UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Isabela Carneiro Leão Menezes

Análise comparativa de ferramentas de *diff* textual e sintático

RECIFE
2025
UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

ISABELA CARNEIRO LEÃO MENEZES

Análise comparativa de ferramentas de *diff* textual e sintático

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Orientador(a): Paulo Borba

RECIFE

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Menezes, Isabela Carneiro Leão.
Análise comparativa de ferramentas de diff textual e sintático / Isabela Carneiro Leão Menezes. - Recife, 2025.
54 p. : il., tab.

Orientador(a): Paulo Borba
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Sistemas de Informação - Bacharelado, 2025.
Inclui referências, apêndices.

1. Diff. 2. Gumtree. 3. Git. 4. Diff Sintático. 5. ASTs. 6. Evolução de Software. I. Borba, Paulo. (Orientação). II. Título.

000 CDD (22.ed.)

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Isabela Carneiro Leão Menezes

Análise comparativa de ferramentas de diff textual e sintático

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovado em: 04/04/2025

Banca Examinadora:

Orientador(a)

Paulo Borba
Doutor(a)

Examinador(a)

Kiev Santos da Gama
Doutor(a)

AGRADECIMENTOS

Agradeço, primeiramente, à minha família, que sempre me apoiou e me deu colo durante todo o processo de graduação. Agradeço à minha mãe, Renata, por sempre buscar me ajudar e apoiar em todas as minhas escolhas. Ao meu pai, Henrique, por guiar meus caminhos e ser minha inspiração profissional. À minha irmã, Luiza, que, mesmo de longe, me acompanhou em todas as etapas e sempre foi escuta, inspiração e conforto nos momentos difíceis. E à minha madrasta, Magdala, que me incentivou a seguir em frente, me orientou ao longo desses anos e foi essencial para que eu não desistisse no meio do caminho.

Agradeço também ao meu orientador, Paulo, que me apresentou a um novo caminho dentro da computação. Com ele, conheci muitas pessoas, me senti acolhida no grupo de pesquisa e fui encorajada a explorar novos horizontes, saindo da minha zona de conforto. Além de ser uma grande inspiração como professor, foi uma honra aprender e conviver com ele ao longo desse período.

A todos os meus amigos e colegas de faculdade, especialmente Felipe e João Pedro, que me estenderam a mão e contribuíram, de alguma forma, para que eu não desistisse em nenhuma etapa, mesmo nos momentos em que tudo parecia sem solução, meus sinceros agradecimentos.

RESUMO

O diff textual é amplamente utilizado para a identificação de mudanças entre diferentes versões de um mesmo programa ou arquivo. Apesar de sua popularidade e adoção em diversas ferramentas de desenvolvimento, algoritmos de diff textual possuem limitações que podem dificultar o processo de identificação de mudanças e integração de código. Como alternativa, o diff sintático utiliza o conhecimento da sintaxe da linguagem de programação para calcular as mudanças, comparando Árvores Sintáticas Abstratas (ASTs) em vez de conjuntos de *strings* que representam linhas de código.

Para investigar a percepção dos desenvolvedores sobre os resultados gerados por essas duas técnicas de *diff*, conduzimos uma pesquisa com 54 desenvolvedores, que foram convidados a comparar uma série de resultados gerados por essas ferramentas para os mesmos cenários (pares de arquivos representando duas versões de um mesmo arquivo). Os resultados indicam uma tendência à preferência pelo *diff* sintático em todos os cenários analisados, especialmente naqueles que envolvem movimentação de código, como quando um bloco é deslocado. Observa-se que, em mudanças mais complexas, o *diff* sintático tende a ser preferido em relação ao *diff* textual. Além disso, os desenvolvedores demonstram disposição para testar e incorporar ferramentas de *diff* sintático em seu fluxo de trabalho, com apenas 5,6% se opondo à sua adoção.

Palavras-chave: Diff, Gumtree, Git, Diff sintático, ASTs, Evolução de Software.

ABSTRACT

Textual diff is widely used for the identification of changes between different versions of the same program or file. Despite its popularity and adoption in various development tools, textual diff algorithms have limitations that can hinder the change identification and code merging process. As an alternative, structural diff leverages syntactic programming language knowledge to compute changes by comparing abstract syntax trees instead of strings representing program lines. To investigate developers' perceptions about the results generated by these two diff techniques, we survey 54 developers who are asked to compare a number of results these tools generate for the same scenarios (pair of files representing two versions of the same file). The results indicate a tendency to prefer structural diff in all analyzed scenarios, especially those involving code displacement, like when a block of code is moved. It is observed that, in more complex changes, structural diff tends to be preferable to textual diff. Additionally, developers show a willingness to test and incorporate structural diff tools into their workflow, with only 5.6% opposing their adoption.

Keywords: Diff, Gumtree, Git, Structural Diff, ASTs, Software Evolution.

Sumário

Lista de Tabelas

Tabela de Siglas

Sigla	Significado
GT	Gumtree
ASTs	Árvores Sintáticas Abstratas

1. Introdução

Imagine um cenário em que diversos profissionais realizam alterações e adicionam novas informações a múltiplos arquivos de forma independente. Essa é a realidade do desenvolvimento de software atual: um ambiente dinâmico que busca constantemente automatizar processos, aumentar a eficiência e assegurar o bom funcionamento dos sistemas. Nesse contexto, arquivos de software são modificados diariamente para atender a novas demandas e melhorar a eficácia dos sistemas. No entanto, essa frequência de mudanças torna desafiador manter o controle sobre quem realizou cada modificação, o que foi alterado e os motivos por trás dessas mudanças.

Para lidar com esse problema, sistemas de controle de versão, como o Git, passaram a ser amplamente utilizados para monitorar e gerenciar a evolução dos arquivos de um projeto. O Git[1] é um sistema de controle de versão distribuído extremamente popular, pois permite que cada desenvolvedor mantenha uma cópia completa do repositório localmente. Essa abordagem facilita o acompanhamento das mudanças, promove a colaboração eficiente e assegura que o histórico do projeto seja preservado de forma segura e acessível.

Nesse contexto, ao utilizar sistemas de controle de versão, existem técnicas específicas que auxiliam no acompanhamento das alterações realizadas nos arquivos. Um dos mais importantes é o *diff*, uma técnica que calcula e exibe as diferenças entre dois arquivos, sendo comumente utilizada para identificar as mudanças entre duas versões de um mesmo arquivo (NUGROHO et al., 2020). A sua execução gera um *script* de edição, que indica visualmente, na linha de comando, quais linhas foram adicionadas e removidas na versão atual do arquivo analisado.

O *diff* é amplamente utilizado e para realizar suas análises, ele faz uso de técnicas que são comumente conhecidas como *diffs* textuais. Um dos algoritmos mais conhecidos é o de Myers (MYERS, 1986), que é configurado como padrão no Git e, embora este algoritmo cumpra seu papel com boa performance, ele apresenta algumas limitações importantes.

Primeiramente, é importante lembrar que um arquivo código-fonte não é um simples texto, sem estruturas, ele segue uma série de regras sintáticas que variam de acordo com a linguagem de programação utilizada. Por isso, o *diff* textual possui uma análise limitada já que é baseada na comparação linha por linha, na qual o algoritmo não distingue elementos

[1] http://git-scm.com

irrelevantes, como caracteres de espaço em branco ou pontuações, que podem ser insignificantes no contexto da análise. Além disso, ele ignora completamente a sintaxe da linguagem de programação, o que pode comprometer a precisão na detecção de mudanças estruturais ou semânticas mais complexas. Outro ponto a se considerar é que os resultados são exibidos apenas na linha de comando, o que pode tornar o processo de análise menos intuitivo, especialmente quando se trata de alterações mais profundas e significativas no código. Desta forma, os resultados gerados podem dificultar o processo de interpretação dos desenvolvedores, ocasionando em um atraso na identificação de erros.

Para superar as limitações mencionadas anteriormente, novas técnicas foram desenvolvidas com a proposta de serem *syntax-aware*, ou seja, capazes de compreender as regras da linguagem de programação, estes são chamados de *diffs* sintáticos e buscam oferecer melhores *insights* aos seus usuários. Um exemplo de ferramenta que utiliza essa técnica é o GumTree (FALLERI et al., 2014), que aplica o conceito de Árvores Sintáticas Abstratas (ASTs) para representar a estrutura do código. Este processo de transformação ajuda a entender o que de fato está ocorrendo no código-fonte, baseando-se não apenas em agrupamentos de caracteres, mas em alterações reais na estrutura do arquivo. Além disso, a ferramenta consegue identificar um maior conjunto de ações de edição, se comparado às ferramentas que utilizam técnicas textuais, trazendo como resultado a comparação de arquivos através de uma interface gráfica.

Embora o Gumtree represente uma solução promissora para os problemas mencionados, ele pode apresentar um alto custo computacional. Por isso a inferência de evolução sintática é realizada com o auxílio de heurísticas, ou parâmetros, que precisam encontrar um equilíbrio entre a complexidade do tempo de execução e a qualidade dos resultados, como afirmado por Falleri; Martinez (2024).

Diversos estudos já foram realizados para entender um pouco mais sobre o funcionamento dos *diffs,* focando principalmente na sua performance (YANG; WHITEHEAD, 2019) (FALLERI; MARTINEZ, 2024), na qualidade dos resultados gerados (BARABUCCI et al., 2016) e na sua acurácia (FAN et al., 2021) (MARTINEZ; FALLERI; MONPERRUS, 2023). Porém, ainda não foram realizados estudos que foquem, principalmente, no impacto que estes diferentes resultados podem trazer ao dia a dia dos programadores, como por exemplo, se os resultados dessas técnicas dificultam a interpretação de mudanças, aumentam o tempo necessário para revisar código ou se podem contribuir para erros durante o processo de manutenção.

Por isso, propomos um estudo de investigação para entender como os resultados dessas ferramentas impactam o processo de desenvolvimento de software. É crucial analisar de que forma os resultados gerados influenciam a percepção dos desenvolvedores sobre as mudanças ocorridas, bem como avaliar seu papel em diferentes cenários e no suporte à evolução contínua do código.

