# Análise das Extensões de Teste mais Utilizadas em Frameworks Modernos: Um Estudo sobre Pytest, Jest e Cypress


**Aluno:** Daniel Pires Quirino
**Orientador:** André Hora
**Tipo de Pesquisa:** Tecnológica

## Introdução

A garantia de qualidade em sistemas de software é uma preocupação constante para engenheiros de softwares, especialmente à medida que as aplicações se tornam mais complexas e integradas. Nesse contexto, os frameworks de teste, como **Pytest, Jest e Cypress**, emergiram como ferramentas essenciais, permitindo a automação de testes e ajudando a assegurar a integridade e a funcionalidade dos sistemas. Estes frameworks são frequentemente complementados por uma variedade de plugins que estendem suas funcionalidades, adaptando-se a necessidades específicas de projetos e ambientes de desenvolvimento. Apesar da importância desses plugins, há uma lacuna significativa na literatura e nos estudos empíricos focados na análise e no uso efetivo dessas extensões.

Diante desse cenário, este trabalho se debruçou sobre o seguinte problema central: "Apesar da ampla utilização de frameworks de teste em desenvolvimento de software, pouco se sabe sobre quais plugins são mais utilizados, por que são escolhidos e como impactam a eficácia dos testes em ambientes de desenvolvimento modernos."

Este problema conduz à questão central que norteou a investigação: **"Quais são os plugins mais utilizados nos frameworks de teste Pytest, Jest e Cypress e por que esses plugins são preferidos pelos desenvolvedores e engenheiros de teste?"**

Responder a essa questão não apenas preenche uma lacuna no conhecimento existente, mas também fornece uma base empírica para a seleção de ferramentas de teste, contribuindo para a melhoria da qualidade e eficiência no desenvolvimento de software. Além disso, a compreensão dos fatores que influenciam a escolha desses plugins pode oferecer diretrizes valiosas para o desenvolvimento de novas extensões que melhor atendam às demandas contemporâneas do mercado de tecnologia.

## Importância do Trabalho

**Contribuição para a Comunidade de Desenvolvimento de Software:** Ao identificar e analisar os plugins mais utilizados nesses frameworks de teste, este trabalho oferece insights valiosos para desenvolvedores e engenheiros de teste sobre quais ferramentas são eficazes em diferentes cenários.

**Avanço Acadêmico:** A investigação contribui para o corpo acadêmico de conhecimento em engenharia de software, especialmente na área de testes de software. Embora existam muitos estudos sobre frameworks de teste, poucos se aprofundam especificamente nos plugins.

**Impacto na Qualidade do Software:** Com a compreensão de como determinados plugins influenciam a prática de teste, é possível promover uma melhor qualidade do software desenvolvido. Isso se traduz em softwares mais robustos, confiáveis e que melhor atendem às necessidades dos usuários finais.

**Direcionamento para Desenvolvedores de Ferramentas:** Os resultados deste estudo podem informar os criadores de ferramentas de teste sobre as funcionalidades mais valorizadas pelos usuários, orientando o desenvolvimento de novos plugins e a melhoria dos existentes. Esse feedback é crucial para que as ferramentas evoluam em consonância com as necessidades reais dos desenvolvedores.

**Redução de Custos e Tempo de Desenvolvimento:** Ao facilitar a escolha dos plugins mais adequados para projetos específicos, este estudo pode ajudar equipes de desenvolvimento a reduzir o tempo e os custos associados aos ciclos de teste e manutenção de software, um benefício significativo em um mercado altamente competitivo e em rápida evolução.

A importância deste trabalho se estende, portanto, desde o aumento da eficiência operacional até contribuições significativas para a literatura acadêmica e prática profissional na área de tecnologia. Ao elucidar as preferências e tendências no uso de plugins de teste, abrindo caminho para inovações e melhorias contínuas na qualidade de software.

