitHub Actions*.

Na parte da validação, o CI-Scavenger vasculha dentro do fluxo de trabalho e em seus jobs, nomes que evidenciem o funcionamento como um pipeline de implementação. Ademais, outros aspectos também são

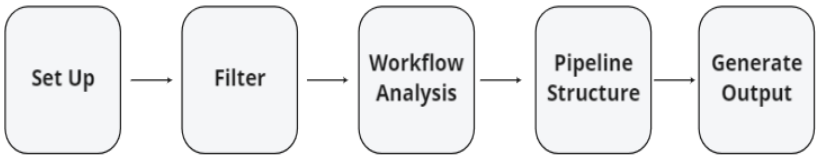

Figura 1. Processo de Execução da Ferramenta

verificados para evitar problemas de funcionamento, como existir pelo menos uma execução aberta. Visto que, as organizações podem manter as execuções dos pipelines fechadas para ao público, e assim não conseguimos obter os dados de suas execuções. Como também, analisamos se o pipeline apresenta execuções recentes (dentro de um ano), pois a API apresenta problemas em disponibilizar dados muito antigos, retornando informações incompletas.

A obtenção de informações relacionadas aos *jobs* é um processo custoso, pois é preciso realizar chamadas individuais às execuções. Assim, o CI-Scavenger foi projetado de modo que o usuário possa escolher entre minerar exatamente o número de execuções solicitadas ou recuperar uma amostra representativa para economizar chamadas à API, escalando a análise.

Resultados:

A seguir, apresentamos um exemplo de aplicação da ferramenta analisando o repositório Django, um framework em Python para desenvolvimento web. Embora esse repositório apresente ao todo seis pipelines, somente cinco foram identificados como pipelines de CI/CD, pois um deles foi descartado por apenas exibir mensagens de boas-vindas a novos contribuidores. A Figura 2 apresenta a estrutura dos pipelines, onde podemos observar que os fluxos de trabalho "*Schedule Tests"* e "*Tests*" adotam uma estratégia interessante disponível no *GitHub Actions*, que é a técnica de matriz. Nessa estratégia, o pipeline é parametrizado para executar os mesmos procedimentos sobre configurações diferentes, o que é útil para testar componentes em múltiplas versões e plataformas diferentes.

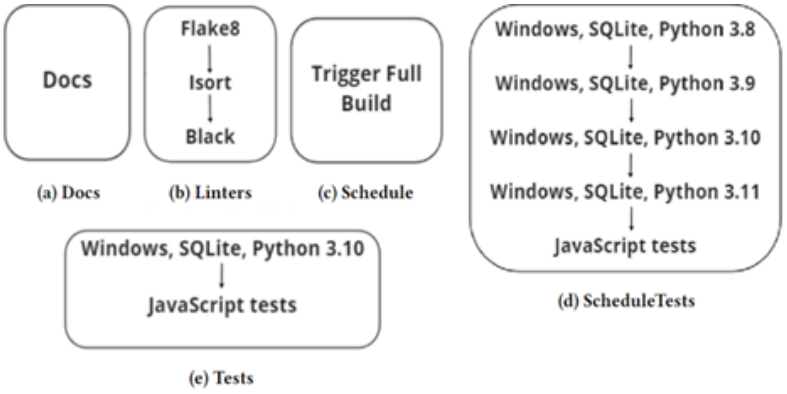

Figura 2. Estrutura dos Pipelines do repositório Django

A Tabela 1 mostra que todos os pipelines estão ativos e que *Tests*, *Docs* e *Linters* são os fluxos de trabalho mais antigos e utilizados de acordo com suas datas de criação e total de execuções. Além disso, pelo tempo de desenvolvimento, observamos que nenhum deles foi atualizado desde sua criação.