Este documento está organizado da seguinte forma: o capítulo 2 apresenta a justificativa da pesquisa e sua motivação; capítulo 3 detalha os objetivos; capítulo 4 discute conceitos e o estado da arte; capítulo 5 descreve a metodologia do estudo; capítulo 6 apresenta os resultados obtidos; capítulo 7 traz uma análise e discussão dos resultados, e, por fim, o capítulo 8 apresenta as conclusões da pesquisa.

2. Justificativa

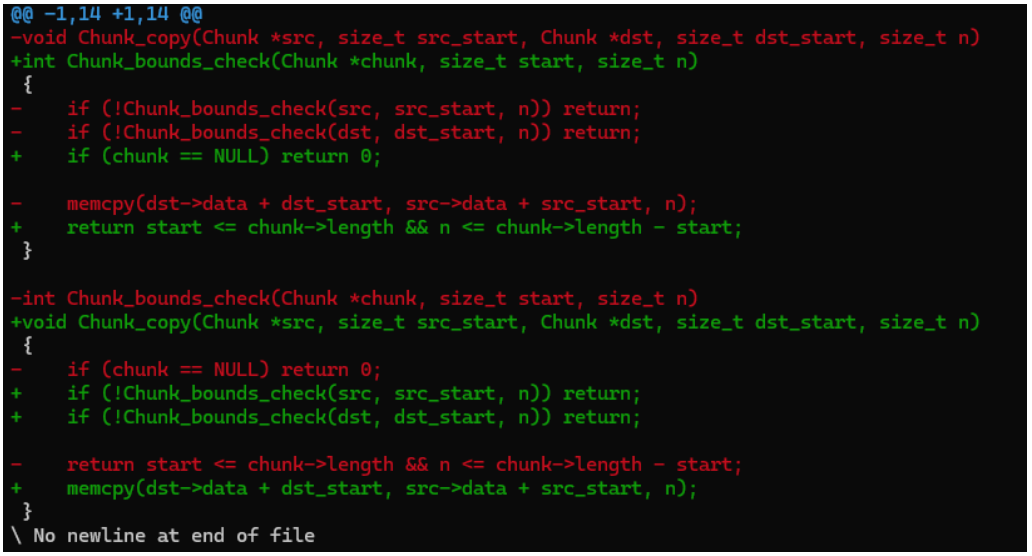
2.1 Cenário Motivador

Nosso cenário motivador baseia-se em um *toy code*, analisado no site[2] do autor Coglan, no qual um bloco de código, representado por um método, é movido. Essa movimentação faz com que a ordem dos métodos no arquivo mais antigo seja invertida na versão atual. Na Figura 1, é possível observar o *script* de edição gerado pelo **Git Diff** em resposta a essa mudança.

Inicialmente, analisamos que o resultado, por fazer uso do *diff* textual, indica uma série de adições e remoções de diversas linhas, além de destacar o conteúdo completo das linhas alteradas. No entanto, a técnica não identifica que, na realidade, todos os trechos de código já estavam presentes no arquivo e apenas foram reposicionados.As técnicas textuais são comumente utilizadas, mas possuem limitações.

Além disso, a visualização é prejudicada pelo fato de não utilizar uma interface gráfica, dependendo exclusivamente da linha de comando e não exibir os resultados lado a lado. Dessa forma, a mudança na ordenação do código torna-se menos intuitiva, dificultando a identificação rápida das alterações.

[2] https://blog.jcoglan.com/2017/03/22/myers-diff-in-linear-space-theory/



```
@@ -1,14 +1,14 @@
-void Chunk_copy(Chunk *src, size_t src_start, Chunk *dst, size_t dst_start, size_t n)
+int Chunk_bounds_check(Chunk *chunk, size_t start, size_t n)
 {
-    if (!Chunk_bounds_check(src, src_start, n)) return;
-    if (!Chunk_bounds_check(dst, dst_start, n)) return;
+    if (chunk == NULL) return 0;

-    memcpy(dst->data + dst_start, src->data + src_start, n);
+    return start <= chunk->length && n <= chunk->length - start;
 }

-int Chunk_bounds_check(Chunk *chunk, size_t start, size_t n)
+void Chunk_copy(Chunk *src, size_t src_start, Chunk *dst, size_t dst_start, size_t n)
 {
-    if (chunk == NULL) return 0;
+    if (!Chunk_bounds_check(src, src_start, n)) return;
+    if (!Chunk_bounds_check(dst, dst_start, n)) return;

-    return start <= chunk->length && n <= chunk->length - start;
+    memcpy(dst->data + dst_start, src->data + src_start, n);
 }
\ No newline at end of file
```

Figura 1. Script de edição gerado pelo Git Diff.

Agora, vamos analisar o resultado gerado pela ferramenta **GumTree (GT)**. A Figura 2 apresenta a interface obtida por meio da execução do comando *webdiff* do GT. O primeiro aspecto perceptível ao examinar a imagem é a diferença no esquema de cores adotado. Se analisarmos a Figura 1, resultado do Git Diff, observa-se que a principal diferença é que o GT consegue identificar não apenas ações de adição e remoção, mas também a movimentação de código, indicada pela cor roxa.

Observa-se também que, nesse caso, somente um trecho de código foi marcado como movido. Esse comportamento evidencia a natureza sintática do *diff*, demonstrando que a ferramenta compreende a estrutura do código, em vez de apenas comparar sequências de caracteres alterados. A sinalização gerada pelo GT reflete de maneira mais precisa o comportamento do desenvolvedor ao realizar a modificação.

Outra vantagem apresentada é que por meio da interface gráfica da ferramenta, a interpretação das mudanças torna-se mais ágil e intuitiva, permitindo uma compreensão mais clara das alterações realizadas entre as versões do arquivo.

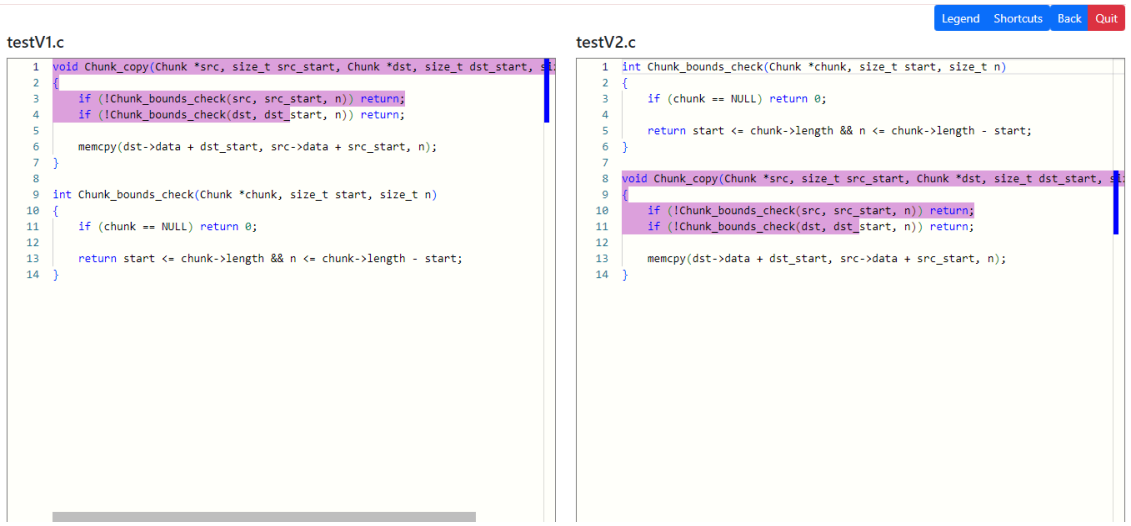


Figura 2. Resultado gerado pela ferramenta Gumtree.

2.2 Motivação

Os algoritmos de *diff* textual e sintático desempenham um papel fundamental na identificação de mudanças no código. Porém ainda não foi amplamente investigado como suas limitações afetam o fluxo de trabalho dos programadores, em quais cenários uma técnica pode ser mais eficaz do que a outra e de que maneira seu uso pode ter impactos positivos ou negativos. É crucial compreender por que os algoritmos de *diff* textual são mais amplamente adotados do que os de *diff* sintático, seja devido à falta de conhecimento sobre as alternativas disponíveis, seja porque, na prática, essas diferenças não impactam tanto a rotina dos programadores.

Identificar em quais cenários cada algoritmo é mais indicado é essencial, pois diferentes usuários possuem necessidades distintas. Dessa forma, os resultados gerados podem ser mais ou menos eficazes dependendo de quem os analisa. Segundo Barabucci et al. (2016), um programador que está desenvolvendo um código e outro que está navegando pelo histórico do código possuem necessidades diferentes. O primeiro precisa de mais detalhamento e de compreender o contexto em que cada mudança foi realizada, enquanto o segundo busca analisar as modificações de uma perspectiva mais ampla.

Dado o contexto apresentado e a importância crescente das empresas de software no mundo globalizado, melhorar o processo de identificação de mudanças no código não é apenas uma questão de eficiência interna, mas também um fator que impacta diretamente a qualidade e a velocidade de entrega de produtos ao mercado. Tornar a identificação de

mudanças o mais intuitiva possível é essencial, pois quanto menos tempo for gasto nesse processo, mais recursos podem ser direcionados para outras atividades cruciais.

Este estudo visa preencher essa lacuna, analisando como as ferramentas Git Diff e GumTree impactam a percepção das mudanças para os desenvolvedores e, consequentemente, impactam na produtividade e na qualidade do software das empresas.

Tendo em mente os desafios do acompanhamento da evolução de um software e as ferramentas disponíveis para a realização de diferenciação de arquivos, este trabalho busca responder à seguinte pergunta de pesquisa: *Qual o impacto do uso de diferentes ferramentas de diff na percepção de mudança para os programadores?*

3. Objetivos

Esta pesquisa visa analisar, a partir da percepção dos desenvolvedores, as diferenças nos resultados gerados pelas ferramentas GumTree e Git diff, explorando se suas aplicações influenciam significativamente na facilidade do entendimento sobre as alterações ocorridas ao longo do tempo no código-fonte.

Objetivos específicos:

- Identificar em que casos o uso de *diff* sintático é mais efetivo que o uso de *diff* textual;
- Analisar se para os desenvolvedores, faz diferença na prática o tipo de *diff* utilizado;
- Investigar se existe uma predisposição para a inserção de ferramentas de *diff* sintático no cenário empresarial.

