REVISTA OWL (*OWL Journal*)
www.revistaowl.com.br – ISSN: 2965-2634

KARL POPPER, ENGENHARIA DE SOFTWARE E BIBLIOMETRIA: UM MAPEAMENTO DA FALSEABILIDADE EM MÉTODOS ÁGEIS E VALIDAÇÃO DE SISTEMAS

KARL POPPER, SOFTWARE ENGINEERING, AND BIBLIOMETRICS: A MAPPING OF FALSIFIABILITY IN AGILE METHODS AND SYSTEM VALIDATION

DOI: 10.5281/zenodo.21864000

Jonas Ernesto Poli[1]
Luís Fernando Soares Zuin[2]
Avaetê de Lunetta e Rodrigues Guerra[3]

RESUMO

A percepção social de que o software é produto de lógica exata contrasta com a realidade, pois sistemas são falíveis. Por isso, precisam ser mantidos com ciclos contínuos de teste, correção e refatoração. À luz do racionalismo crítico de Karl Popper, práticas de desenvolvimento podem ser vistas como processos de conjecturas e refutações, em que implementações são continuamente expostas a tentativas de falsificação. Este artigo apresenta um estudo bibliométrico descritivo que mapeia como a literatura acadêmica tem articulado Popper com engenharia de software, com foco em métodos ágeis, validação de requisitos e fundamentos de software testing. Utilizando uma busca estruturada na base Lens.org, foram identificados 22 trabalhos, dos quais 17 foram classificados como centrais por vincularem explicitamente conceitos popperianos a problemas de engenharia de software. Esses registros foram analisados quanto à distribuição temporal, tipologia documental, campos de estudo e temas técnicos, com apoio de tabelas dinâmicas e gráficos. Os resultados mostram concentração em três eixos principais: requisitos e planejamento de releases, práticas ágeis como *Test-Driven Development* (TDD) e *Extreme Programming*, e discussões epistemológicas sobre a justificação de hipóteses em testes de software. Conclui-se que a falseabilidade popperiana fornece um enquadramento conceitual robusto para compreender o software como conjectura tecnicamente controlada, reforçando a importância de práticas que promovem falhas rápidas, testes sistemáticos e revisões contínuas.

1 Mestrando em Ciência, Tecnologia e Sociedade, Universidade Federal de São Carlos, UFSCar, Brasil.
2 Doutor em Engenharia de Produção, Universidade Federal de São Carlos, UFSCar, Brasil.
3 Doutorando em Ciência, Tecnologia e Sociedade, Universidade Federal de São Carlos, UFSCar, Brasil.

Palavras-chave: Bibliometria; Karl Popper; Engenharia de software; Test-Driven Development.

ABSTRACT

Software is often perceived as the result of exact and infallible logic, yet real-world systems behave as fallible artefacts whose validity remains provisional and open to failure. Drawing on Karl Popper's critical rationalism, this article interprets software development practices as sequences of conjectures and refutations, in which implementations are constantly exposed to tests designed to falsify them. A descriptive bibliometric study was conducted to map how academic literature explicitly connects Popper's philosophy with software engineering, agile methods and software validation, based on a structured search in Lens.org that identified 22 publications, 17 of which were classified as central to this intersection. The results show that Popperian concepts such as falsifiability are mainly mobilized in three technical areas—requirements and release planning, agile practices such as TDD and Extreme Programming, and epistemological discussions on software testing—supporting the view of software as a technically controlled but always provisional conjecture.

Keywords: Bibliometrics; Karl Popper; Software Engineering; Test-Driven Development.

1. INTRODUÇÃO

A infraestrutura digital contemporânea depende criticamente de sistemas de software que, no imaginário social, são frequentemente tratados como instâncias de lógica correta e definitiva (Sommerville, 2019). No entanto, relatos recorrentes de falhas em sistemas bancários, plataformas de comércio eletrônico e aplicações de inteligência artificial revelam o óbvio: tais sistemas operam, na prática, como artefatos falíveis, sujeitos a defeitos que só se tornam visíveis em uso real (Sommerville, 2019). Essa discrepância entre a imagem de infalibilidade e a experiência de falhas aponta para um problema epistemológico: em que sentido é possível dizer que um software foi "validado"?

