Projeto de Pesquisa

Parece funcionar, mas...

Qualidade do Teste de Software na era dos Modelos de Linguagem

Prof. Dr. Marcelo de Almeida Maia

Universidade Federal de Uberlândia

marcelo.maia@ufu.br

2025

Resumo da proposta

Em 2025, a hipótese de que os modelos de linguagens de larga escala (LLMs[1]) vão desempenhar um papel central no desenvolvimento de software parece não ter maiores objeções na comunidade acadêmica. Muito tem se investigado sobre as aplicações de LLMs nas diversas etapas dos processos de Engenharia de Software. Em particular, os LLMs tem demonstrado uma capacidade robusta de geração de código-fonte com uma qualidade que permite que os mesmos possam ser executados e testados. Além disso, os LLMs também são capazes de analisar código e sugerir melhorias de diversos tipos: como melhoria de legibilidade, melhoria de anomalias de código (*bad smells*, entre várias outras.

Apesar de várias promessas relacionadas à aplicação de LLMs na construção de código-fonte, entendemos que existem perigos à medida que o uso do ferramental de construção de código baseado em LLMs vai se tornando cada vez mais pervasivo. Neste sentido, o objetivo central desta proposta é investigar diferentes aspectos relacionados à garantia da qualidade do código nesta era de desenvolvimento de software com auxílio de LLMs. Neste sentido, teremos como foco desta proposta, investigar lacunas relacionadas a diferentes aspectos de LLMs relacionadas ao teste de software.

Para identificar estas lacunas, iniciamos uma revisão sistemática da literatura, coletando artigos de 2019 a 2024, que mostrassem o uso de LLMs no apoio ao teste de software. Foram selecionados 139 artigos, e uma análise preliminar dos mesmos nos permitiu identificar os principais pontos investigados durante este período. Os pontos incluem: geração de entradas, casos de teste e scripts; melhoria de métricas de cobertura para testes funcionais; identificação e reparo de testes *flaky*; aplicações em domínios específicos, tais como, *fintechs*, *cloud-edge computing*, e sistemas móveis.

No entanto, mesmo com os avanços significativos na geração de testes automatizados, surgem novos desafios. Aspectos como qualidade interna dos testes, manutenibilidade e legibilidade não foram devidamente explorados. Isto é particularmente interessante em nosso caso, pois se conecta adequadamente com nossa pesquisa recente. Assim, pretendemos focar em alguns problemas pouco investigados: os testes gerados automaticamente por LLMs podem ser difíceis de compreender por desenvolvedores humanos; os testes gerados podem conter *bad smells*, tais como, duplicação, excessos de setup, ou testes longos e complexos; a falta de entendimento sobre como os testes gerados se comportam ao longo do tempo com múltiplas alterações no software.

Nota: O ChatGPT foi utilizado nesta proposta como revisor de texto e de estrutura. Toda a concepção e desenvolvimento das ideias do projeto foram de autoria do proponente até porque a concepção do projeto foi fundamentada em uma revisão sistemática da literatura que estamos conduzindo neste tema.

[1]Do inglês, *Large Language Models*.

1 Introdução

Nos últimos anos, modelos de linguagem de larga escala (LLMs[2]), como GPT, BERT e seus derivados, têm revolucionado a forma como tarefas complexas na Engenharia de Software são conduzidas. Treinados em enormes volumes de dados textuais e de código, esses modelos tem demonstrado capacidade de compreensão semântica e geração de conteúdo, permitindo aplicações que vão desde a escrita automatizada de código até a documentação técnica e suporte à decisão em arquiteturas de sistemas. Esses avanços têm mostrado a possibilidade de impactos positivos, como aumento de produtividade durante diversas etapas do desenvolvimento de software.

Neste projeto, em particular, entendemos ser crucial investigar como o teste de software pode ser mais efetivo neste contexto onde LLMs se tornam cada vez mais pervasivos.

Inclusive no próprio contexto de teste de software, que é uma área amplamente estudada há décadas, os LLMs podem ser usados para a automação de processos de teste. A geração de casos e scripts de teste, validação de dados e até mesmo a identificação de falhas são apenas algumas das tarefas que podem ser possivelmente aprimoradas com o uso desses modelos em alternativa aos métodos tradicionais que foram desenvolvidos ao longo dos anos. No entanto, apesar do potencial, entendemos que existem lacunas técnicas e práticas que ainda precisam ser melhor estudadas.

Para entender melhor estas lacunas, iniciamos uma revisão sistemática da literatura, coletando artigos de 2019 a 2024, que mostrassem o uso de LLMs no apoio ao teste de software. Foram selecionados 139 artigos, e uma análise preliminar dos mesmos nos permitiu identificar alguns os principais pontos investigados durante este período.

Um dos pontos é a geração de dados de entrada, onde os LLMs têm mostrado resultados importantes nesse processo que mostramos a seguir. Liu et al. [2023] apresenta uma abordagem para criar dados de entrada contextualmente relevantes, especialmente em interfaces gráficas. Rasool et al. [2024] exploram a geração de entradas para aplicativos semânticos, enquanto Jiang et al. [2024] investigam como LLMs podem ser ajustados para cenários dirigidos de teste.

Os LLMs também têm sido aplicados na automação da geração de scripts de teste. Estudos como Leotta et al. [2024] avaliam o uso de LLMs para criar scripts de testes End-to-End para Web, enquanto Nguyen et al. [2023] e Bhatia et al. [2024] exploram ferramentas para automação de scripts em testes de APIs e unidades.

Outro ponto que tem sido explorado são alternativas para melhorar a cobertura dos testes, o que é um elemento central para construção de software confiável. Guilherme and Vincenzi [2023] e Alshahwan et al. [2024] destacam como LLMs podem identificar lacunas na cobertura e gerar testes adicionais para preenchê-las. Além disso, Lemieux et al. [2023]

[2]Do inglês, *Large Language Models*

apresenta técnicas para superar limitações comuns em abordagens tradicionais, em particular para escapar de platôs de cobertura em testes gerados automaticamente.