4. Estado da Arte

Neste capítulo, apresentamos os principais conceitos e termos essenciais para o entendimento deste trabalho. Discutiremos Árvores Sintáticas Abstratas (AST), Diff Textual e Sintático, além da ferramenta GumTree, utilizada neste estudo. Por fim, exploramos os trabalhos já realizados na literatura relacionados a essa temática.

4.1 Árvores Sintáticas Abstratas (ASTs)

Uma Árvore Sintática Abstrata (AST) é uma estrutura utilizada para representar a organização de um código-fonte (Figura 3). Essa estrutura hierárquica, rotulada e enraizada é composta por nós interligados por arestas, onde cada nó pode ser classificado como "pai" ou "filho", e cada aresta representa uma relação pai-filho (FAN et al., 2021). O único nó sem um nó pai é denominado raiz da árvore, enquanto os nós que não possuem filhos são chamados de folhas. Cada nó na árvore corresponde a um elemento do código-fonte, em que seus rótulos são definidos de acordo com as regras da linguagem de programação. Os nós podem possuir valores que representam os *tokens* reais presentes no arquivo ou não.

A utilização de ASTs apresenta grande potencial para a diferenciação estruturada de arquivos. Em Chawathe et al. (1996), essa técnica foi explorada, garantindo que a estrutura sintática fosse considerada na análise de mudanças e que o algoritmo gerasse *scripts* de edição incluindo, como uma das ações de edição, a movimentação de elementos. Entretanto, o problema de encontrar a sequência de operações, incluindo a ação de movimentação, que transformam uma árvore T1 em uma árvore T2 é classificado como NP-difícil (BILLE, 2005), podendo ser custosa do ponto de vista computacional, prejudicando assim a performance das ferramentas.

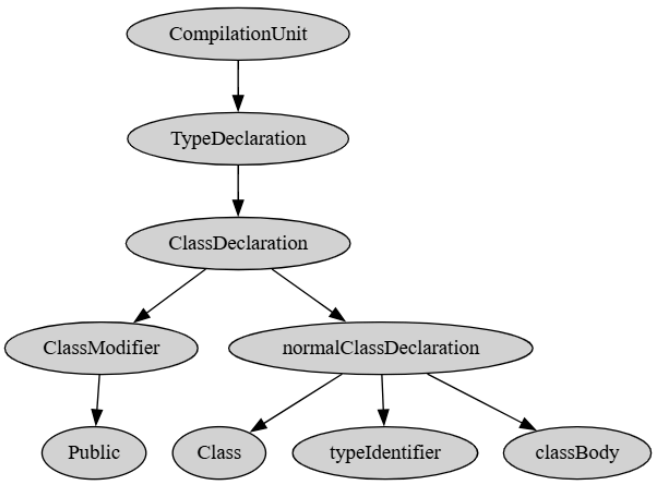


Figura 3. Exemplo de AST

4.2 Diff Textual

O *Diff* Textual ou a Diferenciação Baseada em Linha é uma técnica utilizada para analisar a diferença entre duas versões de um mesmo arquivo. Esta técnica representa o código como linhas textuais e tem como propósito encontrar a sequência mais curta de adições e deleções de blocos de texto que poderia transformar a versão original na versão modificada (FALLERI; MARTINEZ, 2024). De acordo com Nugroho, Hata e Matsumoto (2020), o *diff* textual é bastante disseminado no mercado devido a sua simplicidade e baixo tempo de execução, alguns exemplos de uso deste mecanismo são para obter patches, coletar métricas e identificar a introdução de *bugs*.

Existem diversos tipos de algoritmos de *diff* textual que podem gerar diferentes *scripts* de edição. Neste estudo, o algoritmo escolhido será o de Myers, por ser bastante disseminado no mercado e utilizado como padrão no Git, que oferece três opções além dele, o Minimal, o Patience e o Histogram (Nugroho; Hata; Matsumoto, 2020).

O algoritmo de Myers é considerado um algoritmo guloso e baseia-se na ideia de que a solução para encontrar a subsequência comum mais longa e a busca do menor script de edição para transformar um arquivo A em um arquivo B é equivalente a encontrar o caminho mais curto/longo em um grafo de edição (MYERS, 1986).

Diversas ferramentas utilizam essa técnica para garantir efetividade na comparação entre arquivos. Em Asaduzzaman et al., (2013), é proposta a ferramenta LHDiff, uma técnica híbrida que tem como objetivo rastrear mudanças em linhas de código, fazendo uso do Unix Diff e utilizando técnicas para identificar a similaridade entre conteúdo e contexto das linhas. Para validar a ferramenta, seus resultados são comparados com outras ferramentas disponíveis no mercado, baseando-se em critérios como performance e acurácia. No artigo de Canfora, Cerulo e Penta (2009) é proposta a outra ferramenta, chamada LDiff, que tem como objetivo superar as limitações encontradas no Unix Diff, aprimorando sua análise textual ao incorporar cálculos de similaridade e comparando com os resultados gerados exclusivamente pelo Unix.

A técnica de *diff* textual trata cada segmento de código como uma sequência de caracteres. Para cada conjunto de caracteres, o algoritmo busca correspondências (*matches*) no arquivo a ser comparado. Quando uma sequência de caracteres encontra um *match*, ela é utilizada como guia para identificar as alterações específicas entre os dois arquivos. Caso a sequência não encontre correspondência, ela é considerada uma remoção ou uma adição em relação à versão atualizada do arquivo. Esse processo é repetido iterativamente, analisando todo o conteúdo do arquivo, até que todas as diferenças entre as versões sejam mapeadas e reportadas.

4.3 Diff Sintático

A Diferenciação Sintática (*Diff* Sintático) refere-se a uma técnica alternativa para identificar as diferenças entre dois arquivos de código-fonte, na qual o algoritmo considera a sintaxe da linguagem durante a análise. Para isso, o arquivo é transformado, através de um *parser*, em uma Árvore Sintática Abstrata (AST). Essa nova estrutura representa uma grande vantagem para o processo, pois as expressões do código são estruturadas em subárvores, onde cada nível representa um elemento da linguagem de programação, organizando as operações e instruções de acordo com suas regras sintáticas. Por exemplo, ao adicionar um novo método a um arquivo, um novo nó é inserido na árvore, e cada etiqueta de nós da árvore corresponde a uma expressão específica da gramática da linguagem (FALLERI et al., 2014).

O *diff* sintático apresenta uma forma mais detalhada de análise e pode ser utilizado de diversas maneiras. Como, por exemplo, uma alternativa para analisar conflitos de merge (SIEFERT; SMITH; RIDGWAY, 2021), automatizar o processo de rastreamento de falhas através de uma técnica híbrida (PALIX; FALLERI; LAWALL, 2015) e identificar padrões de bug no Java (HANAM; BRITO; MESBAH, 2016).

Neste estudo, escolhemos o GumTree (FALLERI et al., 2014) como objeto de análise devido à sua capacidade de exibir resultados em uma interface gráfica, funcionar com diversas linguagens de programação, está disponível em repositório no GitHub e ser frequentemente atualizado, a versão mais recente foi publicada em 2024.

Existem diversas ferramentas no mercado que utilizam este tipo de técnica para realizar suas análises. O ChangeDistiller (FLURI et al., 2007) foi uma das primeiras ferramentas a adotar essa técnica, sendo inspirado no estudo de Chawathe et al. (1996), porém com aprimoramentos. Uma diferença significativa em relação ao GumTree, ferramenta analisada nesta pesquisa, é que o ChangeDistiller utiliza árvores mais simples e não trabalha com o mesmo nível de detalhamento na representação da AST.

O IJM (FRICK et al., 2018) é uma ferramenta inspirada na técnica utilizada no GumTree (GT), porém com objetivo de aprimorar a fase de mapeamento. Para avaliar a acurácia da ferramenta, o estudo comparou os resultados apresentados pela ferramenta proposta, o Gumtree e o MTDIFF (DOTZLER e PHILIPPSEN, 2016), baseando-se em quatro critérios, o tamanho do *script* de edição gerado, o tempo de execução, precisão da ação de edição e utilidade do *script* de edição.

O estudo realizado por Fan et al. (2021) avaliou e comparou, de forma automatizada, a acurácia de três ferramentas de mapeamento de ASTs, incluindo o GumTree (GT) e o IJM. A

abordagem foi aplicada a diversas revisões de arquivos, revelando que os algoritmos analisados produziram entre 20% e 36% de mapeamentos imprecisos. Esses resultados evidenciam a necessidade de aprimorar os algoritmos de mapeamento de ASTs, uma vez que seus resultados ainda não são totalmente confiáveis.

O estudo realizado por Decker et al. (2020) propõe uma ferramenta chamada srcDiff, uma técnica híbrida de *diffs*, que utiliza o srcML. O srcML é capaz de extrair tokens léxicos do código-fonte no formato XML, incorporando informações da AST, enquanto o algoritmo de Myers é utilizado para uma diferenciação de sequências mais eficiente. Para avaliar a ferramenta, o estudo comparou os resultados gerados com outras três ferramentas sintáticas e textuais: GumTree, a visualização do GitHub e o Mergely. Os critérios selecionados foram compreensibilidade, legibilidade, preferência pessoal e precisão, eles foram avaliados através de um survey. Além disso, o estudo realiza uma comparação de performance entre o srcDiff e o Gumtree.

4.4 O Gumtree

O Gumtree, proposto originalmente no estudo (FALLERI et al., 2014), é um algoritmo de diferenciação que tem como objetivo computar *scripts* de edição. Este algoritmo é considerado um *diff* sintático, que realiza suas análises baseado em Árvores Sintáticas Abstratas (ASTs) de granularidade detalhada. O GT considera como operações de edição, a adição, a remoção, a atualização e a movimentação, como mostrado na Figura 4. A grande vantagem deste algoritmo é que por trabalhar com a granularidade das ASTs, o *script* de edição alinha-se diretamente com a estrutura do código, diferentemente daqueles que realizam a diferenciação textual. Nas figuras 5 e 6 é possível observar a diferença dos resultados obtidos. Além disso, o GT pode ser utilizado por até dezesseis linguagens de programação, incluindo C, Java, PHP e JavaScript.