## Objetivos e Metodologias

O trabalho teve como objetivo geral investigar a utilização de plugins nos frameworks de teste Pytest, Jest e Cypress, através da análise empírica de dados quantitativos sobre essas extensões. Para alcançar este objetivo, os seguintes objetivos específicos foram estabelecidos:

**1 - Desenvolver um Web Crawler:** Construir um crawler para automatizar a coleta de dados sobre os plugins disponíveis nas páginas oficiais dos frameworks Pytest, Jest e Cypress. O crawler deverá identificar e extrair links que direcionam para as páginas dos plugins.

**2 - Coletar Dados dos Plugins:** Utilizar os links obtidos para acessar repositórios no GitHub e registros em gerenciadores de pacotes como PyPI, com o intuito de extrair informações relevantes como quantidade de downloads, número de estrelas, últimas atualizações, entre outros.

**3 - Organizar os Dados em uma Planilha:** Consolidar todos os dados coletados em uma planilha do Excel, estruturando as informações de maneira que facilitem a análise e comparação.

**4 - Analisar e Categorizar os Plugins:** Analisar os dados para identificar padrões de utilização, preferências dos desenvolvedores e as características mais valorizadas nos plugins. Categorizar os plugins com base em critérios como popularidade, funcionalidade e

frequência de atualizações.

**5 - Elaborar Conclusões sobre a Utilização dos Plugins:** Com base na análise realizada, elaborar conclusões sobre os fatores que influenciam a escolha de plugins nos frameworks estudados e como eles contribuem para a eficácia dos processos de teste de software

## Resultados Alcançados

Neste capítulo, detalhamos as atividades práticas realizadas para atingir os objetivos propostos neste trabalho, abrangendo desde o desenvolvimento das ferramentas de coleta de dados até a organização e análise inicial das informações dos plugins.

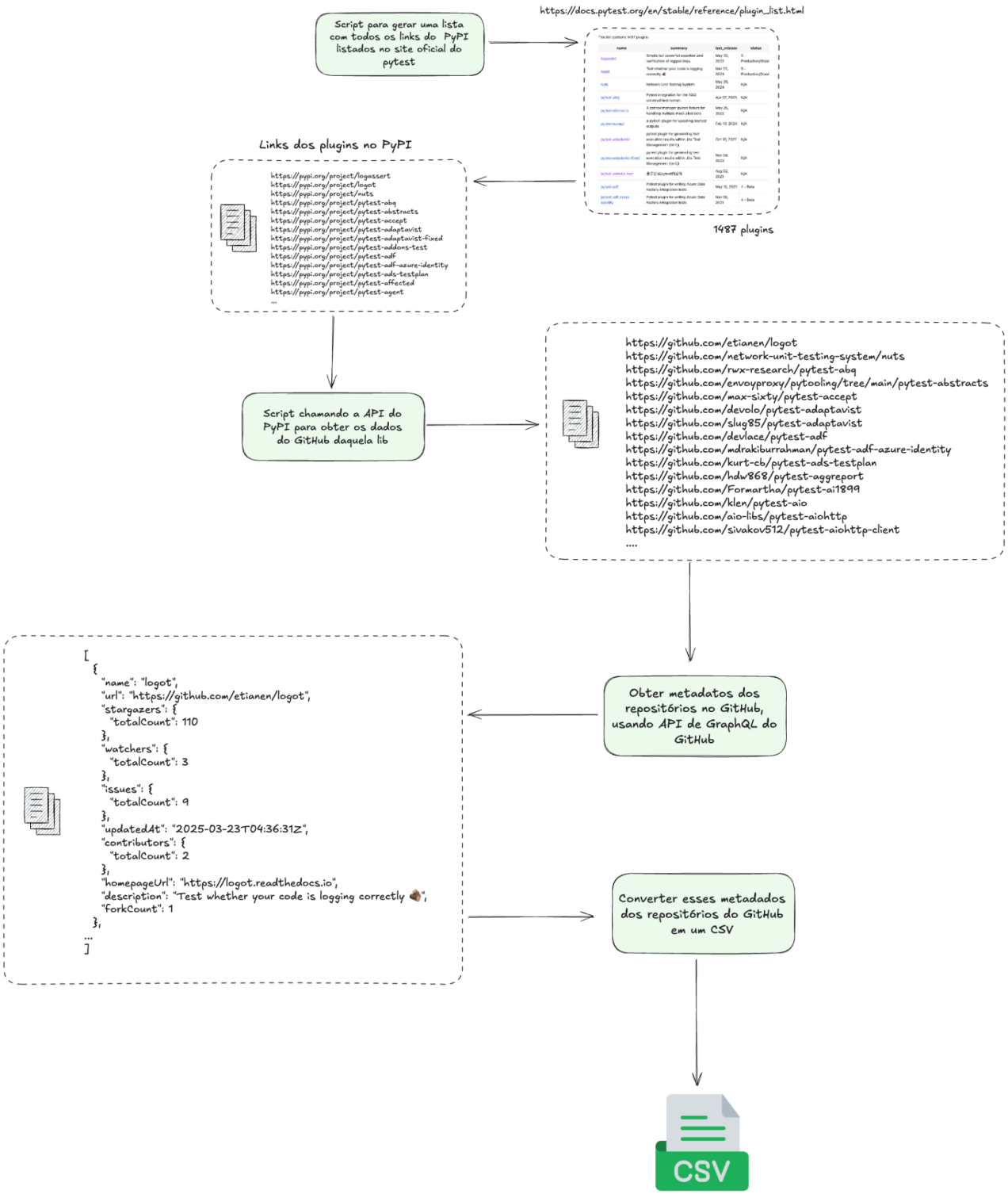
Os resultados alcançados neste estudo são a materialização dos objetivos específicos delineados, que visavam investigar a utilização de plugins nos frameworks de teste Pytest, Jest e Cypress. As etapas de desenvolvimento de ferramentas de automação e a subsequente coleta de dados constituem a base empírica para as análises e conclusões deste trabalho.