A instabilidade dos testes é um problema crítico na prática de desenvolvimento. Os testes instáveis[3] são testes que passam e falham de forma aleatória, mesmo sem alterações no código ou no ambiente de execução. Chen et al. [2024] propõem uma abordagem neuro-simbólica para identificar e corrigir testes *flaky*, enquanto Akli et al. [2023] explora técnicas de aprendizado do tipo *few-shot learning* para categorizar e priorizar testes instáveis.

O uso de LLMs em domínios específicos tem sido investigados recentemente. Em FinTech, por exemplo, trabalhos como Xue et al. [2024b] mostram como esses modelos podem ser usados em testes de aceitação automatizados. Em áreas como computação em nuvem e sistemas móveis, Xue et al. [2024a] destaca estratégias para adaptar LLMs a requisitos específicos desses contextos. Shin et al. [2024] também mostra como a geração de casos para teste unitário pode se beneficiar da adaptação ao domínio.

Embora os avanços mencionados anteriormente sejam promissores, pudemos perceber lacunas que precisam ser melhor estudadas. Este projeto propõe abordar três questões principais:

- **Presença de *bad smells*:** Testes gerados frequentemente apresentam problemas de qualidade, como duplicação, setups excessivos e métodos longos e complexos. Esta é uma pesquisa de longa data como mostra Panichella et al. [2022]. Entretanto, estudos que avaliam testes gerados por LLMs ainda não estão amplamente disponíveis.

- **Legibilidade dos testes gerados automaticamente:** Testes gerados por LLMs podem ser difíceis de compreender, o que compromete sua adoção e manutenção. Shamshiri et al. [2018] mostrou que teste automaticamente gerados são em geral mais difíceis de ler e interpretar que o escritos manualmente. Inclusive, Roy et al. [2021] propôs DeepTC-Enhancer para melhorar a legibilidade de testes gerados automaticamente. Entretanto, se considerarmos os testes gerados por LLMs, a área de legibilidade de código ainda carece de estudos mais aprofundados sobre a efetiva prática dos desenvolvedores em relação a este aspecto.

- **Manutenção de testes ao longo do tempo:** A evolução do software frequentemente exige a adaptação dos testes. Le Dilavrec et al. [2021] propõe uma abordagem automatizada para detectar co-evolução entre código e teste independentemente da história do *commits*. Kim et al. [2021] investigou os *smells* de teste sob a perspectiva da sua evolução no tempo. Estes estudos não consideraram testes gerados por LLMs.

O objetivo deste projeto é avançar o estado da arte, propondo novas abordagens para resolver esses problemas e validando as soluções propostas em cenários reais de teste de

[3]Do inglês, *flaky tests*

software.

2 Justificativa

A relevância deste projeto pode ser justificada por três aspectos principais: avanços tecnológicos, desafios da prática e lacunas na literatura.

2.1 Impacto Tecnológico

Modelos de linguagem de larga escala têm revolucionado a forma como a inteligência artificial é aplicada na engenharia de software. Contudo, ainda não está claro que estes modelos estejam livres de gerar código subótimo e frequentemente afetado por padrões de má qualidade, em particular, no código utilizado para testes. Assim, a investigação de como estes códigos estão sendo gerados ao longo do tempo, pode trazer luz para mitigar os potenciais problemas, garantindo que os benefícios da automação não sejam ofuscados por limitações de qualidade nos testes. Os resultados deste projeto podem melhorar a qualidade dos testes em ambientes que adotem LLMs para auxiliar na construção dos testes, seja por meio de ferramentas que pretendemos disponibilizar, seja por evidências que nossas investigações poderão trazer para melhorar a prática de evolução e manutenção dos testes pelas equipes de desenvolvimento.

2.2 Impacto na produtividade

Testes automatizados são cruciais para garantir a qualidade e a confiabilidade do software, mas sua escrita manual é frequentemente custosa e sujeita a erros. Embora LLMs possam ajudar a diminuir esse esforço, problemas como *setups* excessivos, assertivas pouco claras e testes frágeis (*test smells*) podem limitar a produtividade das equipes de desenvolvimento, aumentando o custo de manutenção e reduzindo a confiança na automação. Assim, ao revisitar e melhorar a prática de análise de smells e respectiva refatoração em testes gerados por LLMs espera-se que a evolução e manutenção destes testes sejam mais produtivas, bem como o desenvolvedor possa ter confiança para aprimorar a qualidade dos testes de forma a torná-los mais efetivos na detecção de bugs.

2.3 Impacto científico

Sobre o ponto de vista científico, fundamentamos este projeto iniciando uma revisão sistemática da literatura que mostrasse o estado da arte sob o ponto de vista científico no apoio de LLMs para geração de teste de software. Selecionamos cerca de 130 publicações

para identificarmos lacunas na literatura e organizarmos nossos objetivos de pesquisa científica em direção ao preenchimento destas lacunas.

Embora a literatura sobre test smells e legibilidade de código esteja razoavelmente bem estabelecida, estudos que conectem essas áreas ao uso de LLMs são escassos. A maioria dos trabalhos foca em ferramentas de geração de testes ou na identificação de smells, mas poucos exploram o efetivo impacto da qualidade dos testes gerados por LLMs. Assim, nosso trabalho vai no sentido de preencher esta lacuna de pesquisa.

Ferramentas e diretrizes que aprimorem a qualidade dos testes gerados por LLMs podem aumentar a eficiência das equipes de desenvolvimento, reduzindo custos e tempo de entrega de software. Além disso, o projeto pretende contribuir em questões científicas ao mostrar o estado atual de como os teste gerados por LLMs se comportam em termos de diferentes avaliações de legibilidade e manutenibilidade, bem elucidar conhecimento sobre como ajustar modelos de linguagem para tarefas específicas na construção dos testes. Esta também é uma lacuna que ainda não está preenchida em termos científicos.

Por fim, ao disponibilizar ferramentas e benchmarks, este projeto pode ampliar o acesso a soluções de alta qualidade, permitindo colaboração entre pesquisadores e profissionais da área. Assim, estaremos também promovendo os princípios de Ciência Aberta.

3 Perguntas de pesquisa

Problema de Pesquisa

O uso crescente de modelos de linguagem de larga escala (LLMs) na geração de testes automatizados tem trazido avanços significativos para a automação de tarefas em teste de software. Contudo, pode ser que os testes gerados por LLMs apresentem problemas de qualidade conhecidos como *test smells*, além de dificuldades relacionadas à legibilidade e manutenibilidade. Esses problemas podem comprometer a eficiência dos testes e possivelmente aumentar os custos de desenvolvimento e manutenção.

