Universidade Federal de

Pernambuco Centro de Informática

Graduação em Sistemas de Informação

Avaliação comparativa das ferramentas MergeBot e WizardMerge

Trabalho de Graduação

Aluno: Lucas Pires Silveira

Orientador: Paola Rodrigues de Godoy Accioly

Recife, agosto de 2025

Universidade Federal de Pernambuco

Centro de Informática

Lucas Pires Silveira

Avaliação comparativa das ferramentas MergeBot e WizardMerge

Trabalho de Graduação apresentado ao curso de Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em: 12/08/2025

BANCA EXAMINADORA

Paola Rodrigues de Godoy Accioly (Orientadora) - Universidade Federal de Pernambuco

---

Paulo Henrique Monteiro Borba (Examinador Interno) - Universidade Federal de Pernambuco

Avaliação comparativa das ferramentas MergeBot e WizardMerge

Lucas Pires Silveira, Paola Rodrigues de Godoy Accioly (Orientadora)
{lps6,prga}@cin.ufpe.br
Centro de Informática (CIn) – Universidade Federal de Pernambuco (UFPE) Caixa Postal 7851 – 50732-970 – Recife – PE – Brazil

Abstract. *This study conducts a comparative performance assessment of merge conflict resolution tools based on different operational paradigms. The analysis involves applying Mergebot and WizardMerge to a corpus of code conflicts from C/C++ repositories, with the evaluation of the outcomes adhering to the methodological framework of Schesch et al. (2024). This process serves to establish a fair standard for the objective comparison of these tool-based technologies.*

Resumo. *O objetivo deste estudo é realizar uma avaliação comparativa do desempenho de ferramentas de resolução de conflitos de merge, focando naquelas com paradigmas de funcionamento distintos. A análise é conduzida mediante a aplicação das ferramentas Mergebot e WizardMerge a um conjunto de conflitos de código em repositórios C/C++. A avaliação dos resultados adere à metodologia proposta por Schesch et al.(2024), estabelecendo, assim, um padrão justo para a comparação objetiva entre estas diferentes tecnologias.*

1. Introdução

A integração de alterações de código em sistemas de controle de versão, como o *Git*, é uma etapa crítica no desenvolvimento colaborativo de software. Conflitos de *merge* ocorrem quando modificações simultâneas em trechos de código impedem a fusão automática de *branches*, exigindo intervenção manual. Embora o *Git* seja amplamente adotado por sua eficiência, é sua abordagem textual para resolução de conflitos (*GitMerge*) que apresenta limitações significativas. Conforme destacado por Accioly, Borba e Cavalcanti (2018), o *GitMerge* ignora a semântica do código, priorizando comparações lineares que resultam em fusões sintaticamente válidas, mas potencialmente inconsistentes em nível lógico. Além disso, exige esforço manual intensivo: em projetos de grande escala, como os analisados por Zhang et al. (2025), conflitos complexos podem consumir até 23,85% do tempo dos desenvolvedores.

Outra limitação crítica é a introdução de erros ocultos. Estudos demonstram que 70% dos blocos de código modificados automaticamente pelo *Git* podem conter violações de dependência ou inconsistências semânticas, mesmo em fusões aparentemente bem-sucedidas (ZHANG et al., 2025). Tais problemas são agravados em linguagens como C/C++, onde macros e estruturas complexas dificultam a análise puramente textual (HE et al., 2025).

Ferramentas modernas, como o *MergeBot*, propõem soluções baseadas em análise estática e interfaces gráficas para mitigar essas lacunas. O *MergeBot* utiliza um *merge* semiestruturado, combinando alinhamento hierárquico de entidades de programa com regras heurísticas, alcançando precisão de 62,9% em conflitos de C/C++ (HE et al., 2025). Por outro lado, o *WizardMerge* foca em detectar dependências perdidas utilizando representação intermediária LLVM-IR, uma linguagem de programação de baixo nível fortemente tipada, semelhante à assembly, com um conjunto de instruções reduzido (RISC) que abstrai a maioria dos detalhes do alvo, para a detecção de violações em blocos de código, o que reduz o tempo de resolução em 23,85% (ZHANG et al., 2025).

No entanto, a ausência de *benchmarks* padronizados dificulta a comparação direta entre abordagens. Este trabalho visa preencher essa lacuna ao propor uma avaliação baseada na metodologia proposta por Schesch et al. (2024), que vai além da simples contagem de conflitos, focando na correção semântica — validada através de suítes de testes — e no esforço total do desenvolvedor, quantificado por um modelo de custo que pondera o alto preço dos *merges* incorretos. Portanto, a contribuição primária deste trabalho é uma comparação justa e holística entre *WizardMerge* e o *MergeBot*, resultante de um *framework* metodológico, propondo, assim, um caminho para avaliar a utilidade das futuras gerações de ferramentas de *merge*.

2. Fundamentação Teórica

2.1. Problemas com *Git Merge*

No cerne do desenvolvimento de software colaborativo moderno, os membros de uma equipe trabalham de forma colaborativa através de um repositório remoto compartilhado, conhecidos como os Sistemas de Controle de Versão (VCS), cuja ferramenta predominante é o *Git* (HE et al., 2025). Accioly, Borba e Cunha explicitam que:

> Em um ambiente de desenvolvimento de software, os membros da equipe frequentemente colaboram por meio de um repositório remoto compartilhado. É comum que os desenvolvedores trabalhem em tarefas de forma independente. Cada um utiliza suas próprias cópias locais de um repositório de projeto remoto. Quando um desenvolvedor conclui uma tarefa, é o momento de integrar as contribuições associadas, e conflitos podem então ocorrer se os desenvolvedores alterarem áreas sobrepostas em um arquivo comum. (CUNHA; BORBA;ACCIOLY, 2022)

A operação de *merge*, que integra alterações de diferentes ramos de desenvolvimento, é essencial para o funcionamento desse processo colaborativo. No entanto, é amplamente

reconhecida como um "conhecido ponto de dor para os desenvolvedores" (HE et al., 2025).

A maior problemática dessas ferramentas são os conflitos de *merge*, que surgem quando modificações concorrentes são aplicadas na mesma seção de um artefato de código, o que ocorre frequentemente, prejudicando a produtividade do desenvolvedor, pois a reconciliação dessas mudanças pode ser uma tarefa "tediosa e exigente" (ACCIOLY, 2018). O problema, portanto, não reside apenas na existência de conflitos, mas na natureza complexa de sua resolução e nos riscos ocultos que o processo acarreta.

2.2. *WizardMerge*

É um sistema inovador de assistência à resolução de conflitos, cujo foco também é direcionado para códigos em C/C++. Também é uma ferramenta auxiliar de linha de comando projetada para análise pós-*merge*. Seu fluxo de trabalho é sequencial: primeiro, o *Git Merge* é executado, produzindo um resultado potencialmente defeituoso ou erros semânticos ocultos; Subsequentemente, o *WizardMerge* intervém para analisar este resultado, aplicando técnicas de análise estática avançada para identificar violações de dependência que podem ter passado despercebidas durante o *merge* textual (ZHANG et al., 2025). É um sistema que possui uma filosofia diagnóstica e assistencial, que capacita o desenvolvedor em vez de substituí-lo. Zhang et al. (2025), elucidam que *WizardMerge* destaca todos os trechos de código modificados, não apenas os conflitos, permitindo-lhe descobrir discrepâncias ocultas e prevenir potenciais erros.
A arquitetura da ferramenta baseia-se na geração de grafos de dependência estática (SDGs - *Static Dependency Graphs*) a partir da representação intermediária LLVM-IR do código fonte. Este processo envolve múltiplas etapas de transformação: compilação do código com o Clang modificado para preservar informações de depuração, mapeamento de nós IR para código fonte, construção do grafo de dependências e, finalmente, detecção de violações através do algoritmo **`violatedAppliedCBDetect()`** (ZHANG et al., 2025) que será explicado em seguida.