Karl Popper propõe que teorias científicas não podem ser **justificadas** por acumulação indutiva de observações favoráveis, mas apenas **corroboradas** enquanto resistem a tentativas rigorosas de **refutação** (Popper, 2023). Transferida para o contexto técnico, essa perspectiva sugere que o valor epistêmico de um sistema de software não reside na quantidade de testes que ele passou, e sim na intensidade e variedade de testes que tentaram fazê-lo falhar (Popper, 2023; Sommerville, 2019). Práticas contemporâneas de desenvolvimento ágil, como ***Test-Driven Development*** (TDD) e a cultura ***Fail-Fast,*** apontam nessa direção ao estruturar o

trabalho como sequência de experimentos rápidos e reversíveis sobre o código (Shore; Warden, 2022; Sommerville, 2019).

A literatura que articula explicitamente Popper, engenharia de software e métodos ágeis ainda é relativamente dispersa, distribuída entre artigos filosóficos, estudos de caso e relatos de experiência. Este artigo busca mapear esse território por meio de um estudo bibliométrico descritivo, identificando quais problemas de engenharia são tratados com referência direta à falseabilidade popperiana, quais práticas técnicas aparecem associadas a esse enquadramento e como essa discussão evoluiu ao longo do tempo. A pergunta central é: como a literatura científica tem apropriado o racionalismo crítico de Popper para enfrentar desafios de validação, teste e projeto de processos na engenharia de software?

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Racionalismo crítico e falseabilidade

Popper critica o verificacionismo ao mostrar que nenhuma quantidade de observações concordantes é capaz de demonstrar a verdade de uma hipótese universal, enquanto um único contraexemplo é suficiente para refutá-la (Popper, 2023). Em lugar do acúmulo indutivo, propõe um método de conjecturas e refutações: diante de um problema, formulam-se soluções teóricas ousadas, deduzem-se consequências testáveis e buscam-se ativamente situações em que essas consequências falhem (Popper, 2023). Teorias que resistem a testes severos são ditas **corroboradas**, mas nunca **verificadas**; a qualquer momento podem ser substituídas por conjecturas mais informativas que expliquem também os antigos falsos-negativos (Popper, 2023). Essa concepção transforma o erro em elemento estrutural do progresso do conhecimento. Em vez de ser visto como falha moral ou técnica, o erro torna-se indicador de que uma hipótese foi exposta a um teste suficientemente forte para revelar suas limitações (Popper, 2023). A engenharia de software, se adotasse essa postura de maneira explícita, tenderia a organizar seus processos em torno da descoberta sistemática de defeitos, e não em

torno da busca por confirmações que reforçam uma falsa sensação de segurança (Sommerville, 2019).

2.2 Software como artefato empírico e caixa-preta

Sommerville (2019) descreve sistemas de software como artefatos complexos que emergem da interação entre requisitos incompletos, decisões de projeto e ambientes operacionais dinâmicos; por isso, sustentam uma incerteza residual irreduzível mesmo após extensa verificação. Em sistemas grandes, a combinatória de estados possíveis torna inviável qualquer evidência forte de correção global a partir de testes finitos, o que aproxima a prática de desenvolvimento de um processo empírico no qual se procura, na melhor das hipóteses, mostrar que erros conhecidos foram removidos. (Sommerville, 2019).

Latour (2000), ao seguir cientistas e engenheiros “sociedade afora”, mostra como artefatos técnicos se tornam “**caixas-pretas**” quando seus processos internos deixam de ser problematizados, restando apenas entradas e saídas aparentemente estáveis. Sistemas digitais em produção frequentemente ocupam esse lugar: uma vez aceitos, são tomados como dados naturais, e o caráter conjectural de suas implementações desaparece sob camadas de infraestrutura e rotinas de uso (Latour, 2000). A reabertura crítica dessas caixas-pretas, por meio de auditorias, testes, simulações e refatorações, é condição para que o erro volte a ser tratado como motor de aprendizado, e não como mero ruído operacional (Latour, 2000; Sommerville, 2019).