A ferramenta GumTree propõe um algoritmo para ser utilizado na fase em que é necessário buscar a correspondência entre nós, por meio da comparação de duas ASTs, identificando nós equivalentes entre ambas as estruturas. A técnica empregada para a criação do *script* de edição final é uma implementação já proposta por Chawathe et al. (1996).

É possível obter uma descrição detalhada do algoritmo GT mais atualizado, através do artigo de Falleri e Martinez (2024). O processo de correspondência de nós ocorre em três etapas principais: *Top-down*, *Bottom-up* e *Recovery*. Na primeira etapa, *Top-down*, a técnica

busca de maneira gulosa as maiores subárvores isomórficas, ou seja, aquelas que possuem a mesma estrutura em ambas as ASTs. Em seguida, a fase *Bottom-up* aproveita os mapeamentos estabelecidos na etapa anterior para expandi-los aos nós ancestrais. Por fim, a fase *Recovery* é executada sempre que um novo nó é adicionado durante a fase *Bottom-up*, permitindo a identificação e o mapeamento de descendentes que ainda não tenham sido mapeados. (FALLERI et al., 2014)

Apesar de gerarem *scripts* de edição que são corretos no sentido de transformar uma AST em outra, uma investigação manual revelou que esses scripts podem conter ações marcadas de forma imprecisa, especialmente em casos de movimentação e atualização, como evidenciado por Frick et al. (2018), demonstrando que, em certas situações, o *diff* sintático pode não ser tão preciso quanto o esperado.

Com o objetivo de tornar a ferramenta menos custosa computacionalmente, o GT utiliza de heurísticas para limitar o tamanho das ASTs analisadas. Entretanto, estas heurísticas podem gerar *scripts* de edição falsos e comprometer a acurácia da ferramenta, como citado anteriormente. Por isso, o estudo de Martinez, Falleri e Monperrus (2023) propõe um *Diff Auto Tuning* (DAT), que tem como objetivo automatizar o processo de definição de parâmetros, ajustando-os conforme o conjunto de arquivos analisados.

Visando aprimorar a ferramenta e consequentemente melhorar seu alto custo computacional, o estudo de Falleri e Martinez (2024) propõe uma nova versão para a fase final do algoritmo de *Recovery*, buscando eliminar o parâmetro utilizado anteriormente. Esta mudança proporcionou maior velocidade, gerando *scripts* de edição menores e mais compreensíveis em relação à versão anterior.

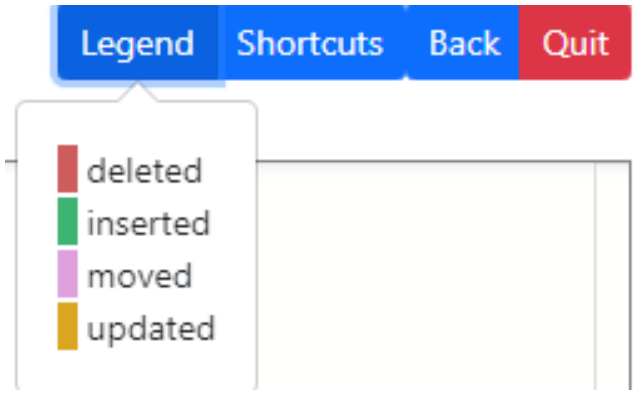


Figura 4. Legenda de cores da ferramenta GT para o *script* de edição.

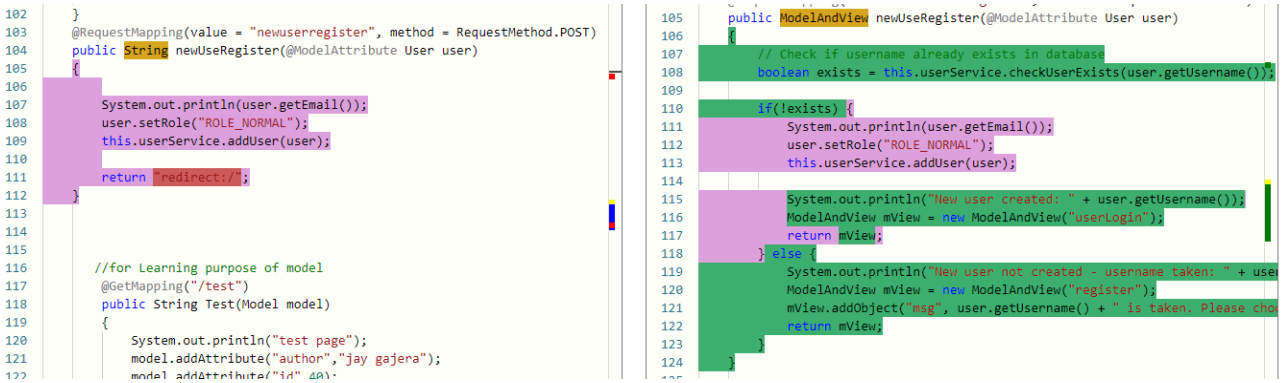


Figura 5. Resultado do GT do em cenário de exemplificação.

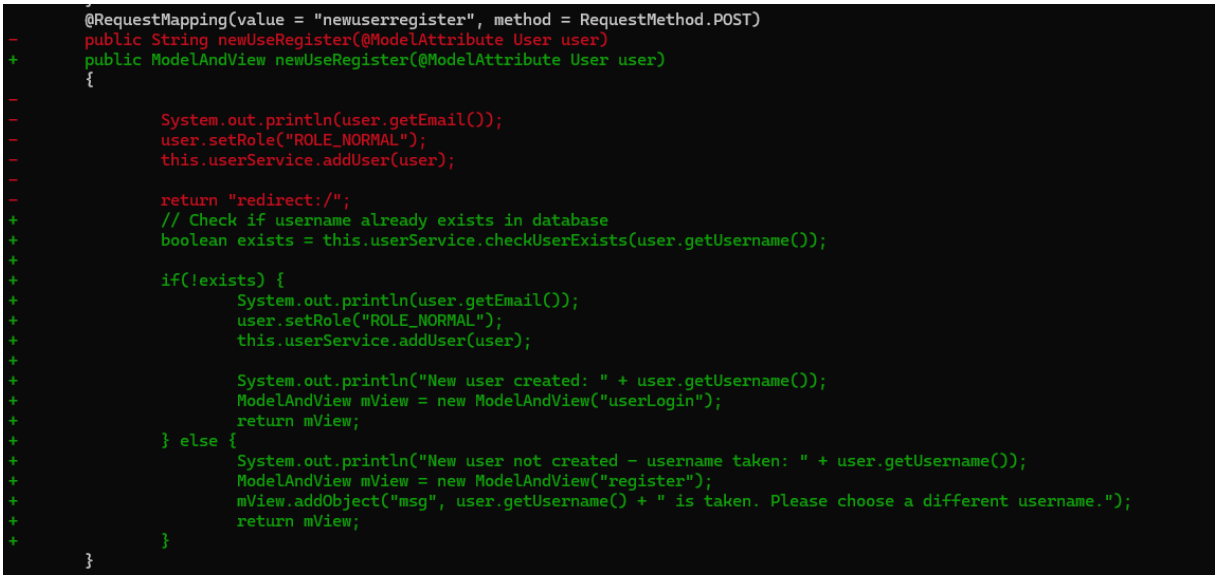


Figura 6. Resultado do Git diff do em cenário de exemplificação.

5. Metodologia

Foi utilizada uma abordagem multimétodos combinando técnicas quantitativas e qualitativas com o objetivo de responder às perguntas de pesquisa delimitadas anteriormente. O estudo envolve inicialmente uma pesquisa pesquisa bibliográfica, em seguida foram realizados testes com as ferramentas alvo, e finalmente uma pesquisa do tipo survey com a aplicação de um questionário para analisar a opinião dos desenvolvedores sobre a temática. Antes de sua divulgação, o questionário foi submetido e aprovado no Comitê de Ética em Pesquisa - CEP, CAAE 84860524.2.0000.5208.

Hoje, a palavra "survey" é usada com mais frequência para descrever um método de coleta de informações de uma amostra de indivíduos. Essa amostra geralmente é apenas uma fração da população que está sendo estudada (SCHEUREN, 2004).

5.1 Pesquisa Bibliográfica

O processo de pesquisa iniciou-se a partir de indicações de trabalhos já conhecidos pelo grupo de pesquisa *Software Productivity Group* (SPG) do Centro de Informática da UFPE. A partir destas indicações, as ferramentas foram estudadas e selecionadas como objeto de estudo para este trabalho.

O processo de pesquisa foi iniciado com a utilização de palavras-chave, como *"Gumtree", "diff", "Git", "line-based diff"*, que são diretamente relacionadas à temática de investigação, o tempo de publicação das pesquisas selecionadas não foi delimitado. Utilizando os mecanismos de busca Google Scholar, IEEE e Scorpus, foi possível obter informações sobre como estas ferramentas e a temática já haviam sido estudadas. Com esse entendimento, foi possível elaborar um referencial teórico sólido, que sustenta o aprofundamento da pesquisa e contribui para responder aos objetivos propostos.

5.2 Exploração das ferramentas

Para compreender o funcionamento das ferramentas e a forma como geram seus resultados, foram explorados cenários simples de mudanças em arquivos de código. Para exemplificação, optou-se pelas linguagens de programação C e Java, considerando sua ampla utilização em sistemas e aplicações contemporâneas. Os cenários utilizados como exemplos foram extraídos de um toy code utilizado no site[3] do autor Coglan e de dois repositórios de código aberto disponíveis no GitHub, E-commerce-project-springBoot[4] e onlinebookstore[5].

Para a diferenciação entre arquivos no *diff* textual, foi utilizado o comando *git diff* no formato: git diff -w [código hash do commit comparado1] [código hash do commit comparado2] ou [código hash do commit de merge]. A opção -w instrui o Git a ignorar

[3] https://blog.jcoglan.com/2017/03/22/myers-diff-in-linear-space-theory/
[4] https://github.com/jaygajera17/E-commerce-project-springBoot
[5] https://github.com/shashirajraja/onlinebookstore

diferenças em espaços em branco, focando apenas nas alterações sem interferências de formatação.