Embora existam ferramentas para detecção de *test smells* e pesquisas sobre métricas de qualidade de teste, poucas abordagens investigam como estas soluções se aplicam para testes gerados por LLMs. Não está claro como esses modelos podem ser ajustados para evitar a geração de *test smells* e como métodos automáticos se aplicam para refatorar os problemas existentes. Assim, o problema de pesquisa central deste projeto pode ser formulado da seguinte maneira:

Qual é a prevalência e impacto de *test smells*, e como mitigar estes smells e melhorar a legibilidade e manutenibilidade de testes automatizados gerados por LLMs, promovendo a qualidade desses testes?

Para detalhar o problema de pesquisa, o projeto se propõe a responder às seguintes

perguntas de pesquisa:

1. *Ocorrência de Test Smells em Testes Gerados por LLMs*

    1.1. Quais tipos de *test smells* são mais comuns em testes gerados por LLMs?

    1.2. Como a presença de *test smells* afetam a cobertura, confiabilidade e tempo de execução dos testes gerados?

2. *Legibilidade de Testes Gerados por LLMs*

    2.1. Quais são as métricas mais apropriadas para avaliar a legibilidade de testes gerados por LLMs?

    2.2. Como os desenvolvedores percebem a legibilidade desses testes gerados?

    2.3. Como e se os desenvolvedores atuam para melhorar a legibilidade de testes gerados?

3. *Manutenibilidade de Testes Gerados*

    3.1. Quais desafios surgem na manutenção de testes gerados por LLMs frente a mudanças no código de produção?

    3.2. Como os testes gerados evoluem ao longo do tempo e como co-evoluem com o código?

    3.3. Como os test smells se relacionam com a atividade para manutenção de testes?

4. *Detecção e Refatoração Automática de Test Smells*

    4.1. Quão eficazes são os algoritmos automáticos para detectar e corrigir test smells em testes gerados por LLMs?

    4.2. Quais são as limitações atuais dessas ferramentas e como elas podem ser aprimoradas?

    4.3. Até que ponto a remoção de *test smells* melhora a eficiência dos testes?

5. *Melhoria na Geração de Testes com LLMs*

    5.1. Quais modificações em prompts e configurações de treinamento podem reduzir a geração de test smells?

    5.2. Como os LLMs podem ser ajustados para gerar testes mais legíveis?

4 Objetivos Geral e Específicos

O objetivo geral deste projeto é investigar como mitigar problemas de *test smells*, legibilidade e manutenibilidade em testes automatizados gerados por LLMs, propondo ferramentas, diretrizes e melhorias nos modelos para promover a qualidade desses testes. Isso envolve tanto a análise dos problemas (test smells, dificuldades de manutenção e legibilidade) quanto a proposta de soluções práticas (ferramentas automáticas, *guidelines* e melhorias nos LLMs).

4.1 Objetivos Específicos

1. *Identificar e mapear os tipos mais comuns de test smells em testes gerados por LLMs.*

   Este objetivo visa construir um entendimento baseado em evidências[4] sobre os problemas de *smells* mais frequentes nos testes gerados. A partir dessa análise, será possível desenvolver estratégias específicas para mitigá-los.

2. *Avaliar experimentalmente a legibilidade dos testes automatizados, considerando métricas objetivas e a percepção dos desenvolvedores.* Este objetivo envolve a medição de fatores como tamanho, clareza e organização dos testes, conseguido por meio de linters, tais como, SonarLint, PMD, Checkstyle, além de realizar estudos de mineração para capturar percepção de desenvolvedores para validar aspectos qualitativos.

3. *Examinar os desafios associados à manutenção de testes gerados, incluindo sua co-evolução com o código.* Testes com qualidade devem ser resilientes e eventualmente fáceis de se adaptarem a mudanças no código que eles validam. Este objetivo explora como os testes gerados se comportam em cenários reais de evolução de software, além de investigar como os problemas relacionados a como *test smells* e legibilidade impactam a manutenção.

4. *Aplicar métodos automáticos para detecção e refatoração de test smells em testes gerados.* Este objetivo é aplicado e técnico, buscando utilizar e eventualmente adaptar ferramentas que identifiquem e corrijam (ou sugiram correções) automaticamente problemas nos testes, como setups excessivos, métodos longos ou duplicação de código.

5. *Propor melhorias nos prompts e ajustes nos modelos de LLMs para reduzir a geração de test smells e aprimorar a legibilidade dos testes.* Melhorar o uso dos próprios LLMs para evitar a geração de problemas é um passo preventivo. Esse objetivo pode incluir a experimentação com prompts mais eficazes ou ajustes no treinamento e *fine-tuning* dos modelos.

[4] Do inglês, *empirical*

6. *Avaliar o impacto da remoção de test smells na eficiência, cobertura e confiabilidade dos testes.* Aqui, busca-se quantificar como a mitigação de problemas melhora os resultados práticos. A avaliação pode incluir métricas como tempo de execução dos testes, abrangência de cobertura e incidência de falsos positivos/negativos.

7. *Disponibilizar ferramentas e benchmarks como projetos open-source para a comunidade acadêmica e industrial.* Este objetivo garante que os resultados da pesquisa tenham impacto direto, promovendo a ciência aberta e facilitando a replicação de estudos futuros. Além disso, ferramentas de código aberto ajudam a aumentar a confiabilidade e a colaboração.

5 Metas

1. **Identificação e Catalogação de Test Smells Específicos**. Mapeamento e documentação dos principais *test smells* encontrados em testes gerados por LLMs, diferenciando-os daqueles observados em testes manuais ou gerados por ferramentas tradicionais.

2. **Ferramentas de Detecção e Refatoração Automática**. Desenvolvimento e validação de ao menos duas ferramentas automáticas para detecção e sugestão de refatoração de *test smells* específicos em testes gerados por LLMs, utilizando Java e Python.

3. **Estudo do Impacto dos Test Smells**. Obtenção de evidências que correlacionem ou não *test smells* com métricas de qualidade, como confiabilidade, cobertura de código e propensão a defeitos.