**1.1. Desenvolvimento do Web Crawler para Identificação de Plugins:** Para automatizar a identificação dos plugins, desenvolvemos um conjunto de *web crawlers* personalizados para cada framework: **Pytest**, **Jest** e **Cypress**. A escolha da tecnologia para cada *crawler* foi guiada pela natureza dos websites e pela estrutura de dados disponível.

**1.2. Coleta de Dados Detalhados dos Plugins em Repositórios e Gerenciadores de Pacotes:** Após a identificação dos URLs dos plugins, a segunda fase consistiu em acessar esses links para coletar dados quantitativos e qualitativos sobre cada extensão. As principais fontes de dados foram o GitHub (para métricas de código aberto) e gerenciadores de pacotes como  PyPI (para métricas de uso e distribuição).

**1.3. Armazenamento e Organização dos Dados em Planilha**: Após a etapa de coleta automatizada, a massa de dados brutos sobre os plugins dos frameworks Pytest, Jest e Cypress necessitava de uma estrutura organizada para facilitar a análise e a comparação. Para isso, optou-se por consolidar todas as informações em uma **planilha de dados**, utilizando o **Google Sheets** como ferramenta principal.

IC - Plugin de tests



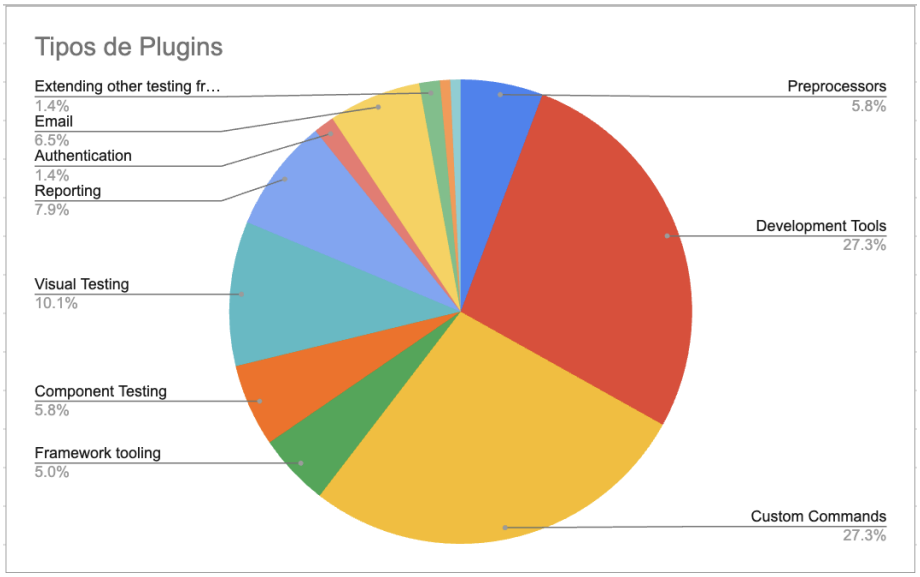

Exemplo do fluxo para a geração de dados dos plugins do pytest