Para a utilização da ferramenta GumTree como *diff tool* no Github, é necessário, primeiramente, integrá-la ao repositório. Esse processo envolve a edição do arquivo de configuração do Git (*git config*), adicionando a ferramenta à configuração de *diff tool*. Após essa configuração, a diferenciação pode ser realizada utilizando o seguinte comando, especificando o caminho do repositório: C:\gumTree\example\nome_do_repositório> git difftool -d [código hash do commit comparado1] [código hash do commit comparado2] ou [código hash do commit de merge].

Os resultados gerados pelas ferramentas de teste serviram de base para a construção do questionário aplicado. Eles foram utilizados como exemplos para que os participantes selecionassem suas preferências.

5.3 Elaboração do Questionário

O questionário tem como objetivo analisar a percepção dos participantes sobre os resultados gerados pelas técnicas de diff textual e sintático. Por meio dele, buscamos compreender como as diferentes visualizações podem impactar e influenciar o processo de identificação de mudanças e revisão de código.

O questionário foi estruturado em três etapas.

- A primeira etapa contém três perguntas fechadas, cujo objetivo é compreender a experiência e as habilidades dos participantes.
- Na segunda etapa, são apresentados quatro cenários de mudanças em arquivos de software, seguidos de quatro perguntas relacionadas aos cenários e às preferências dos participantes.Os cenários foram baseados em ações simples e frequentes no cotidiano dos programadores. Foram escolhidos os seguintes casos: movimentação de código (Cenário 1), modificação em declarações de *import* (Cenário 2), alterações internas em um método (Cenário 3) e mudanças em estruturas condicionais (Cenário 4).
- A terceira e última etapa avalia as conclusões dos participantes após completarem o questionário contemplando duas perguntas: uma aberta, visando captar a opinião geral, e outra fechada para entender se o participante estaria disposto a utilizar as ferramentas em questão no futuro.

A escolha dos cenários utilizados no questionário foi realizada com o objetivo de abranger mudanças comuns no dia a dia de arquivos código-fonte. Foram selecionadas modificações que explorassem ao máximo os recursos das ferramentas, com destaque para cenários que envolvessem movimentação e atualização de valores, já que esse é um dos principais diferenciais da técnica sintática, de forma que seus resultados pudessem impactar visualmente os participantes. Além disso, para contemplar diferentes níveis de conhecimento, buscamos por cenários simples, garantindo que qualquer pessoa com experiência em código pudesse compreendê-los.

O questionário foi elaborado e aplicado na plataforma Google Forms e pode ser consultado no Apêndice A. Antes de sua aplicação foi realizado um teste piloto com estudantes para coletar feedback e avaliar o tempo necessário para completá-lo. Com base nesses feedbacks foram feitos os ajustes necessários para aprimorá-lo.

Importante salientar que aspectos ligados ao custo computacional e funcionamento das ferramentas não foram testados com os participantes do survey, já que o objetivo principal da pesquisa é analisar o impacto da interpretação de diferentes *scripts* de edição. O processo de utilização e teste de performance da ferramenta não faz parte do escopo do projeto.

5.4 Aplicação e análise do questionário

O público-alvo foi composto por desenvolvedores de software, com mais de 18 anos e com mais de 6 meses de experiência no mercado. Esta escolha baseou-se na amostra de pessoas que podem se beneficiar com a utilização das ferramentas de diferenciação em sua rotina de trabalho.

O questionário foi divulgado na rede social Linkedin e compartilhado no email institucional do Centro de Informática da UFPE no período de 03 de Janeiro de 2025 a 03 de fevereiro de 2025. Foram estimados 100 participantes, considerando a viabilidade da coleta de dados dentro do prazo estabelecido de um mês. Os dados coletados foram analisados de forma quantitativa e qualitativa. As perguntas abertas serviram como referência para compreender o sentimento dos participantes em relação às ferramentas e complementar a análise.

6. Resultados

Este capítulo tem como objetivo avaliar os dados obtidos através da divulgação do questionário. Serão discutidas principais tendências observadas e suas possíveis implicações a temática. Os testes realizados anteriormente com as ferramentas serviram como base para apoiar os exemplos utilizados no questionário. Os resultados são apresentados de forma a responder às questões de pesquisa levantadas anteriormente, permitindo uma melhor compreensão dos impactos das ferramentas de diferenciação de código.

O survey foi respondido por 54 pessoas no período de Janeiro de 2025 a Fevereiro de 2025 das quais 57,4% possuíam pelo menos cinco anos de experiência em desenvolvimento de software. 27,8% possuíam de dois a cinco anos de experiência e 14,8% possuíam de 6 meses a um ano de experiência.

Para avaliar o impacto das ferramentas de *diff* na rotina dos desenvolvedores, foi essencial identificar o nível de conhecimento dos participantes em relação às técnicas de *diff* textual e sintático. Os dados coletados revelaram que 44,4% dos participantes nunca haviam ouvido falar sobre o conceito "*diff* sintático", enquanto 25,9% conheciam o conceito, mas nunca haviam utilizado tais ferramentas e 29,6% já haviam utilizado ferramentas que utilizam a técnica sintática. Esses resultados indicam que a maioria dos participantes não possui familiaridade prévia com essa técnica, o que pode, inclusive, justificar sua baixa adoção.

Cenários	Cenário 1	Cenário 2	Cenário 3	Cenário 4
Sintático	74,1%	63,0%	64,8%	53,7%
Textual	16,7%	16,7%	25,9%	22,2%
Equivalentes	9,3%	20,4%	9,3%	24,1%

Tabela 1 - Preferência entre os resultados gerados.

Cenários	Cenário 1	Cenário 2	Cenário 3	Cenário 4
Sintático	9,3%	5,6%	20,4%	13,0%
Textual	33,3%	13,0%	25,9%	11,1%
Ambas claras	51,9%	79,6%	50,0%	75,9%

Tabela 2 - Dificuldade de compreensão dos deltas gerados.

Um dos principais objetivos neste trabalho é identificar em quais cenários o uso de *diff* sintático pode ser mais efetivo do que o uso de *diff* textual. Para isso, foram utilizados quatro cenários simples, que podem ser evidenciados no Apêndice A, em que acredita-se serem

recorrentes no dia a dia dos desenvolvedores. Podemos observar, por meio da Tabela 1, que a preferência pelo resultado gerado pela técnica sintática, em vez da textual, é significativamente maior em todos os casos. Em muitos cenários, essa diferença é, inclusive, pelo menos duas vezes maior.

Dentre esses cenários, o que apresentou o impacto mais evidente foi o Cenário 1 (movimentação de bloco de código), seus resultados podem ser analisados na Figura 1 e 2, com uma taxa de preferência de 74,1% em relação ao *diff* textual. É importante destacar que, como o *diff* textual exibe apenas ações de adição e remoção, muitos participantes da pesquisa consideraram que, neste caso, os resultados exibidos por esta técnica ficaram confusos, como afirma o participante P31, ao justificar sua preferência pelo resultado sintático: "No primeiro cenário, existem trechos de código que apenas foram movidos de posição, mas são marcados como novas adições/remoções. Já no segundo cenário, fica evidente que o trecho não foi deletado ou adicionado, apenas movido".

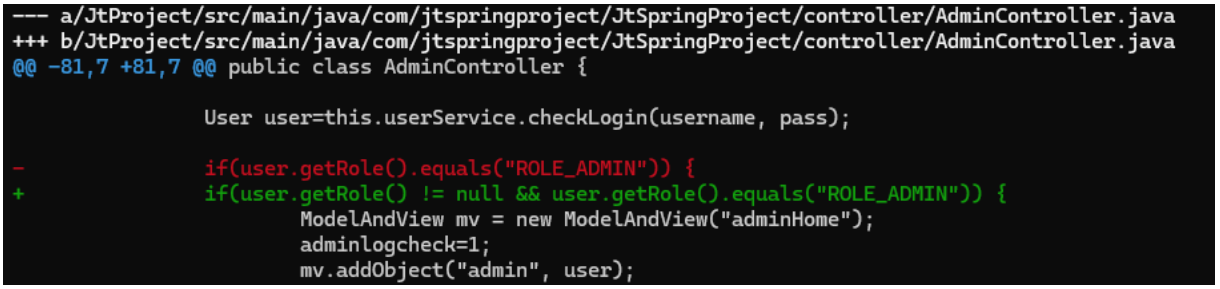
Como indicado na Tabela 2, cerca de 33,3% dos participantes afirmaram que a imagem do *diff* textual era difícil de interpretar, dificultando a compreensão da ação que estava ocorrendo. Diversos participantes sinalizaram em suas respostas abertas a importância da capacidade de identificar partes do código que permanecem inalteradas, como feito pelo Gumtree (ver Seção 4.4), o participante P23 afirma que: "Acredito que uma das principais ocorrências do fenômeno apresentado seja para adequar um comportamento existente a um novo contexto. Portanto, acredito que o resultado 5 (sintático) seja mais claro ao demonstrar com a cor roxa que uma dada sequência de instruções já estava presente, mas agora sob novo contexto. O resultado 6 não deixa isso evidente. Em uma rápida análise, leva a entender que determinado comportamento fora substituído por um novo, sem necessariamente ter relação com o anterior". Esta habilidade pode ser essencial para a detecção de bugs, refatorações e apoiar na análise de pull requests.