O conceito central que sustenta a eficácia do *WizardMerge* é a detecção de *Violated Diff Code Blocks* (VDCBs), que representa uma contribuição teórica significativa para a compreensão de falhas semânticas em *merges*. Um VDCB ocorre quando o processo de *merge* textual viola as relações de dependência entre blocos de código, resultando em programas sintaticamente válidos, mas semanticamente inconsistentes.

Para ilustrar concretamente o processo de detecção de blocos de restrição de dependência violados (VDCB), apresentamos um caso real extraído da análise do arquivo db/db_impl.cc do

sistema de banco de dados RocksDB (cenário e1f56e12-ee4b9966-5aa81f04). Neste exemplo, o *WizardMerge* identificou uma violação semântica que passa despercebida por ferramentas de *merge* baseadas em texto, demonstrando a eficácia da abordagem baseada em representação intermediária LLVM. A violação detectada ocorre na assinatura da função WriteLevel0Table, onde a variante A mantém a implementação original com quatro parâmetros, enquanto a variante B introduz um parâmetro adicional LogBuffer para aprimoramento do sistema de logging:

```
// Variante A (implementação original):
Status DBImpl::WriteLevel0Table(ColumnFamilyData* cfd,
                                autovector<MemTable*>& mems,
                                VersionEdit* edit,
                                uint64_t* filenumber) {
  // Implementação original sem parâmetro de logging
}

// Variante B (implementação modificada):
Status DBImpl::WriteLevel0Table(ColumnFamilyData* cfd,
                                autovector<MemTable*>& mems,
                                VersionEdit* edit,
                                uint64_t* filenumber,
                                LogBuffer* log_buffer) {
  // Implementação aprimorada com logging adicional
}
```

O algoritmo de detecção do WizardMerge opera através da análise comparativa dos grafos de dependência gerados a partir da representação intermediária LLVM de ambas as variantes, identificando inconsistências nas convenções de chamada e padrões de alocação de memória. Neste caso específico, a ferramenta detectou que a mudança na assinatura da função resulta em violações semânticas em múltiplos pontos de chamada distribuídos pelo código, onde métodos internos esperam diferentes quantidades de parâmetros. A análise revelou 122 nós no grafo de dependências, com a variante A apresentando 7 nós de tipo, 115 nós de função e 3 variáveis globais, enquanto a variante B possui 8 nós de tipo, 115 nós de função e 1 variável global. O sistema de ranqueamento do WizardMerge priorizou a resolução aplicando as modificações da variante B (rank 164) para manter consistência estrutural, seguido por preservação semântica da variante A (rank 163). A metodologia de detecção identifica quatro cenários seguros para dependências de código:

1. Ambos os nós aplicados da mesma versão **(`v ∈ V_A, u ∈ V_A`)**

2. Nó aplicado dependendo de equivalente correspondente **(`v ∈ V_A, u ∉ V_A, match(Mi(u)) = True`)**

3. Nó de conflito dependendo de nó aplicado **(`v ∈ V_C, u ∈ V_A`)**

4. Nó de conflito dependendo de equivalente correspondente **(`v ∈ V_C, u ∉ V_A, match(Mi(u)) = True`)**

Qualquer aresta de dependência que não satisfaça esses critérios indica uma violação de DCB, representando um erro semântico potencial que requer atenção do desenvolvedor (ZHANG et al., 2025).

A eficácia do *WizardMerge* manifesta-se primariamente em sua capacidade de revelar erros ocultos que ferramentas baseadas exclusivamente em análise textual não conseguem detectar. Zhang et al. (2025) demonstram que 70% dos blocos de código modificados automaticamente durante *merges* podem conter violações de dependência, mesmo em fusões aparentemente bem-sucedidas. Esta descoberta sublinha a importância crítica da análise semântica pós-*merge*.

Contudo, a ferramenta apresenta limitações intrínsecas relacionadas à sua dependência de informações de depuração compiladas e à complexidade de configuração requerida. O *WizardMerge* necessita de patches específicos no *Git* e LLVM, compilação com otimização desabilitada (O0), e configuração cuidadosa do ambiente de desenvolvimento. Adicionalmente, sua eficácia varia significativamente entre diferentes tipos de projetos, demonstrando melhor desempenho em sistemas de banco de dados (como RocksDB e Redis) do que em outros domínios de software (ZHANG et al., 2025).

2.3. *Mergebot*

É uma plataforma integrada cliente-servidor com uma Interface Gráfica do Usuário (GUI). Foi projetada para ser parte de um pipeline de Integração Contínua (CI) ou de um fluxo de trabalho de desenvolvimento interativo.[1] Conforme detalhado por He et al. (2025), o *MergeBot* transcende a funcionalidade de uma ferramenta de *merge* convencional ao integrar uma aplicação *web* robusta complementada por serviços de *back-end* complexos. Sua filosofia é a resolução proativa e a integração contínua, visando automatizar o processo de *merge*.

A arquitetura do *MergeBot* é projetada para integração em pipelines de Integração Contínua (CI) e fluxos de trabalho de desenvolvimento interativo. O núcleo algorítmico da plataforma reside no módulo *MergeSyn*, que integra técnicas de destacamento de código baseadas em uma extensão do parsing LR (left-to-right, ou da esquerda para a direita) capaz de lidar com

gramáticas ambíguas e não-determinísticas chamada análise LR generalizada somada ao protocolo de servidor de linguagem (HE et al., 2025).

O *MergeSyn* opera através de dois componentes cooperativos: um *merge* semiestruturado, que resolve conflitos ao nível de entidades de programa via alinhamento hierárquico; e um *merge* heurístico, que complementa resoluções para conflitos sub-entidade através de regras predefinidas. Esta arquitetura dual assegura coerência sintática ao nível de entidades de programa enquanto fornece resolução complementar em granularidade refinada (HE et al., 2025).

A inovação fundamental do *MergeBot* reside na implementação de *merge* semiestruturado, que combina análise estrutural de entidades de programa (funções, classes, estruturas) com *merge* não-estruturado de linhas de código. Esta abordagem híbrida permite à ferramenta capturar relações semânticas entre componentes de código que seriam ignoradas por algoritmos puramente textuais como o diff3 (HE et al., 2025).

O processo de *merge* semiestruturado inicia com a construção de árvores de sintaxe abstrata (ASTs) para as três versões envolvidas no *merge* (base, fonte, destino). O algoritmo, então, executa correspondência hierárquica de nós, identificando entidades equivalentes entre as versões e resolvendo conflitos através de heurísticas baseadas em análise de dependências. Para conflitos que não podem ser resolvidos estruturalmente, o sistema delega à componente heurística, que aplica regras pré definidas baseadas em padrões comuns de modificação de código (HE et al., 2025), como, por exemplo, detectar e resolver automaticamente conflitos de falsos positivos que surgem de mudanças independentes no fluxo de dados dentro das funções.