4. **Métricas e Diretrizes de Avaliação de Legibilidade**. Obtenção e validação de um conjunto de métricas específicas para avaliar a legibilidade de testes automatizados, utilizando dados de mineração de repositórios.

5. **Melhorias na Geração de Testes**. Ajuste modelos de linguagem (LLMs) para gerar testes mais legíveis, por meio de *prompt engineering* e *fine-tuning*, avaliando seu impacto em métricas como clareza de código e ausência de *test smells*.

6. **Análise de Co-evolução**. Obtenção de evidências sobre o comportamento da co-evolução entre o código de produção e os testes gerados por LLMs, utilizando dados de ao menos 10 repositórios de código aberto.

7. **Padrões de Evolução e Manutenção**. Documentação de padrões de manutenção específicos para testes gerados por LLMs, incluindo a frequência e o tipo de alterações realizadas.

8. **Redução do Esforço de Manutenção**. Desenvolvimento de ao menos uma estratégia prática que reduza o esforço de manutenção dos testes gerados, com validação por evidências em sistemas reais.

9. **Produção de Benchmarks**. Criação e disponibilização de ao menos dois conjuntos de dados público contendo, dados sobre as diversas formas de geração de testes, 1) com e sem test smells, e 2) com problemas e melhoria de legibilidade, incluindo dados históricos ao longo do tempo.

10. **Produção Científica**. Publicação de no mínimo quatro artigos científicos em conferências e periódicos de alto impacto nas áreas de Engenharia de Software e Inteligência Artificial.

11. **Contribuição para a Comunidade Open Source**. Submissão de ferramentas, benchmarks e guias de melhorias, na forma de projetos de código aberto, promovendo a ciência aberta e o impacto prático do projeto.

6 Metodologia

A metodologia proposta busca abordar de forma estruturada as perguntas de pesquisa e atingir os objetivos gerais e específicos do projeto. Ela está organizada nas etapas a seguir:

6.1 Identificação e Mapeamento de Test Smells

6.1.1 Coleta de Dados

Os sistemas serão selecionados com base em critérios como popularidade, diversidade tecnológica (linguagens de programação e frameworks), e a disponibilidade de histórico em repositórios públicos, como GitHub. A seleção se inspirará em abordagens descritas em estudos anteriores, como Tufano et al. [2016], Bavota et al. [2015], Martins et al. [2024] que analisaram test smells em sistemas de código aberto.

Entretanto, em nosso projeto será necessária uma abordagem metodologicamente diferente pois será necessário obter testes gerados por LLMs. Recentemente nossos estudos em legibilidade de trechos gerados por LLMs nos indicam que os *pull requests* em geral informam que um determinado trecho de código foi gerado por LLMs Silva et al. [2024]. Assim, espera-se minerar adequadamente testes gerados por LLMs em repositórios de software.

Outra alternativa para obtenção de testes gerados por LLMs é usar os diversos modelos para fazer a respectiva geração. Para os sistemas selecionados, serão utilizados diferentes LLMs (ex.: ChatGPT, Llama, Gemini) para gerar testes automatizados. As configurações dos modelos e prompts serão ajustadas para capturar diferentes cenários e estilos de geração.

A maior parte dos sistemas estudados e respectivos detectores de *test smells* foram propostos para a linguagem Java. Em princípio, vamos trabalhar com esta linguagem, mas também pretendemos expandir para Python.

6.1.2 Classificação Manual e Automática

Essa etapa investiga como ferramentas existentes e novas abordagens serão utilizadas e desenvolvidas para identificar e corrigir *smells* nos testes gerados por LLMs e como a detecção e refatoração nestes testes gerados por LLMs se comparam a outros tipos de testes. Estudos como Tufano et al. [2016] e Peruma et al. [2020] destacaram a possibilidade de ferramentas automáticas para detectar *test smells*. Serão empregadas ferramentas tais como: 1) TSDetect (Peruma et al., 2020) e JNose (Virgínio et al. [2021]) para Java; e PyNose para Python. Outras ferramentas listadas por Aljedaani et al. [2021] poderão ser consideradas. Além disso, *linters* especializados como SonarLint, PMD e Checkstyle poderão ser configurados e utilizar para avaliar padrões de qualidade relacionados a *smells* em testes.

A classificação manual será conduzida para validar e complementar os resultados das ferramentas automáticas, identificando falsos positivos e falsos negativos, além de classificar casos ambíguos. Assim, o foco será ajustar e validar essas classificações com base em padrões reconhecidos, como os descritos por van Deursen et al. [2001], Aljedaani et al. [2021] e Panichella et al. [2022].

6.1.3 Análise e Comparações

Esta etapa avaliará a frequência de ocorrência e as co-ocorrências de test smells, explorando como elas impactam métricas de qualidade, como legibilidade, manutenibilidade (propensão a mudanças e defeitos) e confiabilidade dos testes. Dentre as métricas que estudaremos incluem, frequência de mudanças, tamanho das mudanças, frequência de testes que falham sem motivo válido, testes que não detectam falhas reais no código, cobertura de código Panichella et al. [2022], Spadini et al. [2018], Shamshiri et al. [2018].

As co-ocorrências serão analisadas por meio de técnicas de análise estatística e mineração de dados, tais como, regras de associação e/ou clusterização de grafos.

Também pretendemos conduzir uma análise comparativa dos diversos aspectos previamente mencionados entre testes gerados por LLM, testes gerados automaticamente por outras abordagens e testes manuais.

6.2 Avaliação e melhoria da Legibilidade dos Testes

6.2.1 Uso de scores de avaliação de legibilidade

Avaliação de legibilidade de código, de uma maneira geral, é um aspecto ainda aberto, dada a complexidade de se obter uma medição precisa e ter um componente humano bastante

importante, pois diferentes pessoas podem ter diferentes percepções de legibilidade.

Estudos como o de Scalabrino et al. [2019] e Buse and Weimer [2010] propuseram scores de legibilidade. Em particular, Scalabrino et al. [2019] propõe um *score* que se converte em uma resposta binária que define se o código está legível ou não.