O cenário com a menor diferença em preferência foi o Cenário 4 (alteração de um condicional), que pode ser analisado através das Figuras 7 e 8, no qual 53,7% dos participantes optaram pelo *diff* sintático, enquanto 22,2% consideraram o resultado do textual superior.

```
if(user.getRole().equals("ROLE_ADMIN")) {
    ModelAndView mv = new ModelAndView("adminHome");
    adminlogcheck=1;
    mv.addObject("admin", user);
    return mv;
}
else {
    ModelAndView mv = new ModelAndView("adminlogin");
    mv.addObject("msg", "Please enter correct username and password");
    return mv;
}
```

```
if(user.getRole() != null && user.getRole().equals("ROLE_ADMIN")) {
    ModelAndView mv = new ModelAndView("adminHome");
    adminlogcheck=1;
    mv.addObject("admin", user);
    return mv;
}
else {
    ModelAndView mv = new ModelAndView("adminlogin");
    mv.addObject("msg", "Please enter correct username and password");
    return mv;
}
```

Figura 7. Resultado do GT do Cenário 4 utilizado no questionário.



```
--- a/JtProject/src/main/java/com/jtspringproject/JtSpringProject/controller/AdminController.java
+++ b/JtProject/src/main/java/com/jtspringproject/JtSpringProject/controller/AdminController.java
@@ -81,7 +81,7 @@ public class AdminController {

                User user=this.userService.checkLogin(username, pass);

-               if(user.getRole().equals("ROLE_ADMIN")) {
+               if(user.getRole() != null && user.getRole().equals("ROLE_ADMIN")) {
                        ModelAndView mv = new ModelAndView("adminHome");
                        adminlogcheck=1;
                        mv.addObject("admin", user);
```

Figura 8. Resultado do Git diff do Cenário 4 utilizado no questionário.

Ao compararmos os resultados dos Cenários 1 e 4 na Tabela 1, aqueles com as diferenças mais marcantes de preferência, podemos identificar uma similaridade em relação às mudanças envolvidas, ambos sinalizam a ação de movimentação de código em diferentes aspectos. Mas o que de fato faz com que a escolha pelo sintático no Cenário 1 seja maior do que no Cenário 4 é, acreditamos, que a expressividade da movimentação do código. Se analisarmos o Cenário 4, a movimentação ocorre de forma muito sutil, em apenas uma linha do código, com a adição de um novo condicional. Já no Cenário 1, existe uma realocação de todo um bloco de código, de forma muito mais expressiva, o que faz com que o resultado sintático e a identificação da ação de movimentação seja muito mais relevante para aqueles que estão analisando o código.

Além da diferença na preferência pelo *diff* sintático, também observamos um aumento de 1,5 no número de participantes que consideraram os resultados do Cenário 4 equivalentes, em comparação com o Cenário 1. Mais uma vez, isso reforça que movimentações menos expressivas têm um impacto menor na visualização do *script* de edição.

Um ponto interessante levantado nas respostas abertas foi a poluição visual que o diff textual pode causar nos cenários avaliados. Ficou evidente que, muitas vezes, essa técnica marca adições e remoções do mesmo trecho de código, o que pode dificultar a compreensão das mudanças. Em contraste, o GumTree proporciona uma visualização objetiva e intuitiva das reais alterações realizadas. Os participantes destacaram que a técnica sintática apresenta com mais clareza as adições e remoções efetivas do código, permitindo compreender que determinado trecho já existia e apenas foi realocado para um novo contexto.

Outro ponto a se observar na Tabela 2 é que no Cenário 4 (alteração de um condicional) e no Cenário 2 (mudança de *import*), que pode ser visualizado nas Figuras 9 e 10, a maioria dos participantes concordou que ambos os resultados eram claros. No Cenário 4,

75,9% dos participantes afirmaram que ambas imagens eram claras de entender e a ação bem representada, enquanto 79,6% concordaram que a ação no cenário de mudança de *import* estava igualmente clara. Isso sugere que, em mudanças mais simples, a forma de visualização tem um impacto menor, pois, nesses casos, a análise se concentra em uma única linha modificada e a comparação se torna clara, independente do delta utilizado.

Uma característica bastante interessante de ser observada, que foi reforçada principalmente no Cenário 2 (Figuras 9 e 10), é a marcação do trecho que foi alterado e não de uma linha inteira. Diversos participantes sinalizaram esta perspectiva de visualização sendo a mais relevante neste cenário, exemplificando que em mudanças mais sutis, em que por exemplo, classes tivessem nomes semelhantes, o *diff* sintático seria mais eficiente. Vale ressaltar que essa capacidade de identificar mudanças apenas em partes da linha ocorre devido à comparação entre ASTs.

```
index a4d527d..e4350ea 100644
--- a/src/main/java/com/bittercode/model/StoreException.java
+++ b/src/main/java/com/bittercode/model/StoreException.java
@@ -2,7 +2,7 @@ package com.bittercode.model;

 import java.io.IOException;

-import com.bittercode.constant.ErrorCodes;
+import com.bittercode.constant.ResponseCode;

 public class StoreException extends IOException {

@@ -17,7 +17,7 @@ public class StoreException extends IOException {
         this.errorMessage = errorMessage;
     }

-    public StoreException(ErrorCodes errorCodes) {
+    public StoreException(ResponseCode errorCodes) {
         super(errorCodes.getMessage());
         this.statusCode = errorCodes.getCode();
         this.errorMessage = errorCodes.getMessage();
```

Figura 9. Resultado do Git diff do Cenário 2 utilizado no questionário.

```
4
5  import com.bittercode.constant.ErrorCodes;
6
7  public class StoreException extends IOException {
8
9      private String errorCode;
10     private String errorMessage;
11     private int statusCode;
12
13     public StoreException(String errorMessage) {
14         super(errorMessage);
15         this.errorCode = "BAD_REQUEST";
16         this.setStatusCode(400);
17         this.errorMessage = errorMessage;
18     }
19
20     public StoreException(ErrorCodes errorCodes) {
21         super(errorCodes.getMessage());
22         this.statusCode = errorCodes.getCode();
23         this.errorMessage = errorCodes.getMessage();
24         this.setErrorCode(errorCodes.name());
25     }
```

```
4
5  import com.bittercode.constant.ResponseCode;
6
7  public class StoreException extends IOException {
8
9      private String errorCode;
10     private String errorMessage;
11     private int statusCode;
12
13     public StoreException(String errorMessage) {
14         super(errorMessage);
15         this.errorCode = "BAD_REQUEST";
16         this.setStatusCode(400);
17         this.errorMessage = errorMessage;
18     }
19
20     public StoreException(ResponseCode errorCodes) {
21         super(errorCodes.getMessage());
22         this.statusCode = errorCodes.getCode();
23         this.errorMessage = errorCodes.getMessage();
24         this.setErrorCode(errorCodes.name());
25     }
```

Figura 10. Resultado do GT do Cenário 2 utilizado no questionário.

É importante destacar que, apesar de no Cenário 4, a porcentagem de participantes que afirmam que o resultado do diff sintático é mais confuso do que o textual ser maior como observamos na Tabela 2, ao analisarmos os números no gráfico exibido na Figura 11, percebemos que a diferença entre as duas escolhas é de apenas uma pessoa, o que, na prática, pode ser considerado um empate.

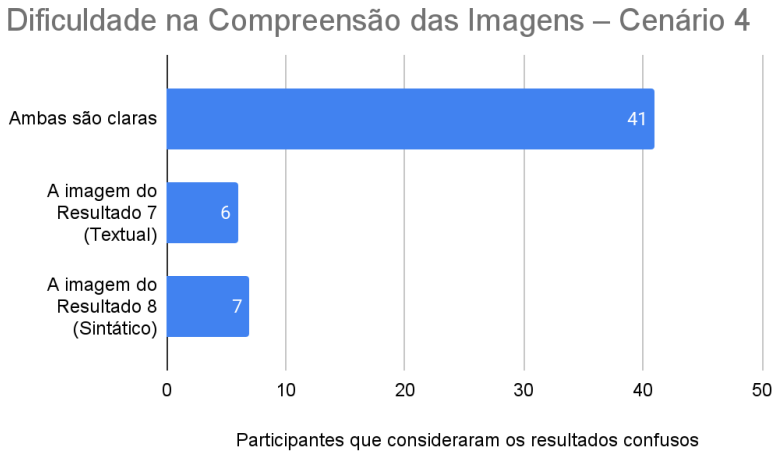


Figura 11 - Gráfico da quantidade de participantes que consideraram os resultados do Cenário 4 confusos.

Ao observar a Tabela 2, percebe-se que no Cenário 3 (mudanças internas em um método) (Figuras 5 e 6) a diferença na porcentagem de participantes que consideram as imagens do *diff* sintático e do *diff* textual confusas é maior do que nos Cenários 4 e 2. Quando observamos os números (ver Figura 12), percebemos que a diferença entre as escolhas corresponde a apenas três pessoas, o que não representa uma quantidade tão expressiva. É curioso observar que esse cenário apresenta a maior quantidade de ações de mudança e uso de cores para sinalização. A partir das respostas abertas e analisando o cenário em questão, podemos interpretar que a principal questão para muitos participantes, especialmente aqueles acostumados com o *diff* textual, é que a grande quantidade de cores torna a visualização mais poluída. No entanto, essa percepção não é unânime, pois outros participantes afirmam que a ênfase visual e o destaque nos trechos de código já existentes tornam o processo mais eficiente. Por isso, existe uma proximidade maior entre estas duas porcentagens.

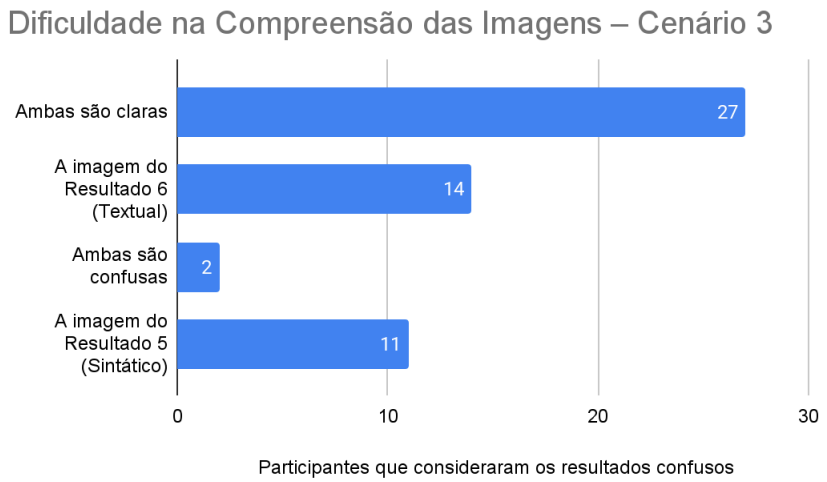


Figura 12 - Gráfico da quantidade de participantes que consideraram os resultados do Cenário 3 confusos.