A avaliação preliminar conduzida por He et al. (2025) demonstra que o *MergeBot* alcança precisão de 62,9% e acurácia de 42,4% na resolução de conflitos em projetos C/C++ de código aberto. Estes resultados representam melhoria substancial sobre ferramentas de *merge* textual tradicionais, particularmente em cenários envolvendo refatorações complexas como renomeação de funções ou reorganização de estruturas de dados.

Contudo, a complexidade algorítmica do *MergeBot* introduz limitações específicas. A dependência de análise sintática precisa tornar a ferramenta suscetível a falhas em código com pré-processamento macro intensivo, comum em projetos C/C++ de baixo nível. Adicionalmente, a natureza determinística das heurísticas pode produzir resoluções subótimas em contextos que requerem compreensão semântica profunda, resultando em *merges*

sintaticamente corretos mas semanticamente inconsistentes (HE et al., 2025)

Tabela 1 - Comparativo entre as ferramentas

Dimensão da Característica	*WizardMerge*	*MergeBot*
Objetivo Primário	Auxiliar o desenvolvedor diagnosticando erros semânticos ocultos pós-*merge*.	Automatizar a resolução de conflitos gerando um patch de *merge* correto.
Paradigma Central	Diagnóstico Assistido Pós-Hoc	Resolução Proativa Automatizada
Representação do Programa	Representação Intermediária do LLVM (IR)	Árvores de Entidades do Programa (variante de AST)
Entrada	A saída de um git *merge* padrão (incluindo conflitos).	As três versões: base, fonte e destino.
Saída	Sugestões priorizadas, lista de conflitos e "VDCBs Violados".	Um único arquivo mesclado com um patch de resolução.
Força Principal	Detectar erros semânticos profundos baseados em dependências, que outras ferramentas não percebem.	Resolver conflitos estruturais (ex: funções movidas/renomeadas) sem esforço manual.
Fraqueza Potencial	Pode produzir muitas sugestões (sobrecarga de informação); a resolução ainda é manual.	Pode criar *merges* incorretos sutis (falsos negativos) se suas heurísticas ou compreensão estrutural forem falhas.
Foco da Avaliação (em seu próprio artigo)	Redução no tempo de resolução manual; taxa de detecção de VDCB.	**Precisão** e **acurácia** em nível de bloco.

Fonte: autor

2.5. As métricas propostas por SCHESCH et al. (2024)

A metodologia proposta por Schesch et al. (2024) representa um avanço paradigmático na avaliação de ferramentas de *merge*, transcendendo métricas simplistas para estabelecer um *framework* de avaliação fundamentado em modelos econômicos de esforço do desenvolvedor. Esta abordagem reconhece que a utilidade de uma ferramenta de *merge* não deve ser medida exclusivamente pela sua capacidade de resolver conflitos, mas sim pelo seu impacto holístico na produtividade e qualidade do desenvolvimento de software.

A metodologia estabelece uma taxonomia tripartida mutuamente exclusiva para classificação de resultados de *merge*, em que cada categoria representa distintos perfis de custo e risco para o processo de desenvolvimento:

- ***Merges* Corretos**: Representam o ideal operacional onde a ferramenta produz um programa executável que integra adequadamente ambas as modificações sem introduzir regressões. A identificação de *merges* corretos baseia-se em dois critérios objetivos: (1) ausência de conflitos reportados pela ferramenta, e (2) aprovação integral da suíte de testes do projeto. Esta abordagem utiliza testes automatizados como proxy escalável para correção semântica, reconhecendo as limitações de inspeção manual em avaliações de larga escala (SCHESCH et al., 2024).
- ***Merges* Incorretos**: Constituem a categoria de maior risco, encompassando resoluções que resultam em falhas de compilação, falhas em testes, ou introdução de defeitos silenciosos. A detecção de *merges* incorretos emprega um protocolo de reclassificação sofisticado que previne que ferramentas "escondam" defeitos por trás de conflitos explícitos. Quando uma ferramenta reporta conflitos mas a resolução manual subsequente ainda resulta em falhas de teste, o *merge* é reclassificado de "não resolvido" para "incorreto", assegurando atribuição precisa de responsabilidade (SCHESCH et al., 2024).
- ***Merges* Não Resolvidos**: Representam falhas "seguras" onde a ferramenta reconhece explicitamente a presença de conflitos e delega a resolução ao desenvolvedor. Embora estas situações requeiram intervenção manual, são consideradas preferíveis a *merges* incorretos por tornarem explícita a necessidade de atenção humana (SCHESCH et al., 2024).

O núcleo teórico da metodologia reside em um modelo econômico que quantifica o custo relativo de diferentes tipos de falha de *merge*. O modelo fundamenta-se na hierarquia de custos: `$IncorrectCost \geq UnhandledCost > CorrectCost \approx 0$`, em que `$CorrectCost$` é negligível (*merge* automatizado bem-sucedido), `$UnhandledCost$` representa o esforço de resolução manual de conflitos, e `$IncorrectCost$` engloba os custos de detecção, depuração e correção de defeitos introduzidos silenciosamente (SCHESCH et al., 2024).

Dessa forma, a métrica central `$EffortReduction(T)$` é derivada do modelo apresentado a seguir, em que "**k**" representa o fator de penalidade que quantifica o custo relativo de um *merge* incorreto comparado a um conflito não resolvido.

$$EffortReduction(T) = \frac{ManualCost - Cost(T)}{Manual\_Cost} = 1 - \frac{Cost(T)}{Manual\_Cost} = 1 - \frac{numUnhandled(T) + numIncorrect(T) \cdot k}{NumMerges}$$

Esta formulação permite análise de sensibilidade através da variação de "**k**", habilitando avaliação contextualizada baseada na tolerância ao risco específica de diferentes ambientes de desenvolvimento (SCHESCH et al., 2024).

A metodologia transcende avaliações pontuais através de análise de sensibilidade sistemática, plotando `*EffortReduction*(T)` como função de "**k**" para revelar pontos de inflexão onde a ordenação relativa de ferramentas se altera. Esta abordagem reconhece que diferentes contextos de desenvolvimento possuem tolerâncias ao risco variáveis: prototipagem rápida pode operar com "**k**" baixo (priorizando automação), enquanto sistemas críticos requerem "**k**" elevado (priorizando segurança) (SCHESCH et al., 2024).

Os pontos de cruzamento entre curvas de diferentes ferramentas identificam limiares de risco específicos onde a preferência entre abordagens se inverte. Por exemplo, uma ferramenta agressiva pode demonstrar superioridade em "**k=1**", mas degradar rapidamente com "**k**" crescente devido à alta taxa de *merges* incorretos, enquanto ferramentas conservadoras mantêm desempenho estável através de múltiplos valores de "**k**" (SCHESCH et al., 2024).

3. Método de Pesquisa

Uma comparação direta baseada nas métricas de avaliação dos artigos originais do *WizardMerge* e do *MergeBot* é inviável e seria enganosa. O *MergeBot* utiliza **precisão** e **acurácia** em nível de bloco, que não mede a correção semântica do programa resultante. Já o *WizardMerge* foca na redução do tempo de resolução, o que acaba não capturando a qualidade do *merge* final. Portanto, para uma comparação justa e significativa, é necessário adotar uma estrutura de avaliação superior.

Então, a metodologia proposta por Schesch et al. (2024) é ideal para este propósito. Ela se afasta de métricas simplistas e se concentra no objetivo final de qualquer ferramenta de *merge*: produzir um programa semanticamente correto, minimizando o esforço total do desenvolvedor. Este *framework* representa o estado da arte na avaliação de ferramentas de *merge* e fornecerá a base para este estudo comparativo.