2.3 Métodos ágeis, TDD e Fail-Fast

Métodos ágeis são formas de organizar o desenvolvimento de software em ciclos curtos, com entregas frequentes e ajustes constantes a partir do retorno de usuários e de testes (Shore; Warden, 2022). Essa abordagem trata cada versão do sistema como um experimento: testa-se uma solução rapidamente, observa-se o que funciona ou não e, em seguida, faz-se uma nova tentativa, aproximando o desenvolvimento da ideia popperiana de que o conhecimento avança por conjecturas e refutações (Popper, 2023).

Dentro desse universo, o desenvolvimento orientado a testes (***Test-Driven Development***, ou TDD) segue uma regra simples: primeiro cria-se um teste automático que o programa ainda não consegue passar e só depois se escreve o código necessário para fazer esse teste funcionar (Shore; Warden, 2022). Em outras palavras, define-se explicitamente em que condições o novo trecho de código será considerado errado, e o teste funciona como uma tentativa organizada de refutar essa "conjectura" de solução (Popper, 2023).

O TDD costuma ser descrito pelo ciclo ***Red-Green-Refactor***. "Red" é o momento em que o teste falha, mostrando que a hipótese ainda não funciona; "Green" é quando se escreve o mínimo de código para fazer o teste passar; "Refactor" é a etapa de melhoria estrutural do código, mantendo todos os testes como uma coleção de possíveis refutações futuras (Shore; Warden, 2022). Esse ciclo encena, em pequena escala, o método popperiano: primeiro formula-se um teste exigente, depois propõe-se uma solução que tenta sobreviver a esse teste e, por fim, preservam-se as condições de crítica para que novas falhas possam ser descobertas (Popper, 2023).

O princípio ***Fail-Fast*** ("falhar rápido") reforça essa lógica ao recomendar que sistemas e processos sejam desenhados para deixar os erros aparecerem o quanto antes, de forma clara e limitada, em vez de escondê-los (Shore; Warden, 2022). Em vez de manter o software como uma "caixa-preta" opaca, busca-se criar mecanismos que exponham rapidamente comportamentos indesejados, permitindo corrigi-los antes que causem danos maiores (Latour, 2000). Do ponto de vista popperiano, essa cultura é valiosa porque aumenta as oportunidades de refutar hipóteses sobre o funcionamento do sistema e, assim, impulsiona ciclos curtos e sucessivos de correção e aprendizado (Popper, 2023).

3. METODOLOGIA BIBLIOMÉTRICA

3.1 Desenho do estudo

A bibliometria busca aplicar técnicas quantitativas à análise da produção científica, revelando padrões de crescimento, colaboração e circulação do conhecimento (Araújo, 2006). Em vez de tratar referências apenas como apoio argumentativo, a bibliometria toma os

próprios documentos e seus metadados como objeto de estudo, permitindo mapear campos de pesquisa e identificar focos temáticos com base em contagens, distribuições e redes de citação (Araújo, 2006). Neste trabalho, a bibliometria é utilizada de modo descritivo, para caracterizar como a literatura conecta Popper e engenharia de software em um corpus restrito, sem pretender extrapolações estatísticas para toda a produção internacional.