Também buscamos analisar se os cenários exemplificados são realmente comuns na rotina dos desenvolvedores. Os resultados indicaram que, de maneira geral, as ações descritas ocorrem com frequência no ambiente de trabalho: 81,5% dos participantes afirmaram que a movimentação de código acontece regularmente (Figura 13), 87% mudanças no *import* (Figura 14), 94,4% alterações em métodos internos (Figura 15), e 94,4% a alteração de condicionais (Figura 16) como tarefas recorrentes em seu dia a dia.

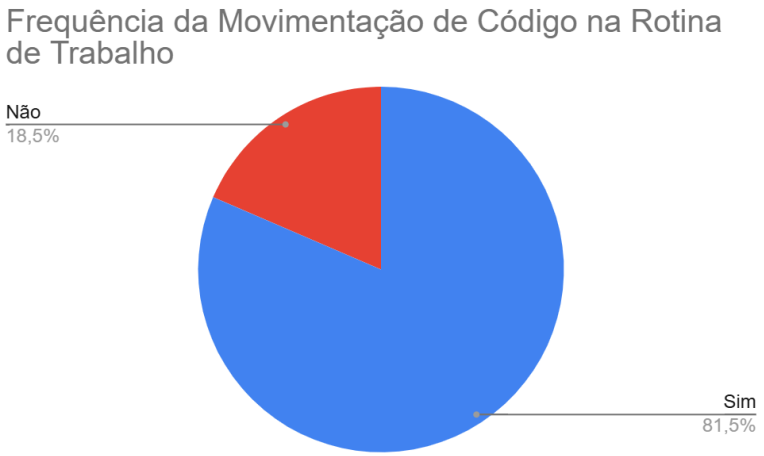


Figura 13 - Gráfico da porcentagem que acredita que o Cenário 1 é recorrente na sua rotina.

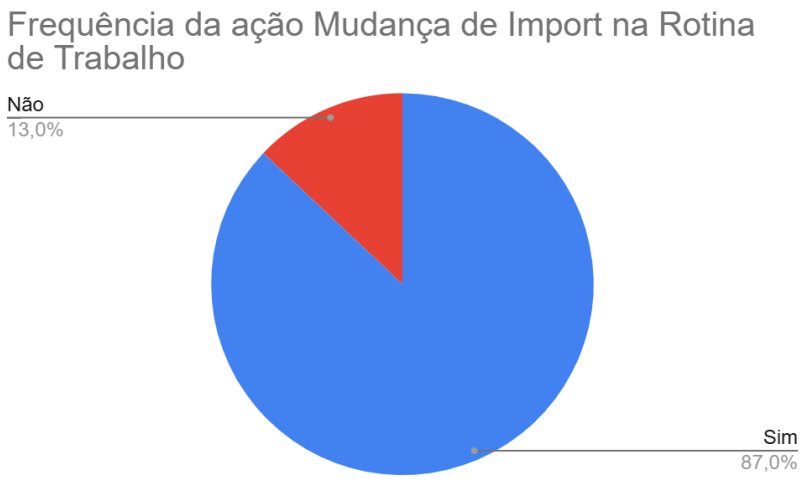


Figura 14 - Gráfico da porcentagem que acredita que o Cenário 2 é recorrente na sua rotina.

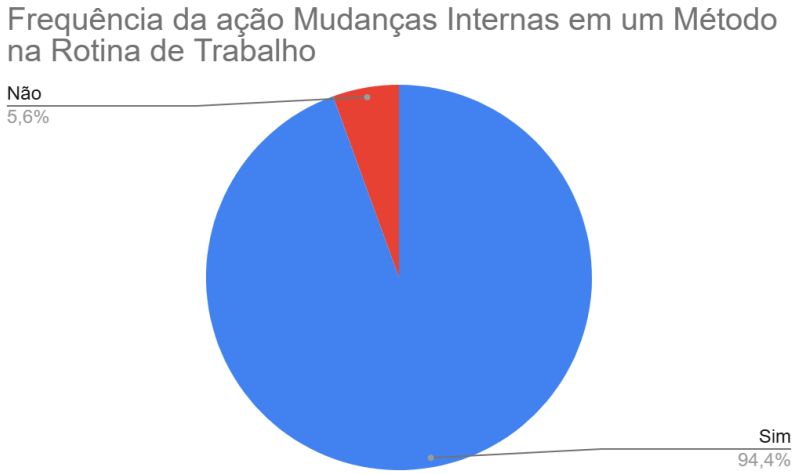


Figura 15 - Gráfico da porcentagem que acredita que o Cenário 3 é recorrente na sua rotina.

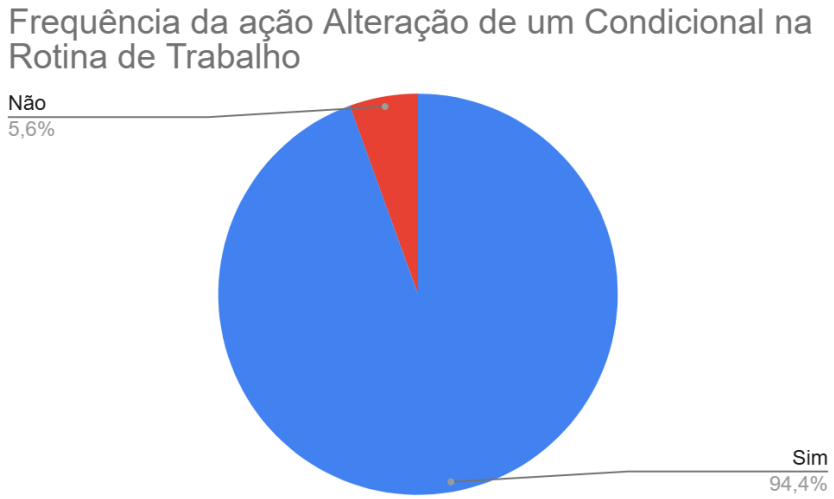


Figura 16 - Gráfico da porcentagem que acredita que o Cenário 4 é recorrente na sua rotina.

Para finalizar, investigamos a predisposição dos desenvolvedores em adotar ferramentas de diff sintático em sua rotina de trabalho, e apenas 5,6% dos participantes se mostraram contrários a essa adoção, como podemos observar na Figura 17. Com isso, podemos concluir que, embora exista certa resistência a mudanças nos métodos de trabalho, a maioria dos participantes estaria disposta a testar e utilizar essas novas ferramentas.

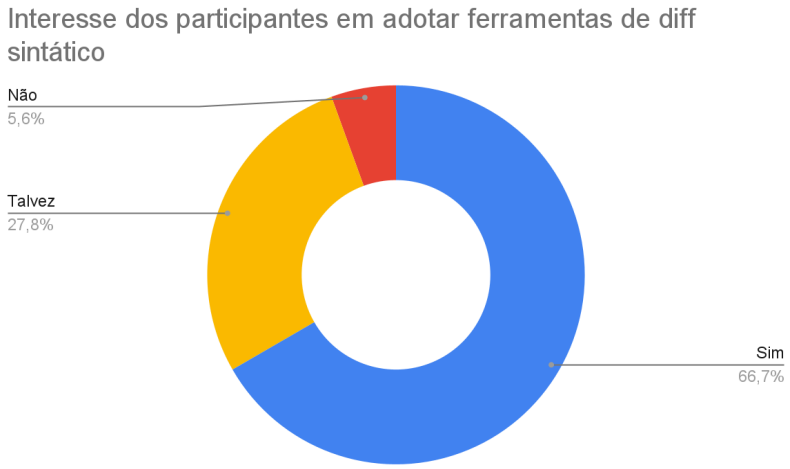


Figura 17 - Gráfico da porcentagem dos participantes que têm interesse em adotar ferramentas de diff sintático.

7. Discussão

Nesta seção apresentamos as implicações identificadas a partir dos resultados obtidos e as ameaças à validade do estudo.

7.1 Implicações

O *diff* é um mecanismo essencial para diversos tipos de usuários, nesta pesquisa, o objetivo foi dar foco aos programadores e como essas ferramentas podem influenciar no processo de desenvolvimento. Através do estudo, foi possível entender, primeiramente, que existe um desconhecimento geral por parte dos desenvolvedores sobre a existência de ferramentas que utilizam técnicas sintáticas para computar o *diff*.

Outro ponto relevante é que o cenário e o perfil do usuário influenciam diretamente na escolha entre *diff* sintático e textual. Porém, apesar de utilizarmos cenários mais simples do que os encontrados na rotina real dos programadores, mesmo quando a dificuldade de

compreensão não é tão alta, a maioria das pessoas no nosso estudo ainda assim opta pelo *diff* sintático.

Ao avaliar em quais cenários a escolha da ferramenta tem maior relevância, observa-se que, em situações que envolvem mudanças mais simples e sutis ou quando o programador não necessita de um contexto detalhado, o *diff* sintático pode não agregar muito valor aos interpretadores. Então, acredita-se que em projetos mais complexos, com equipes grandes, muitas integrações e mudanças, a tendência seria utilizar a técnica sintática, para garantir resultados mais intuitivos e detalhados. Já em projetos mais simples, no qual a complexidade das mudanças não são tão extensas, o textual seria o suficiente.

A partir dos resultados, pode-se afirmar que a ferramenta GumTree possui grande valor e potencial no mercado, mas é improvável que substitua completamente o *diff* textual. Ela pode ser especialmente útil para quem está iniciando na área e precisa de um detalhamento maior para entender a evolução do código, assim como para aqueles que revisam frequentemente mudanças para integração e necessitam de uma análise mais aprofundada. No entanto, programadores mais experientes podem não dar valor em utilizá-la no dia a dia, devido à necessidade de integração e à interface diferente da qual estão acostumados. O alto custo computacional é outro ponto a ser debatido já que a ferramenta GT realiza uma série de processos que a técnica textual não necessita realizar e se comparado com outras ferramentas sintáticas, como o IJM, o *runtime* do processo de correspondência dos nós é maior em 60% dos casos (FRICK et al., 2018).