Vamos avaliar empiricamente como estes *scores* se comportam para avaliar legibilidade de testes de software e como eles se correlacionam com outras métricas, tal como foi conduzido em Buse and Weimer [2010].

6.2.2 Avaliação com *linters*

Usaremos também outras alternativas para avaliar a legibilidade. Em nosso trabalho, de Carvalho Dantas et al. [2023b], comparamos a legibilidade de *snippets* de código gerado por LLMs com snippets de código equivalentes escritos manualmente, usando como critério, as *issues* detectadas pelo SonarLint. Neste projeto, vamos considerar também outros *linters* como PMD, Checkstyle e Pylint.

A legibilidade dos testes gerados por LLMs será comparada com testes manuais e aqueles gerados por ferramentas tradicionais, considerando métricas previamente definidas.

6.2.3 Mineração de *pull requests*

Vamos explorar técnicas de mineração de repositórios de software para minerar como desenvolvedores melhoram legibilidade de teste automaticamente gerados por LLMs ou manuais. Usaremos uma metodologia similar ao nosso trabalho (de Carvalho Dantas et al. [2023a]), onde analisamos *pull requests* para verificar quais *issues* de legibilidade os desenvolvedores em geral estão preocupados em corrigir, comparando-as com as issues produzidas pelos *linters*.

6.2.4 Recomendação automática de melhoria legibilidade

Com base nos resultados anteriores de avaliação de legibilidade de testes gerados por LLMs, vamos propor métodos automáticos para identificar padrões recorrentes de baixa legibilidade em testes gerados.

Para esta etapa, vamos comparar diferentes abordagens de treinamento de modelos, tais como, classificação de texto e *fine-tuning* de LLMs.

6.3 Análise de Evolução e Manutenção

Esta etapa metodológica servirá para avaliar como os testes gerados por LLMs se comportam ao longo do tempo. Basicamente, a metodologia desta etapa é baseada na mineração de repositórios de software. Vamos organizar esta etapa em duas diferentes perspectivas: a primeira em relação exclusivamente à evolução e manutenção do teste gerado por LLM e a

segunda em relação à co-evolução do código de teste gerado por LLM e do respectivo código testado.

6.3.1 Evolução e manutenção dos testes gerados por LLMs

Inspirado nos estudos de Kim et al. [2021] e Tufano et al. [2016], uma das investigações a ser feita é entender como a densidade de test smells varia ao longo do tempo para testes gerados por LLMs e comparar com testes gerados por outras abordagens e/ou testes escritos manualmente.

Outra investigação que será conduzida é a análise de *pull requests* para entender se a remoção de *test smells* em testes gerados por LLMs segue o mesmo padrão dos outros testes automatizados, isto é, se os testes são removidos com tarefa exclusiva para isto, ou se ocorre como consequência de outras atividades de manutenção.

Além disso, pretendemos avaliar como os *smells* em testes gerados por LLMs estão associados à propensão em defeitos *post-release*, quando comparado com outros tipos de testes. Serão usados métodos estatísticos usuais para mensurar esta possível associação, testando a comparação entre as diferentes amostras.

Além, da avaliação dos testes gerados por LLMs em relação aos *test smells*, vamos avaliar a evolução destes testes em relação aos critérios de legibilidade. Neste caso, vamos analisar as alterações feitos neste tipo de teste procurando filtrar mudanças que foram feitas exclusivamente para melhoria de legibilidade do teste. Vamos utilizar uma metodologia de filtragem similar à que utilizamos em nosso trabalho para avaliação de melhoria de legibilidade de código de produção de Carvalho Dantas et al. [2023a].

6.3.2 Co-evolução e co-manutenção dos testes gerados por LLMs e respectivo código de produção

Para investigar a relação entre a evolução do código de produção e os testes gerados por LLMs, a metodologia será baseada em abordagens já descritas por Zaidman et al. [2011], Tufano et al. [2016] e Spadini et al. [2018]. Essa investigação será expandida para comparar testes gerados por LLMs, testes manuais e testes gerados por outras ferramentas automatizadas. A metodologia incluirá: 1) A análise do histórico de *commits* e *pull requests* para identificar mudanças no código de produção e nos testes associados; 2) A análise de padrões de co-evolução, procurando identificar como a frequência e o tipo de alterações simultâneas nos testes e no código ocorrem; 3) Avaliação do tempo médio entre mudanças no código e atualizações nos testes para comparar a rapidez na adaptação de testes gerados por LLMs em relação a testes manuais e outras formas de automatização; 4) Medição da quantidade de commits e do volume de alterações (linhas adicionadas, removidas ou modificadas) necessárias para ajustar os testes após mudanças no código de produção; 5) Análise qualitativa

de pull requests que envolvem alterações em testes para entender as razões por trás dessas mudanças.

É importante ressaltar que as análises acima devem ser conduzidas também de forma comparativa, procurando identificar diferenças nos padrões de co-evolução e co-manutenção entre os testes gerados por LLMs e testes não gerados por LLMs.

6.4 Melhoria na detecção e Sugestão de Refatoração Automática

6.4.1 Melhoria na detecção de *smells* para testes gerados por LLMs

Esta etapa será baseada nos resultados obtidos por meio da etapa da Seção 6.1, quando a análise manual nos permitirá entender melhor as limitações das abordagens de detecção para testes gerados por LLMs. A partir daí, proporemos alternativas customizadas para mitigar tais limitações. A ideia inicial é aplicar os próprios modelos de linguagem ajustados para suportar tarefas de detecção, usando abordagens como fine-tuning e prompt engineering.

6.4.2 Investigação de refatoração de test smells gerados por LLMs

Estudos recentes sobre refatoração de *test smells* incluem trabalhos como os de Soares et al. [2020], Lambiase et al. [2020], dentre outros.

Entretanto, ainda não foi claramente estudado como os *test smells* gerados por LLMs são refatorados. Nesta etapa vamos usar estratégias similares ao nosso trabalho em de Carvalho Dantas et al. [2023a] para identificar *pull requests* que tratam de refatorações de *test smells* gerados por LLMs. Uma vez que estes smells sejam minerados, iremos aplicar técnicas de análise qualitativa manual como *open coding* para categorizar e quantificar estas refatorações.