3.2 Fonte de dados e estratégia de busca

A busca foi realizada na base **Lens.org** a partir de uma *string* estruturada que combina termos popperianos com descritores de engenharia de software. No eixo filosófico, foram incluídos os termos "**Karl Popper**", "**Popperian**", "**falsifiability**", "**falsification**", "**critical rationalism**" e "**conjectures and refutations**". No eixo técnico, adotaram-se "**software engineering**", "**software development**", "**software testing**", "**Test-Driven Development**", "**TDD**", "**agile**", "**Extreme Programming**", "**software quality**", "**verification**" e "**validation**". Os termos de cada eixo foram combinados com o operador **OR**, e os dois eixos foram cruzados com **AND**, aplicando-se a mesma lógica aos campos de título, resumo e palavras-chave. A forma geral da query utilizada na Lens.org pode ser representada assim:

```
(
 title:("Karl Popper" OR "Popperian" OR "falsifiability" OR "falsification"
      OR "critical rationalism" OR "conjectures and refutations")
 AND
 title:("software engineering" OR "software development" OR "software testing"
      OR "Test-Driven Development" OR "TDD" OR "agile" OR "Extreme Programming"
      OR "software quality" OR "verification" OR "validation")
)
OR
(
 abstract:("Karl Popper" OR "Popperian" OR "falsifiability" OR "falsification"
       OR "critical rationalism" OR "conjectures and refutations")
 AND
 abstract:("software engineering" OR "software development" OR "software testing"
       OR "Test-Driven Development" OR "TDD" OR "agile" OR "Extreme Programming"
       OR "software quality" OR "verification" OR "validation")
)
OR
(
 keyword:("Karl Popper" OR "Popperian" OR "falsifiability" OR "falsification"
       OR "critical rationalism" OR "conjectures and refutations")
 AND
```

```
keyword:("software engineering" OR "software development" OR "software testing"
        OR "Test-Driven Development" OR "TDD" OR "agile" OR "Extreme Programming"
        OR "software quality" OR "verification" OR "validation")
)
```

A seleção dos descritores filosóficos seguiu diretamente a terminologia central utilizada por Popper (2023) ao formular o critério de falseabilidade e o método de conjecturas e refutações em A Lógica da Pesquisa Científica. Os termos técnicos de engenharia de software e métodos ágeis foram inspirados nos capítulos de processos, requisitos e qualidade apresentados por Sommerville (2019) e nas práticas de desenvolvimento ágil sistematizadas por Shore e Warden (2022), em especial TDD, integração contínua e *Extreme Programming*.

3.3 Tratamento dos dados e critérios de classificação

A partir do arquivo exportado, gerado pelo lens.org, foram adicionadas três novas colunas analíticas: **Classificação**, **Justificativa** e **Tema Técnico**. A coluna **Classificação** recebeu valor 1 quando o título e o resumo indicavam articulação explícita entre conceitos popperianos (falseabilidade, racionalismo crítico, conjecturas e refutações) e problemas de engenharia de software, como teste, validação, requisitos, métodos ágeis ou modelagem de sistemas. Por outro lado, registros que utilizavam Popper em outras áreas (ecologia, epidemiologia, economia) foram marcados com 0. Ao final, 17 dos 22 registros foram classificados como centrais para o recorte deste estudo, compondo o corpus principal de análise.

A coluna **Tema Técnico** agrupou os trabalhos centrais em três eixos: (a) *Requisitos/Release Planning*, reunindo textos sobre planejamento incremental, decisão sob incerteza e validação de requisitos; (b) *Agile/TDD/XP*, cobrindo trabalhos que relacionam explicitamente Popper a métodos ágeis e práticas de teste orientado por testes; e (c) *Fundamentos de Engenharia de Software*, incluindo artigos de filosofia da tecnologia que discutem justificação de hipóteses em software testing e epistemologia de modelos de software. Além disso, a coluna *Fields of Study*, que foi gerada pelo lens.org apresentando diversos campos de estudo de cada um dos trabalhos analisados, foi “explodida” (em outra

planilha) em termos individuais para permitir contagem de frequência das áreas associadas a cada publicação, evidenciando a intensidade da associação com engenharia de software e áreas afins.

4. RESULTADOS

4.1 Distribuição temporal das publicações

A análise temporal dos 17 trabalhos centrais revela que o cruzamento explícito entre Popper e engenharia de software se distribui entre 1999 e 2023, com concentração em meados da década de 2000 e retomada recente na década de 2020. Em 2006 e 2007 observam-se três publicações em cada ano, enquanto 2022 apresenta dois registros, e os demais anos registram um trabalho isolado. Esse padrão sugere que a discussão ganha visibilidade inicial no contexto da consolidação dos métodos ágeis, quando a crítica a abordagens pesadas de planejamento abre espaço para leituras filosóficas sobre experimentação e erro na construção de software.