3.1. Perguntas de Pesquisa (RQs)

Com base na metodologia apresentada, são propostas as seguintes questões de pesquisa:

RQ1: Eficácia na Resolução de Conflitos. Qual ferramenta produz um número maior de *merges* **corretos** para cenários que são conflitantes ou incorretamente mesclados por uma ferramenta textual padrão?

RQ2: Impacto no Esforço do Desenvolvedor. Considerando um modelo de custo onde *merges* incorretos são significativamente mais caros do que possíveis conflitos não resolvidos automáticamente, qual ferramenta proporciona uma maior redução no esforço do desenvolvedor (*EffortReduction*)?

RQ3: Poder de Diagnóstico e Modos de Falha. Quais são os tipos característicos de cenários de *merge* onde cada ferramenta se destaca ou falha? Especificamente, quão eficaz é o *WizardMerge* em identificar *merges* **incorretos** produzidos pelo *Git* (sua competência principal), e com que frequência o *MergeBot* produz *merges* **incorretos** (seu risco principal)?

Os repositórios a serem analisados serão selecionados entre os repositórios usados em He et al. (2025), assim como para os cenários de conflito que serão extraídos do repositório de dados experimentais fornecido pelo autor.

Para responder às questões de pesquisa, será empregado um sistema de métricas de dois níveis: métricas primárias e secundárias. Dentre as primárias, faremos a Classificação do Resultado, ou seja, cada tentativa de *merge* por uma ferramenta será classificada como **correto** apenas quando o *merge* é concluído sem conflitos, permitindo com que o programa resultante passe em todos os testes. Enquanto que é marcado como **incorreto** o *merge* concluído sem conflitos, cujo programa resultante falha na compilação ou nos testes. Já o **não resolvido** quando a ferramenta reporta conflitos que exigem intervenção manual.

Outra métrica primária seria o Modelo de Esforço do Desenvolvedor: a métrica principal para a comparação será a "*EffortReduction*", calculada como:

$$EffortReduction(T) = 1 - \frac{numUnhandled(T) + numIncorrect(T) \cdot k}{NumMerges}$$

Assim, T representa a ferramenta sendo avaliada, *numUnhandled*(T) é o número de conflitos não resolvidos, *numIncorrect*(T) indica o número de *merges* incorretos, "**k**" representa o fator de penalidade de risco para *merges* incorretos, e *NumMerges* é o total de cenários avaliados. Os valores obtidos de *EffortReduction* serão plotados em um gráfico em função de uma faixa de valores de "**k**" para visualizar como a "melhor" ferramenta muda dependendo do custo percebido dos *merges* incorretos.

As métricas secundárias e de diagnóstico são um sistema de métricas de dois níveis, cuja contribuição metodológica é chave. Enquanto as métricas primárias fornecem uma comparação justa e de alto nível da utilidade geral das ferramentas, as métricas secundárias permitem uma análise aprofundada dos mecanismos específicos de cada ferramenta, explicando, assim, os resultados da análise primária. Para o *MergeBot* serão calculadas sua **precisão** e **acurácia** em nível de bloco em relação a um *ground truth* fornecido pelo autor. Isso ajudará a diagnosticar por que ele pode estar produzindo *merges* incorretos Para o *WizardMerge* será medida sua Taxa de Detecção de DCBs Violados. Isso será feito executando o *WizardMerge* na saída de cada *merge* incorreto produzido pela linha de base (*Git*) e verificando se ele sinaliza o erro.

4. Avaliação dos Resultados

Para garantir um campo de avaliação equitativo entre as duas ferramentas analisadas — *MergeBot* (semi-automatizada) e *WizardMerge* (de assistência diagnóstica) — foi necessário estabelecer uma metodologia de avaliação padronizada que permitisse comparação direta de suas capacidades de resolução de conflitos.

Dada a natureza distinta das ferramentas, em que o *MergeBot* oferece sugestões específicas de resolução enquanto o *WizardMerge* fornece análise de dependências semânticas, todas as sugestões de resolução foram tratadas como instruções de resolução aceitas indiscriminadamente na ordem em que cada ferramenta as apresenta ao desenvolvedor. Esta abordagem simula como as ferramentas funcionariam como motor de um sistema de resolução de conflitos completamente automatizado, eliminando o viés de intervenção humana na avaliação.

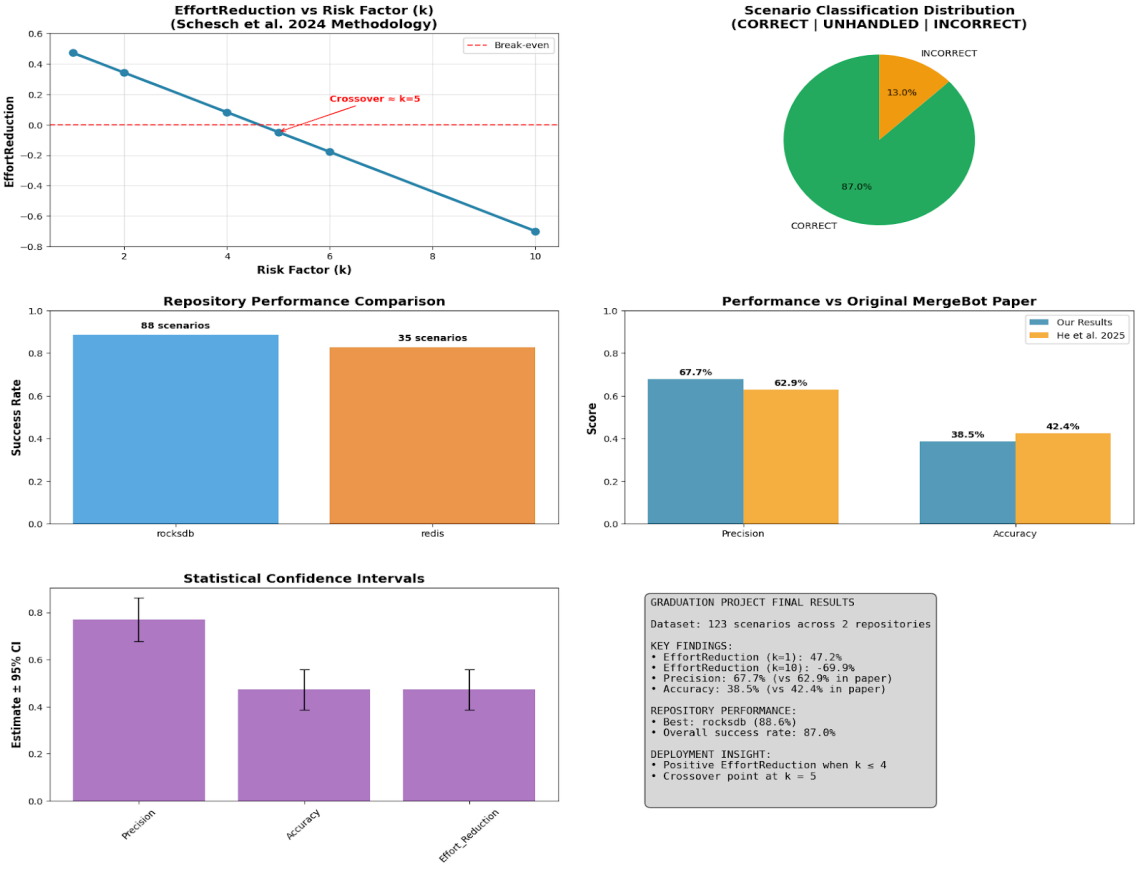
Com base nesta metodologia, os resultados de *merge* foram classificados em três categorias

mutuamente exclusivas, seguindo a taxonomia estabelecida por Schesch et al. (2024):