### 1.4. Análise e Categorização Preliminar dos Plugins

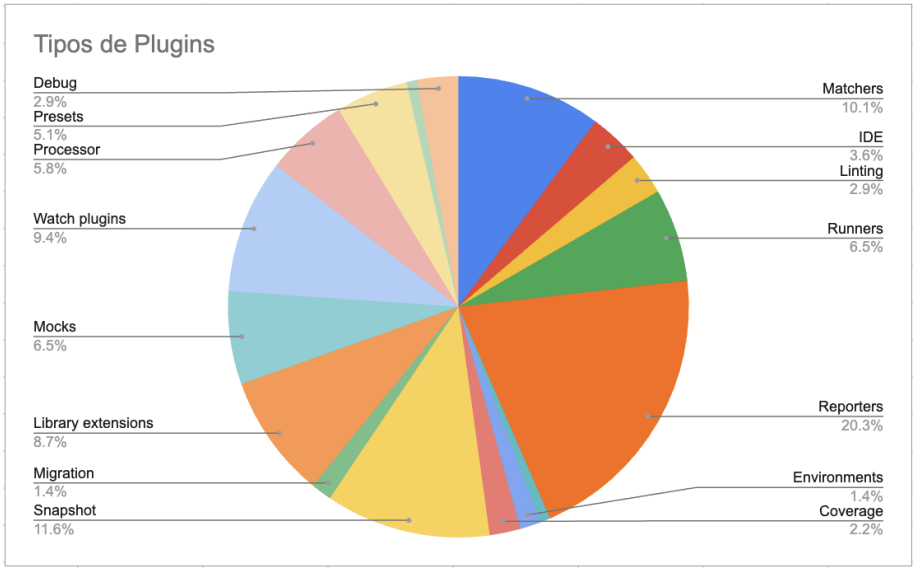
Com os dados dos plugins devidamente coletados, organizados e armazenados no Google Sheets, foi possível dar início à fase de **análise e categorização**. Essa etapa preliminar envolveu a exploração dos dados brutos para identificar os primeiros padrões de utilização, tendências de popularidade e as principais funcionalidades oferecidas pelas extensões.

Através da aplicação de filtros e ordenações na planilha, foi possível extrair *insights* valiosos sobre quais plugins são mais adotados pela comunidade e quais características se destacam em cada ecossistema (Pytest, Jest e Cypress)

Dados dos plugins do Cypress por tipo




Dados dos Plugins do Jest por tipo



# Análise dos Plugins Pytest:

**A Prioridade em Métricas de Qualidade de Código e Relatórios Abrangentes Domina a Preferência da Comunidade Pytest.**

Os dados mostram que os plugins com maior número de estrelas e, consequentemente, maior reconhecimento da comunidade, estão diretamente relacionados à cobertura de código e relatórios de resultados. O plugin pytest-cov, com 1886 estrelas, destaca-se como o mais popular. Sua descrição "Coverage plugin for pytest" indica uma preocupação central dos desenvolvedores em garantir a qualidade do código através da verificação da cobertura de testes.