Figura 1 - Distribuição anual das publicações centrais (1999–2023)

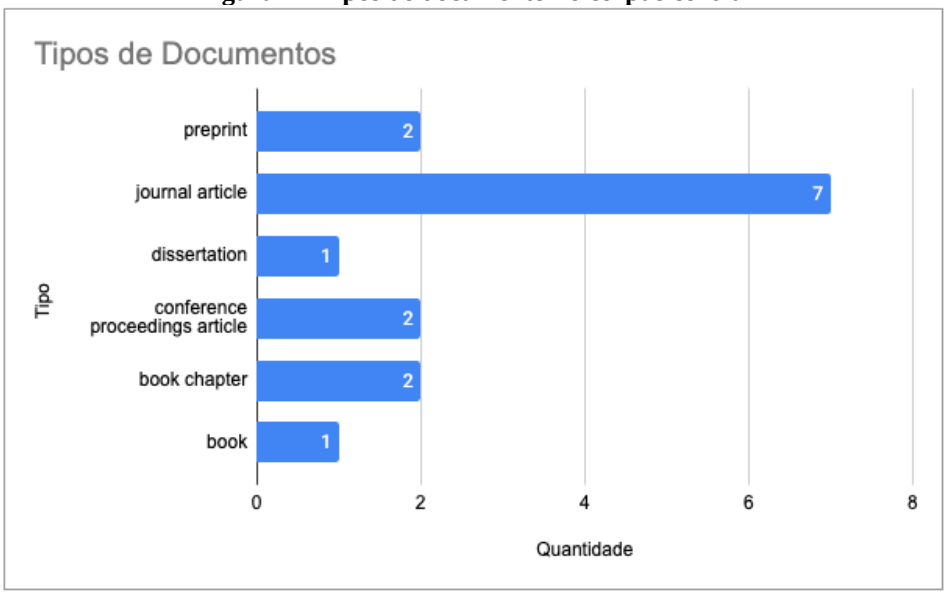
Fonte: Do próprio autor

A Figura 1 apresenta graficamente essa evolução, destacando os pequenos picos em 2006 e 2007 e a retomada em 2022. Embora o corpus seja reduzido, o gráfico indica que o interesse por Popper na engenharia de software não é apenas um fenômeno recente associado à inteligência artificial, mas acompanha a difusão de abordagens iterativas e incrementais ao longo de mais de duas décadas.

4.2 Tipologia documental

Quanto à tipologia, o corpus central inclui sete artigos de periódico, dois artigos em anais de conferência, dois capítulos de livro, um livro, uma dissertação e dois *preprints*, totalizando 15 documentos com tipologia claramente identificada. Os artigos de periódico predominam, sinalizando que parte significativa da discussão se consolidou em veículos com revisão por pares, em diálogo com comunidades estabelecidas de engenharia de software e de filosofia da ciência. A presença de capítulos de livro, livro completo e dissertação, por sua vez, indica que a interlocução entre Popper e engenharia de software também é explorada em obras de maior fôlego e em contextos formativos.