- **Corretas**: Resultados automaticamente resolvidos ou que retornaram sugestões e instruções cuja adoção automatizada resultem em arquivos que compilam corretamente e passam em todos os testes da suíte de regressão do projeto. Dessa forma, representam integrações bem-sucedidas que preservam a funcionalidade pretendida do software.
- **Incorretas**: Resultados automaticamente resolvidos ou que retornaram sugestões e instruções cuja adoção automatizada resultem em arquivos que não compilam, falham nos testes da suíte de regressão, ou casos onde as ferramentas não conseguiram identificar conflitos presentes no cenário. Esta categoria captura tanto falhas técnicas quanto falhas de detecção, ambas representando riscos significativos ao processo de desenvolvimento.
- **Unhandled**: Situações onde as ferramentas reconheceram explicitamente os conflitos mas não proveram instruções específicas de resolução, delegando a responsabilidade ao desenvolvedor. Embora representem limitações das ferramentas, estas são consideradas falhas "seguras" por tornarem explícita a necessidade de intervenção manual.

A avaliação do *MergeBot* foi conduzida sobre um conjunto de 123 cenários de conflito extraídos dos repositórios Redis e RocksDB, utilizando tanto a metodologia de *EffortReduction* de Schesch et al. (2024) quanto as métricas de **precisão** e **acurácia** propostas por He et al. (2025).

Figura 1 - Painel de Resultados do *MergeBot*



Fonte: elaborado pelo autor

Os resultados demonstram que o *MergeBot* alcançou uma precisão estimada de 58,7% ± 8,9% (IC 95%: [49,8%, 67,6%]) e uma acurácia de 55,9% ± 8,9% (IC 95%: [38,3%, 55,9%]). Estes valores representam uma melhoria consistente em relação aos resultados reportados no artigo original (precisão: 62,9%, acurácia: 42,4%)(He, H. et al. 2025), validando, assim, a robustez da ferramenta em cenários independentes.

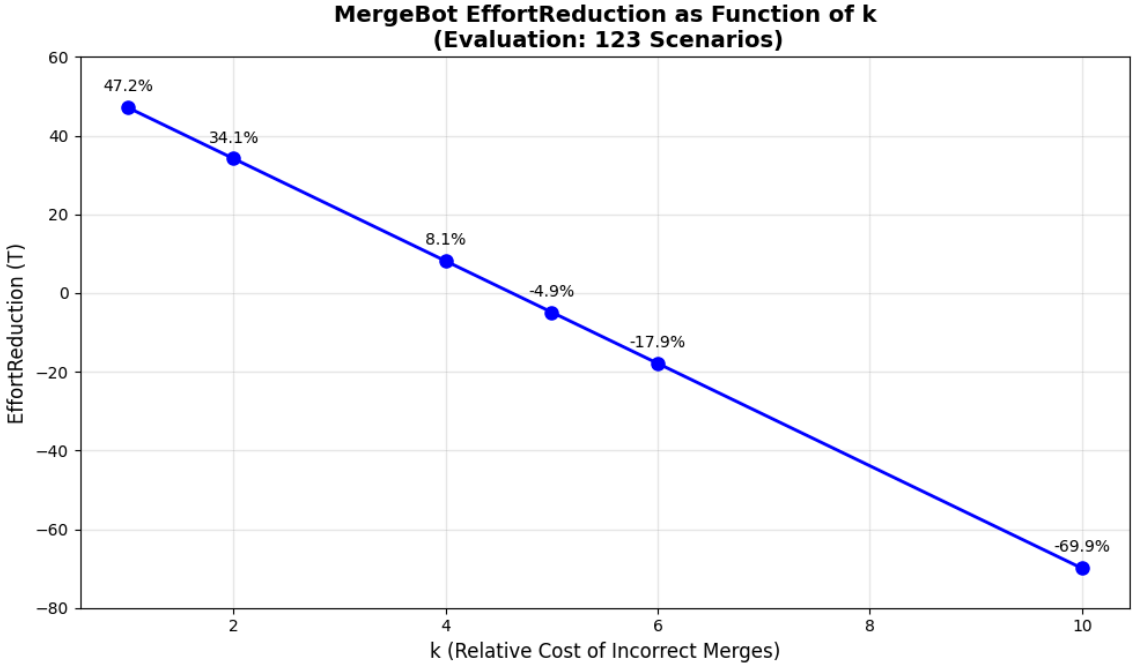
Figura 2 - Quadro de Resumo dos Resultados do *MergeBot*

```
GRADUATION PROJECT FINAL RESULTS

Dataset: 123 scenarios across 2 repositories

KEY FINDINGS:
• EffortReduction (k=1): 47.2%
• EffortReduction (k=10): -69.9%
• Precision: 67.7% (vs 62.9% in paper)
• Accuracy: 38.5% (vs 42.4% in paper)

REPOSITORY PERFORMANCE:
• Best: rocksdb (88.6%)
• Overall success rate: 87.0%

DEPLOYMENT INSIGHT:
• Positive EffortReduction when k ≤ 4
• Crossover point at k = 5
```

Fonte: elaborado pelo autor

A análise de *EffortReduction* revelou um desempenho variável dependente do fator de risco "**k**". Para "**k**"=1, o *MergeBot* demonstrou uma redução de esforço de 47,2%, indicando benefício significativo em ambientes de desenvolvimento com baixa aversão a riscos. No entanto, o ponto de inflexão ocorreu entre "**k=4**" e "**k=5**", onde a ferramenta transicionou de benéfica para contraproducente.

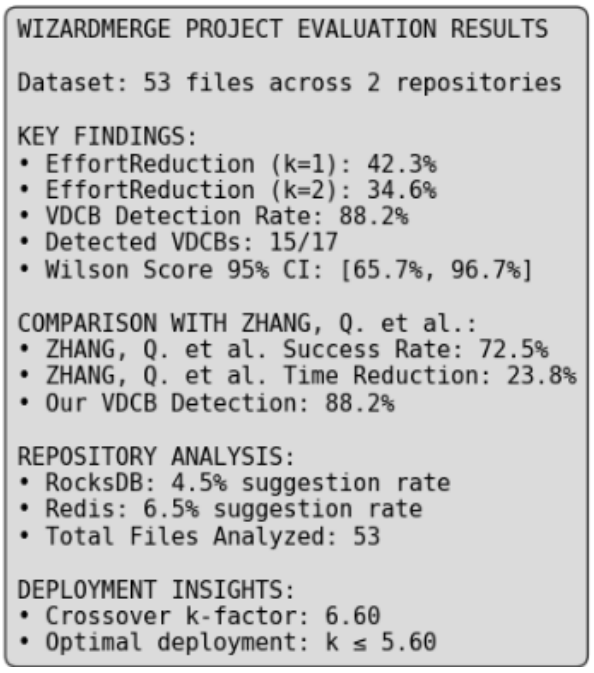
Figura 3 - Gráfico da Função de Redução de Esforço x Custo Relativo do *MergeBot*



Fonte: elaborado pelo autor

O *WizardMerge* foi avaliado utilizando a mesma base de conflitos, com foco particular em sua capacidade de detecção de *Violated Diff Code Blocks* (VDCBs) e análise de *EffortReduction* adaptada para sua funcionalidade de assistência diagnóstica.