6.4.3 Sugestão de refatoração automática

Uma vez que tenhamos investigado como os desenvolvedores efetivamente têm refatorado *test smells* gerados por LLMs, vamos investigar por meio de LLMs, variando diferentes estratégias de *prompt engineering* como os próprios LLMs podem sugerir melhorias para remoção de *smells*.

Neste casos, avaliaremos por meio de submissão de *pull requests* para sistemas de código aberto se as melhorias fazem sentido e podem ser aceitas. Este tipo de avaliação envolve uma questão ética de não sobrecarregar o trabalho das equipes de desenvolvimento *open source* e portanto, requer um uso comedido deste recurso.

6.5 Melhoria geral na geração de testes por LLMs

Diversos fatores podem influenciar a qualidade dos testes gerados por LLMs, tais como a engenharia de *prompts* e o ajuste fino dos modelos.

6.5.1 Análise e Engenharia de Prompts

A estrutura dos prompts desempenha um papel crítico na qualidade dos testes gerados por LLMs. Estudos como os de Ahmed and Devanbu [2022] Naimi et al. [2024] mostram que prompts bem projetados podem melhorar significativamente a saída dos modelos. Nesta etapa vamos verificar como diferentes formatos e níveis de detalhamento nos prompts para determinar padrões mais eficazes para geração de testes com melhores características como legibilidade, cobertura de código e ausência de test smells.

6.5.2 Ajuste Fino (*Fine-Tuning*) de LLMs

Baseando-se em trabalhos como o de Huang et al. [2023], será realizado o ajuste fino dos LLMs para especializá-los na geração de testes automatizados de alta qualidade. As etapas incluem: 1) construção de *datasets* com testes bem projetados, incluindo exemplos manuais e/ou gerados por ferramentas tradicionais; 2) avaliação do impacto do ajuste fino em métricas como eficiência de geração e qualidade dos testes. Esta avaliação será similar à realizada com engenharia de *prompts*

6.6 Divulgação

As soluções desenvolvidas serão disponibilizadas como projetos *open-source* no GitHub, acompanhadas dos pacotes de reprodução dos artigos, bem como com diretrizes práticas para adoção, aderindo aos princípios da Ciência Aberta. Isto, em geral, inclui os *datasets* e ferramentas associadas ao projeto.

Os resultados serão apresentados em conferências e periódicos relevantes para a disseminação dos achados do projeto.

7 Resultados Esperados

Este projeto vai permitir alcançar importantes resultados científicos e tecnológicos. Os resultados esperados estão organizados em três categorias principais, alinhadas aos objetivos e à metodologia proposta: *Test Smells*, Legibilidade e Evolução/Manutenibilidade.

7.1 Resultados Relacionados a Test Smells

7.1.1 Catálogo Atualizado de Test Smells em Testes Gerados por LLMs

Esperamos identificar padrões específicos e frequentes de *test smells* presentes em testes gerados automaticamente por LLMs, considerando diferenças entre linguagens de programação (Java, Python) e ferramentas. Esse catálogo incluirá descrições detalhadas dos *smells* e exemplos ilustrativos.

7.1.2 Impacto de *Test Smells* na Qualidade dos Testes

Pretendemos entregar estudos baseados em evidências (*empirical*) que correlacionem a presença de *test smells* gerados por LLMs com métricas como confiabilidade, cobertura de código e propensão a defeitos no sistema testado.

7.1.3 Evidências e Ferramentas para Detecção e Refatoração

Esperamos entregar um estudo baseado em evidências sobre o estado atual das ferramentas para detecção de *test smells* em testes gerados por LLMs e aprimorar estas ferramentas baseadas no conhecimento obtido. Além disso, pretendemos entregar evidências sobre como os desenvolvedores vêm refatorando testes gerados por LLMs e a viabilidade de sugestão e/ou refatoração automáticas deste subconjunto de *testes*. As ferramentas serão validadas em projetos de código aberto e disponibilizadas como software livre.

7.1.4 Comparação de *Test Smells* em Diferentes Tipos de Testes

Esperamos entregar uma análise das diferenças na frequência e no impacto dos *test smells* em testes gerados por LLMs, testes manuais e testes gerados por ferramentas tradicionais.

7.2 Resultados Relacionados à Legibilidade

7.2.1 Métricas e Avaliação de Legibilidade

Esperamos entregar uma análise de como métricas e ferramentas para avaliar a legibilidade de testes gerados por LLMs estão de fato relacionadas a problemas nos mesmos, considerando fatores como clareza de assertivas, complexidade do código, uso de comentários e nomenclatura.

7.2.2 Diretrizes Práticas para Melhorar a Legibilidade

Esperamos produzir um guia de recomendações baseadas em evidências sobre como melhorar a legibilidade de testes gerados por LLMs, abrangendo boas práticas para geração de *prompts* e possível refatoração dos testes.

7.2.3 Modelos de LLMs Ajustados para Legibilidade

Esperamos entregar evidências que mostrem como melhorar a geração de testes legíveis e bem organizados, usando técnicas de *prompt engineering* e potencialmente ajuste fino (*fine-tuning*).

7.2.4 Estudo da Percepção de Desenvolvedores

Esperamos entregar evidências que demonstrem como desenvolvedores percebem a legibilidade dos testes e como isto pode afetar sua eficácia e facilidade de uso sob a ótica deles.

7.3 Resultados Relacionados à Evolução e Manutenibilidade

7.3.1 Análise de Co-evolução e Co-manutenção

Pretendemos entregar um estudo baseado em mineração de dados sobre como os testes gerados por LLMs evoluem junto com o código de produção, incluindo métricas como frequência de alterações e alinhamento temporal entre mudanças no código e nos testes, e principalmente que medidas podem ser tomadas para tornar o uso de testes gerados por LLMs mais efetivo.

7.3.2 Impacto de Test Smells na Manutenibilidade

Pretendemos mostrar por meio de avaliação quantitativa e qualitativa se (e como) a presença de *test smells* em testes gerados por LLMs afeta o esforço de manutenção, a frequência de falhas e a propensão a defeitos nos testes.

7.3.3 Padrões de Evolução de Testes Gerados por LLMs

Pretendemos mostrar como a densidade de *test smells* e a qualidade dos testes evoluem ao longo do tempo, comparando diferentes abordagens de geração de testes (manual, tradicional e LLMs).