Figura 2 - Tipos de documento no corpus central

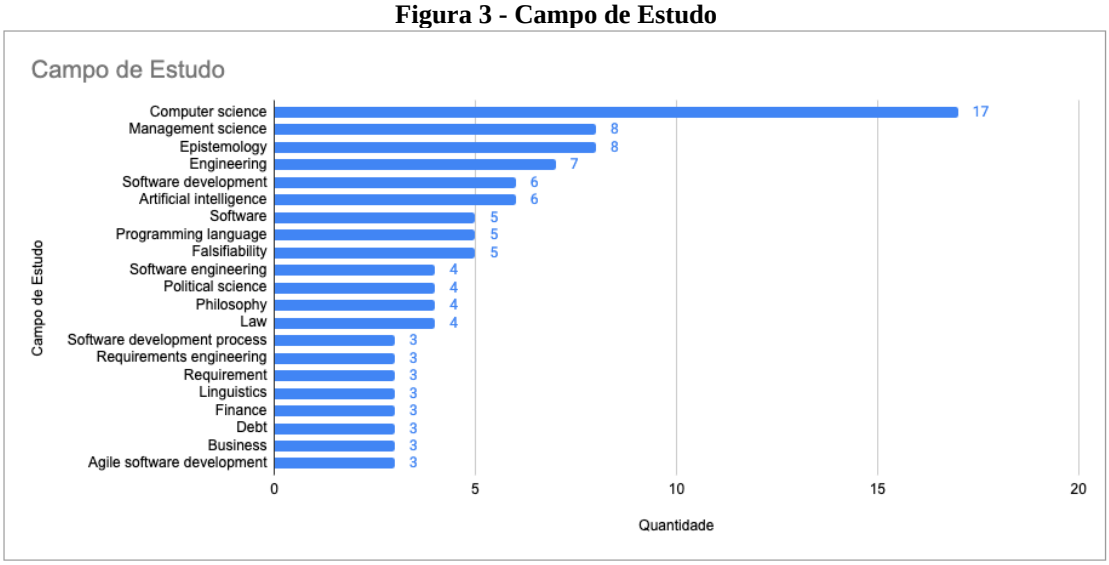


Fonte: Do próprio autor

A Figura 2 explicita essa distribuição, evidenciando a centralidade dos *journal articles* e a pluralidade de formatos em que o tema aparece. Essa pluralidade reforça a hipótese de que o debate não é restrito a um nicho disciplinar, mas atravessa espaços de publicação voltados a sistemas de informação, filosofia da tecnologia e engenharia de software.

4.3 Campos de estudo e termos principais

Figura 3 - Campo de Estudo



Fonte: Do próprio autor

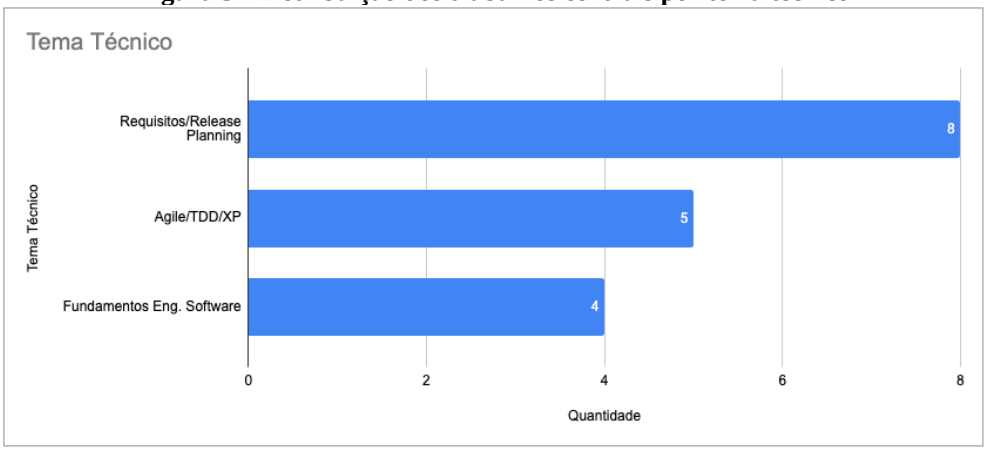
Vemos na figura 3 que a exploração dos campos de estudo mostra que *Computer science* é o rótulo mais frequente, seguido por termos associados a *Management science*, *Epistemology* e *Engineering*, o que indica uma forte interseção entre áreas técnicas e filosóficas. Termos diretamente ligados ao recorte deste estudo, como *Software engineering*, *Software development*, *Requirements engineering* e *Falsifiability*, aparecem com recorrência significativa, sugerindo que a relação entre Popper e engenharia de software é construída tanto a partir de vocabulário técnico quanto de conceitos filosóficos centrais.

Para sintetizar os campos mais diretamente relacionados a software, os termos de *Fields of Study* foram agrupados em três categorias principais: *Software engineering* (13 ocorrências), *Software development* (3 ocorrências) e *Artificial intelligence* (1 ocorrência),

considerando apenas os 17 trabalhos centrais. A Figura 4 mostra que a grande maioria das publicações ancora-se explicitamente em engenharia de software, enquanto menções a inteligência artificial aparecem de forma pontual. Esse resultado sugere que, embora o tema da validação de sistemas de IA seja relevante, o uso de Popper identificado no corpus concentra-se sobretudo em problemas clássicos de construção e garantia de qualidade de software.