Pela frequência de execuções entre os meses, temos que as últimas 100 execuções de todos aconteceram neste ano, principalmente entre os últimos três meses. Por outro lado, o tempo médio entre execuções varia de horas (*Tests*, *Docs* e *Linters*) para um dia (*Schedule Tests* e *Schedule*). Ainda mais, a média e o desvio padrão entre execuções é o mesmo para os pipelines *Tests* e *Linters*, ou seja, os dois são sempre ativados em conjunto.

Django	Tests	Docs	ScheduleTests	Schedule	Linters
State	Active	Active	Active	Active	Active
Created	04/01/2021	25/02/2021	14/03/2022	14/03/2022	20/01/2021
Updated	04/01/2021	25/02/2021	14/03/2022	14/03/2022	20/01/2021
DevTime	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00
Success	82.14%	91.84%	100.0%	100.0%	89.29%
MainBranch	18.0%	25.0%	100.0%	100.0%	18.0%
OtherBranches	82.0%	75.0%	0.0%	0.0%	82.0%
MeanExecutionTime	00:09:41	00:01:59	00:10:12	00:00:15	00:01:30
ExecutionTimeDeviation	00:01:18	00:00:14	00:01:17	00:00:02	00:00:14
JobsMean	2	1	5	1	3
JobsDeviation	0.0	0.0	0.0	0.0	0.0
TotalRuns	6508	2967	61	85	6485
RunsAnalyzed	100	100	61	85	100
MeanTimeBetweenRuns	02:31:09	05:06:18	1 day 8:24:12	1 day 0:00:13	02:31:09
DeviationTimeBetweenRuns	04:17:28	12:58:42	20:11:52	00:08:25	04:17:28
RunsFreq	2022-06: 84.0%	2022-06: 30.0%	2022-06: 06.56%	2022-06: 08.24%	2022-06: 84.0%
	2022-05: 16.0%	2022-05: 70.0%	2022-05: 32.79%	2022-05: 36.47%	2022-05: 16.0%
			2022-04: 36.07%	2022-04: 35.29%	
			2022-03: 24.59%	2022-03: 20.0%	

Tabela 1. Resultados para o repositório Django

Todos apresentam uma elevada taxa de sucesso e alguns pipelines são geralmente ativados por alterações na ramificação principal (*Schedule Tests* e *Schedule*), enquanto os demais são ativados por ramificações secundárias. Além disso, analisando a média e o desvio padrão do tempo de execução dos fluxos de trabalho, vemos que os pipelines são bem rápidos.

Por fim, temos que os pipelines são simples em geral, apresentando uma média de no mínimo um e no máximo cinco jobs. Como nenhum deles foi modificado desde sua criação, todos tem um desvio padrão de *jobs* igual a zero. Ainda, podemos observar que os fluxos de trabalho mais rápidos apresentam menor média de *jobs*. Apesar disso, o pipeline *Tests* possui tempo médio de execução elevado se comparado a sua pequena quantidade de *jobs*, indicando que ele apresenta procedimentos mais complexos ou que o fluxo necessita de melhorias.

CONCLUSÕES:

Pelo exemplo de aplicação, observamos a capacidade da ferramenta de analisar pipelines de CI/CD sobre vários aspectos, como desempenho de execução, frequência de uso e nível de automação. Desse modo, o CI-Scavenger se mostrou pertinente em recuperar informações relevantes relacionadas à estrutura dos pipelines e suas execuções. Provando seu valor em auxiliar equipes de desenvolvimento a compreender o nível de adoção de práticas CI/CD em repositórios de software livre hospedados no GitHub.

Para o futuro, planejamos integrar a ferramenta Garimpeiro ao CI-Scavenger com o objetivo de aprimorar a capacidade de análise dessas técnicas. Além disso, também desejamos incluir a análise do nível de dependência entre pipelines presentes no mesmo repositório.

BIBLIOGRAFIA

1. Humble, Jez; Farley, David. 2010. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson Education.

2. Destro, G.A.; de França, B.B.N. 2020. Mining Software Repositories for the Characterization of Continuous Integration and Delivery. In Proceedings of the 34th Brazilian Symposium on Software Engineering (pp. 664-669).