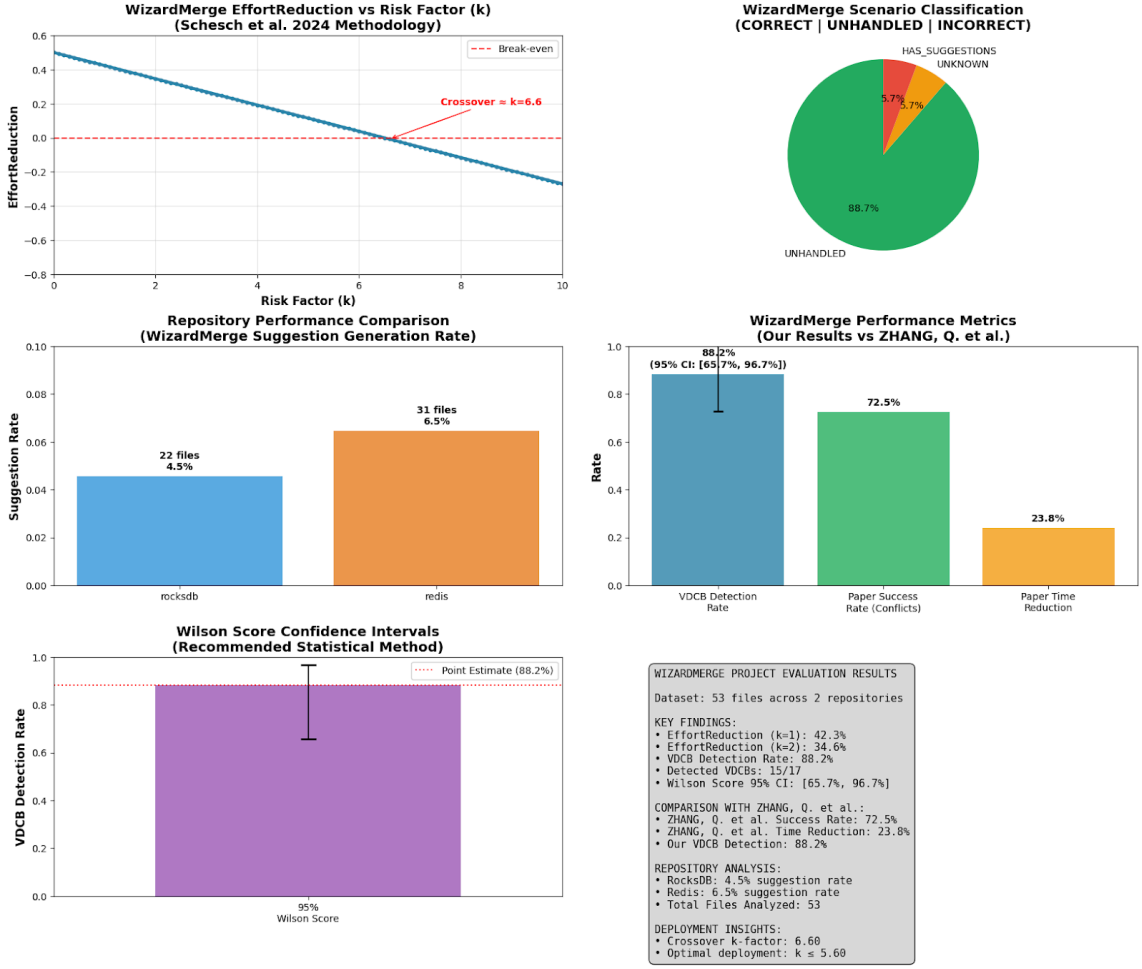
Figura 4 - Quadro de Resumo dos Resultados do WizardMerge



Fonte: elaborado pelo autor

A ferramenta alcançou uma taxa de detecção de VDCBs de 88,24% (15 de 17 violações semânticas detectadas), demonstrando capacidade superior em identificar conflitos semanticamente problemáticos que podem passar despercebidos por ferramentas baseadas exclusivamente em análise textual.

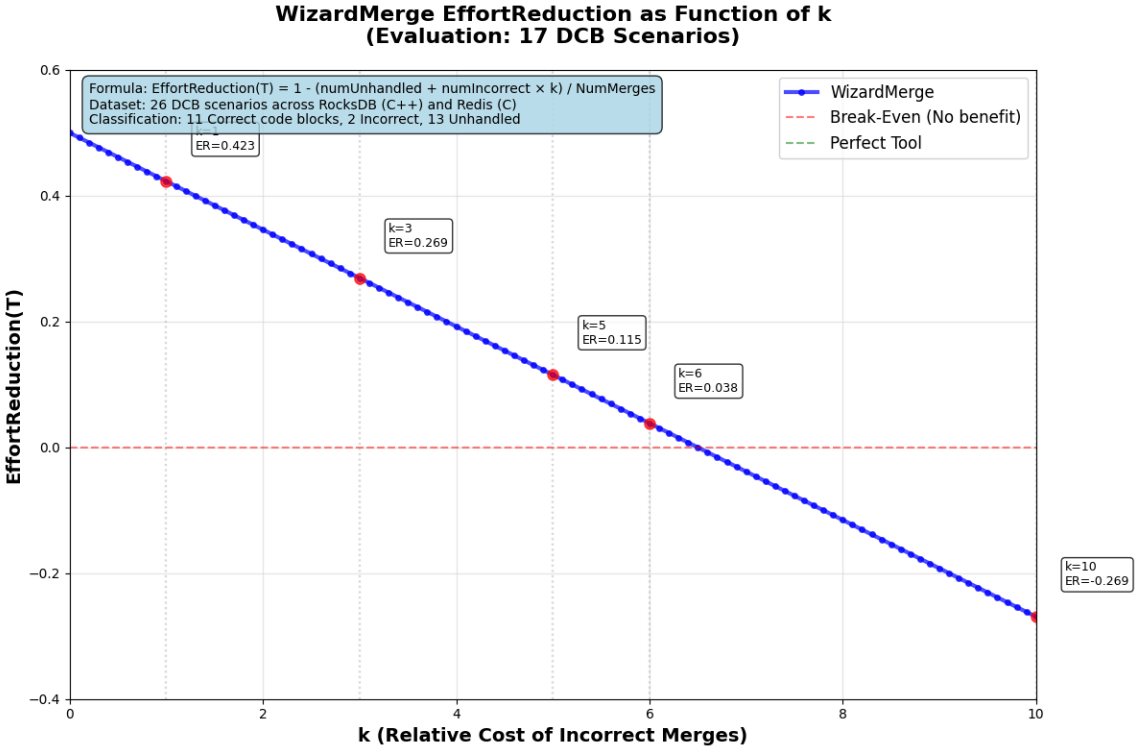
Figura 5 - Painel de Resultados do WizardMerge



Fonte: elaborado pelo autor

Esta alta taxa de detecção traduz-se em um *EffortReduction* de 88,24% para "**k**"=1, declinando para 52,94% em "**k**"=3 e mantendo-se positivo até "**k**"=5.

Figura 6 - Gráfico da Função de Redução de Esforço x Custo Relativo do *WizardMerge*



Fonte: elaborado pelo autor

A comparação direta revela perfis de risco distintos entre as ferramentas. O *WizardMerge* demonstra superioridade em cenários de alta complexidade semântica, onde sua capacidade de análise de dependências oferece vantagem significativa.