4.4 Temas técnicos

Figura 5 - Distribuição dos trabalhos centrais por tema técnico



Fonte: Do próprio autor

A codificação temática dos 17 trabalhos centrais revelou três eixos principais. No eixo ***Requisitos/Release Planning***, 8 publicações tratam de como planejar, em etapas, quais funcionalidades entram em cada versão de um sistema, levando em conta prioridades de negócio, incertezas e riscos associados. Esses trabalhos propõem modelos econômicos e probabilísticos que estimam custos e benefícios esperados de diferentes decisões de requisitos, tratando esses modelos como hipóteses que podem ser rejeitadas à luz de dados de uso e resultados de simulações. Com isso, o planejamento de releases deixa de ser um plano fixo e passa a funcionar como uma sequência de conjecturas sujeitas a teste e possível

refutação, em linha com o ideal popperiano de experimentar soluções sob condições controladas.

O eixo ***Agile/TDD/XP*** reúne 5 publicações que aproximam a filosofia do racionalismo crítico de Popper das práticas de desenvolvimento ágil. Nesses trabalhos, técnicas como desenvolvimento orientado a testes (TDD), integração contínua e ciclos curtos de entrega são entendidas como formas de organizar o trabalho de modo que o código esteja sempre exposto a tentativas rápidas de encontrar erros, em sintonia com o princípio *Fail-Fast* e com a ideia de que o conhecimento avança ao tornar o erro visível e discutível.

Já o eixo **Fundamentos de Engenharia de Software** inclui 4 artigos de filosofia da tecnologia e de testes de software que examinam, em maior detalhe, o que exatamente os testes permitem afirmar sobre um sistema. Esses textos discutem como interpretar as evidências geradas por casos de teste e até que ponto modelos formais representam adequadamente sistemas complexos, argumentando, em diálogo com Popper, que testes não "provam" a correção de um programa, mas apenas podem refutar hipóteses ou reforçar sua corroboração em situações bem delimitadas.

A Figura 5 sintetiza visualmente essas proporções, destacando a predominância de estudos sobre *Requisitos/Release Planning*, seguidos por trabalhos sobre métodos *Agile/TDD/XP* e por discussões epistemológicas de *Fundamentos Engenharia de Software*.

5. DISCUSSÃO

5.1 Requisitos e planejamento de releases como hipóteses

Os trabalhos do eixo *Requisitos/Release Planning* mostram que ideias de Popper ajudam a repensar como as decisões são tomadas em projetos feitos em etapas. Em vez de ver o plano de releases e os cálculos de valor de negócio como algo fixo e "certo", esses estudos tratam essas escolhas como hipóteses que podem ser derrubadas por dados de uso, custo ou desempenho. O foco deixa de ser provar antes do tempo que o plano está correto e passa a ser **montar experimentos que revelem onde e como ele falha**. Essa forma de pensar está de

acordo com Popper (2023), para quem decisões racionais devem expor suas conjecturas a testes fortes, e não apenas juntar exemplos que parecem confirmar o que já se acredita.

5.2 Métodos ágeis, TDD e cultura Fail-Fast

Os trabalhos do eixo *Agile/TDD/XP* defendem que métodos ágeis podem ser vistos como uma maneira de aplicar o racionalismo crítico no dia a dia das organizações (Shore; Warden, 2022). Ao propor ciclos curtos, reuniões de revisão frequentes e verificação constante do código, práticas como *Extreme Programming* criam um ambiente em que erros aparecem rápido, são discutidos em grupo e servem para ajustar o rumo do projeto. Com isso, o software deixa de ser uma "caixa-preta" estável e escondida, no sentido usado por Latour (2000), e passa a ser aberto o tempo todo à crítica. Nesse cenário, o desenvolvimento orientado a testes (TDD) faz com que cada teste automático funcione como uma tentativa explícita de mostrar que a versão atual do código está errada, reforçando a ideia popperiana de que o conhecimento avança quando se critica e descarta hipóteses que não resistem à experiência.