Além disso, plugins como pytest-html (com 735 estrelas, para "generating HTML reports for pytest results") e allure-pytest (com 1683 estrelas, para "Framework integration with PyTest", embora esteja marcado como depreciado, sua alta contagem de estrelas reflete uma necessidade histórica por relatórios ricos) demonstram a importância da visualização clara e detalhada dos resultados de testes.

Isso sugere que os desenvolvedores valorizam ferramentas que não apenas executam testes, mas também fornecem feedback robusto e métricas tangíveis sobre a saúde e a qualidade do código-fonte. A capacidade de gerar relatórios compreensíveis e de monitorar a cobertura do código é crucial para a manutenção de grandes bases de código e para a comunicação do progresso dos testes em equipes.

**Ferramentas de Orquestração e Paralelização de Testes são essenciais para Eficiência em Ambientes Modernos.**

Outra categoria de plugins que se destaca em popularidade são aqueles voltados para a otimização da execução de testes, especialmente em termos de paralelização e distribuição. O pytest-xdist, com 1616 estrelas, é um exemplo proeminente, descrito como um "pytest plugin for distributed testing and loop-on-failures testing modes". Sua alta adoção reflete a necessidade crescente de acelerar os ciclos de feedback dos testes em projetos complexos e em integração contínua (CI). A capacidade de distribuir testes em múltiplos workers ou CPUs permite uma execução muito mais rápida, reduzindo o tempo de espera dos desenvolvedores.

Da mesma forma, pytest-asyncio (1519 estrelas), que oferece "Asyncio support for pytest", é fundamental para testar aplicações assíncronas de forma eficiente. Estes plugins impactam diretamente a eficácia dos testes ao permitir que suítes de testes grandes e demoradas sejam executadas em frações do tempo, tornando a automação de testes mais prática e incentivando uma maior frequência de execução, o que, por sua vez, contribui para a detecção precoce de bugs.

**A Flexibilidade e o Controle Detalhado sobre o Comportamento do Teste são Altamente Valorizados.**

Plugins que oferecem maior controle e flexibilidade sobre como os testes são executados e como interagem com o ambiente de desenvolvimento também demonstram alta popularidade. O pytest-mock (1950 estrelas), descrito como uma "Thin-wrapper around the mock package for easier use with pytest", é um dos plugins mais utilizados. Isso indica a importância de isolamento de dependências externas e a criação de ambientes controlados para testes unitários e de integração, garantindo que os testes sejam rápidos, confiáveis e determinísticos.

Além disso, a presença de pytest-django (1457 estrelas, "A Django plugin for pytest") ressalta a importância de integração específica com frameworks de aplicação e o uso de fixtures para simplificar a configuração de testes. A capacidade de "mockar" objetos, simular comportamentos e interagir de forma controlada com componentes externos permite que os desenvolvedores escrevam testes mais precisos e robustos, adaptando o framework de teste às necessidades específicas do projeto.

## Análise dos Plugins do Cypress:

O Cypress, sendo um framework de teste *end-to-end* com foco em testes de interface de usuário, naturalmente terá uma preferência por plugins que auxiliam na simulação de interações do usuário, na integração com pipelines de CI/CD e na garantia da qualidade visual e acessibilidade

**Testes Visuais e de Acessibilidade Ganham Destaque, Sinalizando uma Crescente Preocupação com a Experiência do Usuário e Conformidade.**

Uma categoria de plugins com forte presença e relevância para a comunidade Cypress é a de testes visuais e de acessibilidade. O plugin Cypress Visual Regression (656 estrelas) e cypress-axe (633 estrelas, para "Test accessibility with axe-core in Cypress") são exemplos claros dessa prioridade. A popularidade de Cypress Visual Regression indica que a validação da interface do usuário através de comparações visuais é fundamental para garantir que as alterações no código não introduzem regressões inesperadas na aparência do sistema. Paralelamente, cypress-axe reflete a crescente conscientização e demanda por acessibilidade web, garantindo que as aplicações sejam utilizáveis por todos os usuários.