7.3.4 Redução do Esforço de Manutenção

A partir dos resultados anteriores, esperamos indicar uma melhor prática para controlar, se necessário, a qualidade ao longo do processo de geração e manutenção dos testes. Mais

especificamente, pretendemos propor estratégias e ferramentas que auxiliem na manutenção de testes gerados, minimizando a carga de trabalho das equipes de desenvolvimento.

7.4 Impactos Transversais

7.4.1 Produção de *Benchmarks*

Pretendemos disponibilizar *datasets* públicos contendo exemplos de testes gerados por LLMs com e sem test smells, bem como sobre problemas de legibilidade, facilitando a replicação de estudos e o desenvolvimento de novas ferramentas.

7.4.2 Contribuição para a Comunidade *Open Source*

Pretendemos construir e disponibilizar ferramentas e recomendações para projetos de código aberto, promovendo o uso eficiente de soluções baseadas em LLMs.

7.4.3 Publicações Científicas e Disseminação de Conhecimento

Esperamos publicar resultados em conferências e periódicos relevantes, contribuindo para o avanço do estado da arte em geração e manutenção de testes automatizados.

Referências

T. Ahmed and P. Devanbu. Few-shot training LLMs for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–5, 2022.

A. Akli, G. Haben, S. Habchi, M. Papadakis, and Y. L. Traon. Flakycat: Predicting flaky tests categories using few-shot learning. In *2023 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 140–151, 2023. doi: 10.1109/AST58925.2023.00018.

W. Aljedaani, A. Peruma, A. Aljohani, M. Alotaibi, M. W. Mkaouer, A. Ouni, C. D. Newman, A. Ghallab, and S. Ludi. Test smell detection tools: A systematic mapping study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, EASE '21, page 170–180, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390538. doi: 10.1145/3463274.3463335. URL https://doi.org/10.1145/3463274.3463335.

N. Alshahwan, J. Chheda, A. Finogenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang. Automated unit test improvement using large language models at Meta. In *Companion Proceedings of the 32nd ACM International Confe-*

rence on the Foundations of Software Engineering, pages 185–196. Association for Computing Machinery, 2024. ISBN 9798400706585. doi: 10.1145/3663529.3663839. URL https://doi.org/10.1145/3663529.3663839.

G. Bavota, A. Qusef, R. Oliveto, A. Lucia, and D. Binkley. Are test smells really harmful? an empirical study. *Empirical Softw. Engg.*, 20(4):1052–1094, Aug. 2015. ISSN 1382-3256. doi: 10.1007/s10664-014-9313-0. URL https://doi.org/10.1007/s10664-014-9313-0.

S. Bhatia, T. Gandhi, D. Kumar, and P. Jalote. Unit test generation using generative AI: A comparative performance analysis of autogeneration tools. In *Proceedings of the 1st International Workshop on Large Language Models for Code*, pages 54–61. Association for Computing Machinery, 2024. ISBN 9798400705793. doi: 10.1145/3643795.3648396. URL https://doi.org/10.1145/3643795.3648396.

R. P. Buse and W. R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010. doi: 10.1109/TSE.2009.70.

Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin. Chatunitest: A framework for LLM-based test generation. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pages 572–576. Association for Computing Machinery, 2024. ISBN 9798400706585. doi: 10.1145/3663529.3663801. URL https://doi.org/10.1145/3663529.3663801.

C. E. de Carvalho Dantas, A. M. Rocha, and M. de Almeida Maia. How do developers improve code readability? an empirical study of pull requests. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2023, Bogotá, Colombia, October 1-6, 2023*, pages 110–122. IEEE, 2023a. doi: 10.1109/ICSME58846.2023.00022. URL https://doi.org/10.1109/ICSME58846.2023.00022.

C. E. de Carvalho Dantas, A. M. Rocha, and M. de Almeida Maia. Assessing the readability of ChatGPT code snippet recommendations: A comparative study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering, SBES 2023, Campo Grande, Brazil, September 25-29, 2023*, pages 283–292. ACM, 2023b. doi: 10.1145/3613372.3613413. URL https://doi.org/10.1145/3613372.3613413.

V. Guilherme and A. Vincenzi. An initial investigation of ChatGPT unit test generation capability. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing*, pages 15–24. Association for Computing Machinery, 2023. ISBN 9798400716294. doi: 10.1145/3624032.3624035. URL https://doi.org/10.1145/3624032.3624035.

K. Huang, X. Meng, J. Zhang, Y. Liu, W. Wang, S. Li, and Y. Zhang. An empirical study on fine-tuning large language models of code for automated program repair. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1162–1174, 2023. doi: 10.1109/ASE56229.2023.00181.

Z. Jiang, M. Wen, J. Cao, X. Shi, and H. Jin. Towards understanding the effectiveness of large language models on directed test input generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 1408–1420. Association for Computing Machinery, 2024. ISBN 9798400712487. doi: 10.1145/3691620.3695513. URL `https://doi.org/10.1145/3691620.3695513`.

D. J. Kim, T. P. Chen, and J. Yang. The secret life of test smells - an empirical study on test smell evolution and maintenance. *Empir. Softw. Eng.*, 26(5):100, 2021. doi: 10.1007/S10664-021-09969-1. URL `https://doi.org/10.1007/s10664-021-09969-1`.

S. Lambiase, A. Cupito, F. Pecorelli, A. De Lucia, and F. Palomba. Just-in-time test smell detection and refactoring: The darts project. In *2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC)*, pages 441–445, 2020. doi: 10.1145/3387904.3389296.

Q. Le Dilavrec, D. E. Khelladi, A. Blouin, and J.-M. Jézéquel. Untangling spaghetti of evolutions in software histories to identify code and test co-evolutions. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 206–216. IEEE, 2021.

C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *Proceedings of the 45th International Conference on Software Engineering*, pages 919–931. IEEE Press, 2023. ISBN 9781665457019. doi: 10.1109/ICSE48619.2023.00085. URL `https://doi.org/10.1109/ICSE48619.2023.00085`.