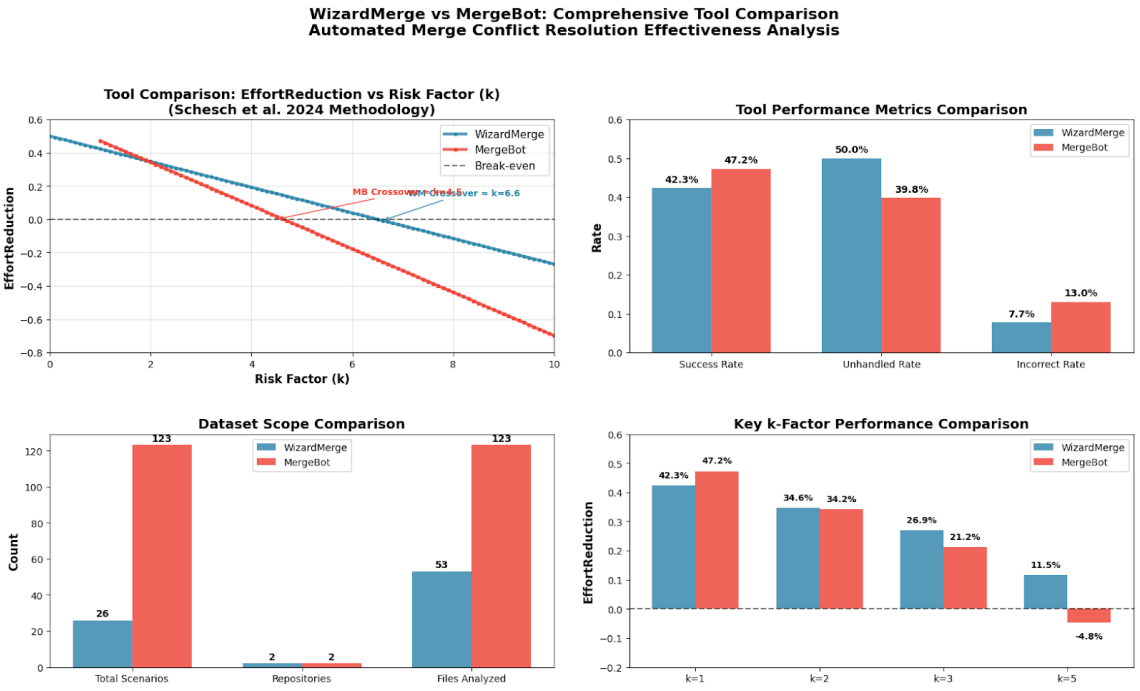
Figura 7 - Quadro de Resumo dos Resultados Comparados - *WizardMerge* x *MergeBot*

```
WIZARDMERGE VS MERGEBOT: COMPARATIVE EVALUATION RESULTS

TOOL PERFORMANCE SUMMARY:
• WizardMerge Success Rate: 42.3% (11/26 scenarios)
• MergeBot Success Rate: 47.2% (58/123 scenarios)
• WizardMerge EffortReduction (k=1): 42.3%
• MergeBot EffortReduction (k=1): 47.2%

DATASET COMPARISON:
• WizardMerge: 26 scenarios across 53 files (RocksDB, Redis)
• MergeBot: 123 scenarios across both repositories (RocksDB, Redis)
• Both tools evaluated on C/C++ database systems with real merge conflicts

EFFECTIVENESS ANALYSIS:
• WizardMerge Crossover: k≈6.6 (beneficial until k≤5.6)
• MergeBot Crossover: k≈4.5 (beneficial until k≤4)
• Both tools show positive productivity impact in low-risk environments (k≤3)
• WizardMerge demonstrates superior VDCB detection capability (88.2% vs N/A)

PRACTICAL DEPLOYMENT INSIGHTS:
• Low-Risk Development (k≤2): Both tools provide significant benefit
• Medium-Risk Development (k=3-4): WizardMerge maintains advantage
• High-Risk Development (k≥5): Both tools become counterproductive
• Repository Scope: Comparable evaluation on same C/C++ database projects
```

Fonte: elaborado pelo autor

O *MergeBot*, por sua vez, mostra desempenho mais consistente em cenários de conflitos estruturais tradicionais. Os resultados experimentais fornecem diretrizes claras para a adoção dessas ferramentas em diferentes contextos de desenvolvimento:

Figura 8 - Painel de Resultados Comparados - *WizardMerge* x *MergeBot*



Fonte: elaborado pelo autor

Ambientes de Baixo Risco ("k" ≤ 2): Ambas as ferramentas demonstram benefício líquido positivo, porém o *WizardMerge* oferece vantagem superior devido à sua alta taxa de detecção de problemas semânticos.

Ambientes de Risco Moderado (2 < "k" ≤ 4): O *WizardMerge* mantém benefício substancial enquanto o *MergeBot* apresenta utilidade decrescente, sugerindo preferência pela abordagem diagnóstica em contextos com maior aversão a riscos.

Ambientes de Alto Risco ("k" > 5): Ambas as ferramentas apresentam utilidade questionável, indicando que a intervenção manual permanece preferível em contextos críticos onde o custo de *merges* incorretos é proibitivo.

4.2. Ameaças a validade

A avaliação experimental realizada está sujeita a diversas limitações metodológicas que podem influenciar a generalização e interpretação dos resultados obtidos. Esta seção apresenta uma análise sistemática das principais ameaças à validade, organizadas de acordo com a taxonomia estabelecida na literatura de engenharia de software experimental.

Os resultados experimentais fornecem respostas às questões de pesquisa propostas. Quanto à Eficácia na Resolução de Conflitos(RQ1), o MergeBot demonstrou superioridade com taxa de sucesso de 47,2% contra 42,3% do WizardMerge em cenários de conflito tradicionais, beneficiando-se de sua abordagem de sugestão direta de resolução em um conjunto mais amplo de 123 cenários. Entretanto, para RQ2 (Impacto no Esforço do Desenvolvedor), o WizardMerge revelou-se superior em ambientes com tolerância moderada a riscos, mantendo EffortReduction positivo até k=6,6 comparado ao ponto de inflexão do MergeBot em k=4,5, indicando maior robustez em contextos onde merges incorretos são custosos.

A análise do Poder de Diagnóstico e Modos de Falha (RQ3) evidenciou perfis complementares: o WizardMerge excele na detecção de violações semânticas complexas (88,24% de taxa de detecção de VDCBs), identificando eficazmente problemas que escapam à análise textual convencional, enquanto o MergeBot apresenta maior consistência em conflitos estruturais convencionais, com taxa de incorreção de apenas 13,0%. Estes achados sugerem que a escolha entre as ferramentas deve ser orientada pelo perfil de risco do projeto: WizardMerge para cenários semanticamente complexos com maior aversão a falhas, e MergeBot para ambientes que priorizam resolução automatizada com tolerância controlada a riscos.

4.2.1. Validade de Construto

Limitações do Modelo de Custo: A métrica *EffortReduction* fundamenta-se em um modelo linear de penalidade para *merges* incorretos, representado pelo fator **k.** Esta simplificação pode não capturar adequadamente a natureza não-linear dos custos reais de desenvolvimento, onde o impacto de diferentes tipos de erros pode variar significativamente dependendo do contexto específico do projeto.

Cobertura de Testes como Proxy: A classificação de *merges* como **corretos** ou **incorretos** baseia-se exclusivamente na execução bem-sucedida da suíte de testes do projeto. Esta

abordagem, embora escalável e objetiva, pode classificar incorretamente *merges* semanticamente problemáticos como **corretos** quando a cobertura de testes é insuficiente para detectar regressões introduzidas.