Esses plugins são cruciais para a eficácia dos testes, pois permitem detectar problemas que testes funcionais tradicionais poderiam ignorar, como mudanças no layout ou falhas na conformidade com padrões de acessibilidade, resultando em uma experiência de usuário mais inclusiva e de maior qualidade.

**Extensões de Comandos Personalizados e de Mock de Requisições são Fundamentais para Testes Comportamentais e Isolados.**

A expressiva quantidade de plugins focados em comandos personalizados (Custom Commands) e na manipulação de requisições demonstra a importância da flexibilidade e do controle fino no ambiente de teste do Cypress. Plugins como cypress-testing-library (1812 estrelas, para "Simple and complete custom Cypress commands and utilities that encourage good testing practices") e cypress-real-events (811 estrelas, para "Fire native system events from Cypress") evidenciam a necessidade de simular interações de usuário de forma mais realista ou de abstrair complexidades da DOM para facilitar a escrita de testes.

Além disso, a presença de @bahmutov/cy-api (2125 estrelas) para "end-to-end API testing" e cypress-graphql-mock-network (562 estrelas) para "Simple network mocking for your GraphQL API" destaca a importância de mockar ou interceptar requisições de rede para isolar o front-end durante os testes ou para simular diferentes cenários de resposta da API. Esses plugins impactam a eficácia permitindo testes mais focados, rápidos e menos suscetíveis a falhas de rede, o que é vital para um desenvolvimento front-end robusto.

## Análise dos Plugins do Jest:

O Jest é conhecido por ser um framework de teste de JavaScript com forte foco em testes unitários e de integração de componentes React/Node.js, além de oferecer um ecossistema rico em funcionalidades como *snapshots*, *mocking* e relatórios.

**A Capacidade de Estender Assertions e Manipular o DOM Virtual é Crucial para Testes Frontend Robustos.**

A popularidade dos plugins do Jest é fortemente dominada por extensões que aprimoram as capacidades de assertividade (matchers) e permitem a interação e validação do DOM virtual. O plugin jest-extended (com 2347 estrelas, para "Additional Jest matchers") e @testing-library/jest-dom (com 4528 estrelas, para "Custom jest matchers to test the state of the DOM") são os exemplos mais claros dessa tendência. O alto número de estrelas de jest-extended demonstra a necessidade dos desenvolvedores por um conjunto mais rico e expressivo de matchers para escrever asserções claras e concisas, indo além do que o Jest oferece nativamente. Mais notavelmente, @testing-library/jest-dom é quase onipresente em projetos React/frontend, permitindo que os testes validem o estado renderizado do DOM de forma semântica e amigável ao usuário. Isso ressalta que a comunidade Jest prioriza ferramentas que facilitam a escrita de testes legíveis, com foco na experiência do usuário e na verificação do comportamento real dos componentes na interface.

**Ferramentas de *Mocking* e Simulação de Ambiente são Fundamentais para o Isolamento e Controle de Dependências.**

Outra área de alta demanda para os plugins do Jest é a de simulação e mocking de dependências. O jest-fetch-mock (com 892 estrelas, para "Jest mock for fetch") e jest-mock-extended (com 877 estrelas, para "Type safe mocking extensions for Jest") ilustram essa necessidade. A capacidade de simular requisições de rede (fetch) é vital para testes unitários e de integração de componentes que interagem com APIs externas, garantindo que os testes sejam rápidos e não dependam de serviços reais. jest-mock-extended, por sua vez, oferece um mocking mais seguro e tipado, o que é especialmente valioso em projetos TypeScript.

Esses plugins permitem que os desenvolvedores isolem a unidade de código sob teste de suas dependências, criando ambientes controlados e reproduzíveis. Isso melhora significativamente a confiabilidade dos testes e acelera o ciclo de desenvolvimento, ao remover a necessidade de configurar backends complexos para cada execução de teste.