M. Leotta, H. Z. Yousaf, F. Ricca, and B. Garcia. AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, pages 339–344. Association for Computing Machinery, 2024. ISBN 9798400717017. doi: 10.1145/3661167.3661192. URL `https://doi.org/10.1145/3661167.3661192`.

Z. Liu, C. Chen, J. Wang, X. Che, Y. Huang, J. Hu, and Q. Wang. Fill in the blank: Context-aware automated text input generation for mobile GUI testing. In *Proceedings of the 45th International Conference on Software Engineering*, pages 1355–1367. IEEE Press, 2023. ISBN 9781665457019. doi: 10.1109/ICSE48619.2023.00119. URL `https://doi.org/10.1109/ICSE48619.2023.00119`.

L. A. Martins, T. A. Ghaleb, H. A. X. Costa, and I. Machado. A comprehensive catalog of refactoring strategies to handle test smells in Java-based systems. *Softw. Qual. J.*, 32 (2):641–679, 2024. doi: 10.1007/S11219-024-09663-7. URL https://doi.org/10.1007/s11219-024-09663-7.

L. Naimi, E. M. Bouziane, M. Manaouch, and A. Jakimi. A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering. In *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, pages 1–5, 2024. doi: 10.1109/ICCSC62074.2024.10616548.

C. Nguyen, H. Bui, V. Nguyen, and T. Nguyen. An approach to generating API test scripts using GPT. In *Proceedings of the 12th International Symposium on Information and Communication Technology*, pages 501–509. Association for Computing Machinery, 2023. ISBN 9798400708916. doi: 10.1145/3628797.3628947. URL https://doi.org/10.1145/3628797.3628947.

A. Panichella, S. Panichella, G. Fraser, A. A. Sawant, and V. J. Hellendoorn. Test smells 20 years later: detectability, validity, and reliability. *Empirical Softw. Engg.*, 27(7), Dec. 2022. ISSN 1382-3256. doi: 10.1007/s10664-022-10207-5. URL https://doi.org/10.1007/s10664-022-10207-5.

A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba. tsDetect: an open source test smells detection tool. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 1650–1654, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370431. doi: 10.1145/3368089.3417921. URL https://doi.org/10.1145/3368089.3417921.

Z. Rasool, S. Barnett, D. Willie, S. Kurniawan, S. Balugo, S. Thudumu, and M. Abdelrazek. LLMs for test input generation for semantic applications. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, pages 160–165. Association for Computing Machinery, 2024. ISBN 9798400705915. doi: 10.1145/3644815.3644948. URL https://doi.org/10.1145/3644815.3644948.

D. Roy, Z. Zhang, M. Ma, V. Arnaoudova, A. Panichella, S. Panichella, D. Gonzalez, and M. Mirakhorli. Deeptc-enhancer: improving the readability of automatically generated tests. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 287–298. Association for Computing Machinery, 2021. ISBN 9781450367684. doi: 10.1145/3324884.3416622. URL https://doi.org/10.1145/3324884.3416622.

S. Scalabrino, G. Bavota, C. Vendome, M. Linares Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically assessing code understandability. *Transactions on Software Engineering (TSE)*, 47(3):595 – 613, 2019. doi: 10.1109/TSE.2019.2901468.

S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser. How do automatically generated unit tests influence software maintenance? In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 250–261, 2018. doi: 10.1109/ICST.2018.00033.

J. Shin, S. Hashtroudi, H. Hemmati, and S. Wang. Domain adaptation for code model-based unit test case generation. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1211–1222. Association for Computing Machinery, 2024. ISBN 9798400706127. doi: 10.1145/3650212.3680354. URL https://doi.org/10.1145/3650212.3680354.

J. R. Silva, C. E. de Carvalho Dantas, and M. de Almeida Maia. What developers ask to ChatGPT in github pull requests? an exploratory study. In A. G. Uchôa and L. Rocha, editors, *Proceedings of the 12th Workshop on Software Visualization, Evolution and Maintenance, VEM 2024, Curitiba, Brazil, September 30, 2024*, pages 125–136. Sociedade Brasileira de Computação, 2024. doi: 10.5753/VEM.2024.3913. URL https://doi.org/10.5753/vem.2024.3913.

E. Soares, M. Ribeiro, G. Amaral, R. Gheyi, L. Fernandes, A. Garcia, B. Fonseca, and A. Santos. Refactoring test smells: A perspective from open-source developers. In *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*, SAST '20, page 50–59, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450387552. doi: 10.1145/3425174.3425212. URL https://doi.org/10.1145/3425174.3425212.

D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli. On the relation of test smells to software code quality. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–12, 2018. doi: 10.1109/ICSME.2018.00010.

M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. An empirical investigation into the nature of test smells. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 4–15, 2016.

A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In M. Marchesi and G. Succi, editors, *Proceedings 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001)*, may 2001.

T. Virgínio, L. Martins, R. Santana, A. Cruz, L. Rocha, H. Costa, and I. Machado. On the test smells detection: an empirical study on the JNose test accuracy. *Journal of Software Engineering Research and Development*, 9(1):8:1 – 8:14, Sep. 2021. doi: 10.5753/jserd.2021.1893. URL https://journals-sol.sbc.org.br/index.php/jserd/article/view/1893.

Z. Xue, L. Li, S. Tian, X. Chen, P. Li, L. Chen, T. Jiang, and M. Zhang. Domain knowledge is all you need: A field deployment of LLM-powered test case generation in fintech domain. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pages 314–315. Association for Computing Machinery, 2024a. ISBN 9798400705021. doi: 10.1145/3639478.3643087. URL https://doi.org/10.1145/3639478.3643087.

Z. Xue, L. Li, S. Tian, X. Chen, P. Li, L. Chen, T. Jiang, and M. Zhang. LLM4fin: Fully automating LLM-powered test case generation for fintech software acceptance testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1643–1655. Association for Computing Machinery, 2024b. ISBN 9798400706127. doi: 10.1145/3650212.3680388. URL https://doi.org/10.1145/3650212.3680388.

A. Zaidman, B. V. Rompaey, A. van Deursen, and S. Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empir. Softw. Eng.*, 16(3):325–364, 2011. doi: 10.1007/S10664-010-9143-7. URL https://doi.org/10.1007/s10664-010-9143-7.