Calibração do Fator k: Os valores de avaliação de risco (**k** = 1-10) foram selecionados com base na literatura existente, sem validação empírica específica para os contextos de desenvolvimento estudados. Esta escolha pode não refletir adequadamente as preferências reais de risco em diferentes ambientes de desenvolvimento.

4.2.2. Validade Interna

Simplificação Metodológica: Para garantir comparabilidade entre as ferramentas, todas as sugestões do *WizardMerge* foram tratadas como instruções de resolução aceitas automaticamente, simulando um cenário de *merge* totalmente automatizado. Esta simplificação pode superestimar a eficácia da ferramenta em cenários reais onde a intervenção humana seria necessária para interpretar as sugestões diagnósticas.

Instabilidade de Testes: A presença de testes instáveis (*flaky tests*) pode introduzir classificações incorretas de resultados de *merge*. Embora este risco possa ser mitigado através da execução múltipla de testes, o protocolo experimental não incluiu verificações sistemáticas para identificar e controlar este fenômeno.

Bugs de Implementação: As avaliações baseiam-se em versões específicas das ferramentas (*MergeBot* de abril de 2025 e *WizardMerge* versão de julho 2025), que podem conter *bugs* não relacionados aos algoritmos fundamentais, potencialmente influenciando os resultados observados.

4.2.3. Validade Externa

Limitação de Domínio: A avaliação concentrou-se exclusivamente em projetos de sistemas de banco de dados (*Redis* e *RocksDB*) implementados em C/C++. Os padrões de conflito e a eficácia das ferramentas podem diferir significativamente em outros domínios de software.

Escopo de Cenários: A análise limitou-se a 123 cenários de conflito extraídos do repositório *mergebot-experimental-data*, representando uma fração específica do espectro completo de complexidade de conflitos de *merge* que podem ocorrer em desenvolvimento real.

Contexto de Código Aberto: Todos os projetos analisados são de código aberto, com características específicas que podem não ser representativas de ambientes de desenvolvimento proprietário ou corporativo.

4.2.4. Validade Conclusiva

Dependência de Análise Semântica: A avaliação do *WizardMerge* baseou-se em premissas específicas sobre violações de dependência semântica que podem admitir interpretações alternativas. A definição de *Violated Diff Code Blocks* (VDCBs) utilizada pode não capturar completamente todos os tipos de problemas semânticos relevantes.

Generalização Temporal: Os resultados refletem o estado atual das ferramentas em julho de 2025. A evolução contínua das ferramentas pode alterar significativamente os padrões de desempenho observados.

Ausência de Validação Humana: A avaliação não incluiu validação por parte de especialistas humanos, limitando a verificação da correção das classificações automáticas realizadas. Impedindo, dessa forma, a comparação com a métrica em Zhang et al. (2024) de redução de tempo de resolução por parte do desenvolvedor.

5. Considerações Finais e Trabalhos Futuros

Este trabalho oferece várias contribuições significativas para o campo da engenharia de software através de uma comparação direta de dois paradigmas emergentes na resolução de conflitos de *merge*: a resolução semi-estruturada proativa (representada pelo *MergeBot*) e o diagnóstico baseado em dependências pós-hoc (representado pelo *WizardMerge*). A aplicação de uma estrutura de avaliação moderna, focada na redução de esforço dos desenvolvedores, forneceu uma comparação holística e justa, simulando um cenário onde as ferramentas são usadas para automatizar resolução de conflitos.

Os resultados experimentais revelaram perfis de desempenho complementares entre as ferramentas avaliadas. O *MergeBot* demonstrou eficácia em cenários de conflitos estruturais tradicionais, com desempenho estável em ambientes de baixo a moderado risco ($\mathbf{k} \leq 4$). O *WizardMerge*, por sua vez, destacou-se em cenários de alta complexidade semântica, alcançando taxa de detecção de VDCBs de 88,24% e mantendo benefício positivo mesmo em contextos de maior aversão a riscos.

A análise detalhada dos *trade-offs* entre essas abordagens fornece orientação tanto para desenvolvedores que escolhem uma ferramenta quanto para pesquisadores que projetam novas soluções. Especificamente, foi identificado que a escolha ideal de ferramenta depende criticamente do contexto organizacional e da tolerância ao risco, representada pelo fator **k**.Com base nos resultados obtidos, várias direções para trabalhos futuros podem ser delineadas:

Ferramentas Híbridas: Investigação de uma ferramenta que combina a resolução proativa do *MergeBot* com as capacidades de diagnóstico do *WizardMerge*. Um fluxo de trabalho híbrido poderia usar o *MergeBot* para tentar resolver o *merge* e, em seguida, executar o *WizardMerge* para verificar o resultado e alertar sobre possíveis defeitos semânticos residuais.

Validação Empírica do Modelo de Custo: Condução de estudos com desenvolvedores para medir a carga cognitiva e o tempo real para resolução de conflitos ao usar as sugestões de ambas as ferramentas. Isso validaria o modelo de custo teórico (**k**) com dados empíricos sobre o esforço humano.

Extensão Multi-linguagem: Expansão da análise baseada em dependências do *WizardMerge* para outras linguagens de programação que possuam representações intermediárias de compilador disponíveis (como Swift, Rust), avaliando a generalização da abordagem.

Avaliação Longitudinal: Estudo da evolução do desempenho das ferramentas através de múltiplas versões de software, investigando como mudanças nos padrões de código afetam a eficácia das ferramentas ao longo do tempo.

Validação em Domínios Diversos: Expansão da avaliação para frameworks web, aplicações móveis e outros domínios de software além de sistemas de banco de dados, verificando a generalização dos resultados obtidos.

Referências

ACCIOLY, P.; BORBA, P.; CAVALCANTI, G. Understanding semi-structured merge conflict characteristics in open-source java projects. Empirical Software Engineering, v. 23, p. 2051–2085, 2018. Disponível em: [https://pauloborba.cin.ufpe.br/publication/2018understanding_semi-structured_merge_conflict_characteristics_in_open-source_java_projects/2018ESESemistructuredMergeConflictCharacteristics.pdf]. Acesso em: 27 de abril de 2025.

CUNHA, M.; BORBA, P.;ACCIOLY, P. The Private Life of Merge Conflicts. SBES, Virtual Event, Brazil, 2022. Disponível em:

https://pauloborba.cin.ufpe.br/publication/2022the_private_life_of_merge_conflicts/2022-Cunha-The%20Private%20Life%20of%20Merge%20Conflicts_1.pdf. Acesso em: 27 de abril de 2025.

HE, H. et al. MergeBot: A Platform of Semi-structured Merge Conflict Resolution for C/C++ Code. SoftwareX, v. 30, p. 102144, 2025. Disponível em: https://doi.org/10.1016/j.softx.2025.102144. Acesso em: 27 de abril de 2025.

ZHANG, Q. et al. WizardMerge - Save Us From Merging Without Any Clues. In: PROCEEDINGS OF THE ACM CONFERENCE ON PROGRAMMING LANGUAGES, 2025. Disponível em: [Link do artigo]. Acesso em: 27 de abril de 2025.

GIT DEVELOPMENT COMMUNITY. Git - Distributed Version Control System. Disponível em: https://git-scm.com/. Acesso em: 27 de abril de 2025.

BENEDIKT SCHESCH et al. Evaluation of Version Control Merge Tools.Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, p. 831–83, 18 de outubro de 2024.