**A Produtividade do Desenvolvedor é Impulsionada por Integração com IDEs e Ferramentas de Relatório Detalhadas.**

A popularidade de plugins que aprimoram a experiência de desenvolvimento dentro do ambiente de trabalho (IDEs) e a geração de relatórios claros é um indicativo forte da preocupação com a produtividade. vscode-jest (com 2872 estrelas, para "The optimal flow for Jest based testing in VS Code") é um dos plugins mais aclamados, facilitando a execução e depuração de testes diretamente no editor de código. Isso agiliza o ciclo de feedback e melhora a eficiência dos desenvolvedores.

Além disso, plugins de relatório como jest-junit (495 estrelas, para "A Jest reporter that creates compatible junit xml files") e jest-html-reporter (2806 estrelas, para "Jest test results processor for generating a summary in HTML") demonstram a importância de visualizar os resultados dos testes de forma compreensível e padronizada. Relatórios detalhados, muitas vezes em formato HTML ou JUnit XML, são essenciais para depuração, para o acompanhamento da qualidade em sistemas de CI e para a comunicação entre equipes, validando a eficácia e a praticidade das ferramentas de teste.

## Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo central investigar os plugins mais utilizados nos *frameworks* de teste Pytest, Jest e Cypress, e compreender as razões subjacentes à sua preferência pelos desenvolvedores e engenheiros de teste. Através da coleta e análise de dados quantitativos de plataformas como GitHub, PyPI, foi possível identificar padrões e tendências cruciais que refletem as prioridades e desafios enfrentados pela comunidade de desenvolvimento de software em diferentes ecossistemas tecnológicos.

Embora cada *framework* opere em um ecossistema tecnológico distinto (Python *backend*/geral vs. JavaScript *frontend*/Node.js), a análise revelou algumas prioridades universais e outras

específicas. Em síntese, enquanto todos os *frameworks* compartilham a busca por qualidade e eficiência, as nuances de suas comunidades e os desafios intrínsecos de seus domínios de aplicação (backend vs. frontend) moldam as funcionalidades e os tipos de plugins mais valorizados. Os plugins mais populares são, portanto, aqueles que melhor preenchem as lacunas de funcionalidade e otimização para os casos de uso mais comuns e críticos em seus respectivos contextos.

## Trabalhos Futuros

Com base nas conclusões e nas limitações deste estudo, diversas direções para pesquisas futuras podem ser exploradas:

**Estudo Qualitativo Aprofundado:** Conduzir entrevistas ou pesquisas com desenvolvedores que utilizam esses plugins para entender as motivações e os *drivers* qualitativos por trás de suas escolhas, como usabilidade, suporte a casos de uso complexos e impacto real na produtividade das equipes.

**Análise Evolutiva da Popularidade:** Realizar um estudo longitudinal para rastrear a evolução da popularidade dos plugins ao longo do tempo, identificando tendências de ascensão e declínio e correlacionando-as com eventos no ciclo de vida dos *frameworks* ou com novas demandas da indústria.

**Impacto no Desempenho dos Testes:** Avaliar empiricamente como a utilização de determinados plugins afeta o desempenho da suíte de testes (tempo de execução, consumo de memória) e a estabilidade dos testes (flakiness), fornecendo dados mais objetivos sobre sua eficácia.

**Expansão para Outros Ecossistemas:** Ampliar a pesquisa para incluir outros *frameworks* de teste relevantes em diferentes linguagens e plataformas, como Go, Ruby, Java, ou plataformas *mobile*, para construir um panorama mais abrangente das tendências de plugins.

**Desenvolvimento de Ferramentas de Recomendação:** Utilizar os dados coletados e as conclusões para desenvolver uma ferramenta ou sistema que possa recomendar plugins a desenvolvedores com base nas características de seus projetos ou nas necessidades de teste que desejam abordar.

Este trabalho serve como um ponto de partida para uma compreensão mais aprofundada do papel vital que os plugins desempenham na otimização e eficácia dos processos de teste em *frameworks* modernos, abrindo caminho para o desenvolvimento de soluções ainda mais alinhadas às necessidades da engenharia de software contemporânea.