5.3 Fundamentos epistemológicos do software testing

Os artigos do eixo Fundamentos de Engenharia de Software discutem até onde os testes permitem confiar no comportamento de um sistema. Em geral, esses autores concordam que testes não garantem que o programa está "correto", mas apenas mostram que, em certas situações, ele ainda não falhou; isso combina com a noção de Popper de que resultados positivos em testes significam apenas uma corroboração provisória, ligada ao conjunto de testes já feito. A partir disso, defendem que as suites de teste também devem ser tratadas como objetos críticos, que podem e devem ser ampliados, modificados ou abandonados, assim como o próprio código. Essa visão se aproxima da posição de Sommerville (2019), segundo a qual a qualidade de software depende tanto do modo como o sistema é desenvolvido e testado quanto das características técnicas do produto final.

6. LIMITAÇÕES

O estudo apresenta limitações inerentes ao delineamento bibliométrico adotado e ao tamanho reduzido do corpus. Em primeiro lugar, a busca foi realizada em uma **única base de dados**, Lens.org, o que pode ter excluído publicações relevantes indexadas apenas em outras plataformas e afetado a representatividade dos resultados. Em segundo lugar, a seleção de registros centrais dependeu de critérios interpretativos de relevância, aplicados a título e resumo, o que introduz **viés do pesquisador** na definição do que conta como apropriação "explícita" de Popper na engenharia de software.

Além disso, a **escassez de palavras-chave** preenchidas nos metadados tornou necessário utilizar os campos de estudo como principal indicador de área, o que limita a granularidade da análise temática. Finalmente, o **número de trabalhos analisados** (17 centrais) impede qualquer generalização estatística robusta sobre a produção internacional, recomendando que os resultados sejam lidos como mapeamento exploratório de um nicho específico em vez de panorama abrangente.

7. CONCLUSÃO

A combinação entre racionalismo crítico popperiano, estudos de engenharia de software e ferramentas bibliométricas permitiu evidenciar que existe, ainda que modesto, um núcleo de literatura que articula de forma explícita conceitos como falseabilidade, conjecturas e refutações com problemas de validação, requisitos e métodos ágeis. A análise do corpus mostrou que essa apropriação se concentra em três frentes: uso de modelos falsificáveis em planejamento de releases e decisões de requisitos, interpretação de práticas ágeis como mecanismos de crítica contínua ao código e discussão epistemológica sobre o estatuto dos testes de software.

Tomados em conjunto, esses resultados reforçam a tese de que o software deve ser compreendido menos como produto de lógica infalível e mais como conjectura técnica sujeita a tentativas sistemáticas de refutação, em linha com a visão de Popper (2023) sobre o

conhecimento científico. Práticas como TDD, integração contínua e Fail-Fast, descritas por Shore e Warden (2022), emergem como instrumentos epistemológicos que aproximam o desenvolvimento de software de um método científico experimental, no qual a crítica, o erro e a correção permanente são centrais. Do ponto de vista social, a leitura de Latour (2000) lembra que a abertura das caixas-pretas digitais — por meio de auditorias, testes e revisões públicas — é condição para que sistemas que impactam a vida coletiva permaneçam sujeitos à crítica, evitando que se tornem autoridades opacas imunes à refutação.

REFERÊNCIAS BIBLIOGRÁFICAS

LATOUR, B. **Ciência em ação: Como seguir cientistas e engenheiros sociedade afora**. São Paulo: Editora Unesp, 2000. 438 p.

POPPER, K. **A Lógica da Pesquisa Científica**. [*S. l.*]: Cultrix, 20 mar. 2023. 456 p.

SHORE, J.; WARDEN, S. **The Art of Agile Development**. Beijing : Boston: O'Reilly Media, 2022. 537 p.

SOMMERVILLE, I. **Engenharia de Software.** 10. ed. [*S. l.*]: Pearson Universidades, 2019. 768 p.

Recebido em: 20/06/2026
Aprovado em: 15/07/2026
Publicado em: 09/08/2026