O Git Diff segue a mesma lógica do GumTree, não sendo ideal para todos os casos, mas podendo se encaixar de forma eficaz em diversos cenários. Por estar integrado ao Git, enfrenta menos barreiras de adoção, porém os usuários acabam perdendo muita informação sobre as mudanças ocorridas. Um dos aspectos mais discutidos sobre a relevância do *diff* sintático foi sua capacidade de identificar trechos de código movidos sem alterações, o que representa um diferencial significativo em relação ao *diff* textual. Como a técnica textual sinaliza apenas adições e remoções, o contexto das mudanças é comprometido e a compreensão da modificação real é prejudicada.

Um ponto importante a destacar são as ferramentas intermediárias que utilizam ASTs e técnicas textuais para realizar a análise, como a interface gráfica do GitHub[6]. Ela pode ser considerada um meio-termo entre as ferramentas estudadas, pois consegue analisar trechos léxicos do código, permitindo marcar partes específicas das linhas. Embora ainda se limite à análise de ações de adição e remoção.

[6] https://github.com

Este estudo mostrou uma forte tendência entre os desenvolvedores participantes de estarem receptivos para novas ferramentas que ofereçam mais contexto sobre o processo de evolução do código. Observou-se que mais de 90% dos participantes da pesquisa demonstraram interesse em testar a ferramenta, o que evidencia sua relevância e potencial de adoção no ambiente de desenvolvimento.

7.2 Ameaças à validade

Uma ameaça à validade deste estudo é a limitação do escopo da pesquisa, que se concentrou exclusivamente na percepção dos desenvolvedores sobre os resultados gerados pela ferramenta. Assim, questões relacionadas ao custo computacional e ao desempenho da ferramenta no uso cotidiano, apontadas como problemáticas na Seção 1, não foram abordadas. Consequentemente, não é possível avaliar como esses fatores influenciam a adoção dessas ferramentas na rotina dos programadores.

Outra ameaça à validade do estudo está na simplicidade e no caráter estático dos cenários utilizados como exemplos. Esses cenários não refletem com precisão a complexidade do ambiente real encontrado na rotina de empresas de software. Como consequência, a percepção dos desenvolvedores sobre os resultados gerados pode ter sido impactada, uma vez que eles estão habituados a lidar com mudanças mais complexas e dinâmicas no código.

Além disso, como o questionário foi aplicado a um público com diferentes níveis de experiência, os exemplos foram intencionalmente simplificados para garantir que todos os participantes pudessem compreendê-los. Isso, no entanto, resultou em uma abordagem mais básica do funcionamento da ferramenta.

8. Conclusão

Ao analisar os resultados obtidos através desta pesquisa, podemos concluir que, primeiramente, existe uma familiaridade muito limitada dos participantes sobre as ferramentas de *diff* sintático. Através do questionário, foi possível observar que grande parte dos participantes nunca haviam tido contato com essas ferramentas anteriormente, justificando a sua ausência na rotina. Mas há uma forte inclinação em adotar e testar essas técnicas em seus projetos.

Os dados da pesquisa exibem uma preferência clara pelos resultados gerados pela técnica sintática. De forma que sua utilização, traz uma visão mais clara e objetiva sobre as reais mudanças ocorridas no código, confirmando a hipótese que seus resultados fazem diferença na prática.

Outro ponto bastante relevante foi identificar que a efetividade da ferramenta utilizada pode variar dependendo do cenário. Observamos que, em casos de mudanças mais complexas, especialmente aquelas que envolvem movimentações extensas de trechos de código, o *diff* sintático tende a ser preferido. Por outro lado, em cenários mais simples, que envolvem alterações pontuais em uma única linha ou pequenas movimentações, o *diff* textual pode ser suficiente para representar as modificações de forma clara.

Embora o cenário influencie o nível de preferência dos participantes, observamos que, em todos os casos analisados nesta pesquisa, houve uma preferência significativamente maior pelo resultado gerado pela técnica sintática. Isso demonstra que, se mesmo em cenários simples o *diff* sintático pode fazer diferença e gerar resultados mais intuitivos, em cenários mais complexos, que demandam uma interpretação mais aprofundada dos programadores, seu impacto tende a ser ainda mais significativo.

9. Bibliografia

ASADUZZAMAN, M.; ROY, C. K.; SCHNEIDER, K. A.; PENTA, M. D. LHDiff: A Language-Independent Hybrid Approach for Tracking Source Code Lines. In: 2013 IEEE International Conference on Software Maintenance, Eindhoven, Netherlands, 2013. p. 230-239. DOI: 10.1109/ICSM.2013.34

BARABUCCI, Gioele; CIANCARINI, Paolo; DI IORIO, Angelo; VITALI, Fabio. Measuring the quality of diff algorithms: a formalization. Computer Standards & Interfaces, v. 46, p. 52-65, 2016. ISSN 0920-5489. DOI: 10.1016/j.csi.2015.12.005.

BILLE, Philip. A survey on tree edit distance and related problems. Theoretical Computer Science, v. 337, n. 1–3, p. 217-239, 2005. ISSN 0304-3975. DOI: 10.1016/j.tcs.2004.12.030.

CANFORA, G.; CERULO, L.; PENTA, M. D. Ldiff: An enhanced line differencing tool. In: 2009 IEEE 31st International Conference on Software Engineering, Vancouver, BC, 2009. p. 595-598. DOI: 10.1109/ICSE.2009.5070564.

CHAWATHE, S. S.; RAJARAMAN, A.; GARCIA-MOLINA, H.; WIDOM, J. Change detection in hierarchically structured information. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD ’96). New York, NY, USA: ACM, 1996. p. 493–504.

DECKER, Michael John; COLLARD, Michael L.; VOLKERT, Lucas G.; MALETIC, John I. srcDiff: A syntactic differencing approach to improve the understandability of deltas. Journal of Software: Evolution and Process, v. 32, n. 4, p. e2226, 2020

DOTZLER, G.; PHILIPPSEN, M. Move-optimized source code tree differencing. In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING (ASE), 31., 2016, Singapore. Anais... Singapore: IEEE, 2016. p. 660-671.

FALLERI, Jean-Rémy; MORANDAT, Floréal; BLANC, Xavier; MARTINEZ, Matias; MONPERRUS, Martin. Fine-grained and accurate source code differencing. In: Proceedings

of the International Conference on Automated Software Engineering, 2014, Västerås, Sweden.

FALLERI, Jean-Rémy; MARTINEZ, Matias. Fine-grained, accurate and scalable source differencing. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024.

FAN, Yuanrui; XIA, Xin; LO, David; HASSAN, Ahmed E.; WANG, Yuan; LI, Shanping. A differential testing approach for evaluating abstract syntax tree mapping algorithms. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2021. p. 1174–1185.

FLURI, B.; WÜRSCH, M.; PINZGER, M.; GALL, H. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. IEEE Transactions on Software Engineering, v. 33, n. 11, p. 725-743, nov. 2007. DOI: 10.1109/TSE.2007.70731.

FRICK, V.; GRASSAUER, T.; BECK, F.; PINZGER, M. Generating accurate and compact edit scripts using tree differencing. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 2018. p. 264-274. DOI: 10.1109/ICSME.2018.00036.

HANAM, Q.; BRITO, F. S. M.; MESBAH, A. Discovering bug patterns in JavaScript. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE ’16). New York, NY, USA: ACM, 2016. p. 144–156.

MARTINEZ, Matias; FALLERI, Jean-Rémy; MONPERRUS, Martin. Hyperparameter optimization for AST differencing. IEEE Transactions on Software Engineering, v. 49, n. 10, p. 4814-4828, 2023.

MYERS, Eugene W. An O(ND) difference algorithm and its variations. Algorithmica, v. 1, n. 1, p. 251-266, 1986.

NUGROHO, Yusuf Sulistyo; HATA, Hideaki; MATSUMOTO, Kenichi. How different are different diff algorithms in Git? Use-histogram for code changes. Empirical Software Engineering, v. 25, p. 790-823, 2020.

PALIX, N.; FALLERI, J.-R.; LAWALL, J. Improving pattern tracking with a language-aware tree differencing algorithm. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2015. p. 43-52. DOI: 10.1109/SANER.2015.7081814.

SCHEUREN, Fritz. What is a survey? Alexandria: American Statistical Association, 2004.

SIEFERT, Christopher M.; SMITH, Timothy A.; RIDGWAY, Elliott M. Evaluation of Programming Language-Aware Diffs for Improving Developer Productivity. 2021. Disponível em: https://doi.org/10.2172/1832300.

SPINELLIS, D. Git. IEEE Software, v. 29, n. 3, p. 100-101, maio/jun. 2012. DOI: 10.1109/MS.2012.61.

YANG, C.; WHITEHEAD, E. J. Pruning the AST with Hunks to Speed up Tree Differencing. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). Hangzhou, China: IEEE, 2019. p. 15-25. DOI: 10.1109/SANER.2019.8668032.

Apêndice A - Questionário

Análise de ferramentas de diff

Oi pessoal!

Este formulário busca avaliar a eficácia de diferentes ferramentas de *diff* textual e sintático. A pesquisa tem como objetivo identificar qual ferramenta apresenta os resultados mais adequados em cada um dos cenários propostos. Os participantes da pesquisa devem ter, pelo menos, 6 meses de experiência como desenvolvedor de software e ter mais de 18 anos.

O formulário leva cerca de 15 minutos para ser respondido.

O formulário foi criado em 2024 por Isabela Menezes, estudante de graduação (iclm@cin.ufpe.br), e supervisionada pelo professor Paulo Borba (phmb@cin.ufpe.br), ambos do Centro de Informática da Universidade Federal de Pernambuco.

A participação de vocês é fundamental para o sucesso da minha pesquisa.

Muito obrigada pela colaboração!

DECLARAÇÃO DE CONSENTIMENTO

Ao responder a este questionário, você concorda em permitir que os pesquisadores responsáveis, Isabela Menezes e Paulo Borba, utilizem e divulguem as informações **anônimas** fornecidas, conforme descrito abaixo.

Durante o estudo, há risco de confidencialidade e vazamento de dados, pois os participantes irão fornecer respostas sobre preferências no ambiente empresarial. Para mitigar isso, **os dados serão anonimizados**. Existe também a possibilidade de desconforto psicológico ao responder sobre práticas de trabalho, o que será minimizado com um formulário claro, objetivo, de no máximo 15 minutos, evitando questões controversas e sobrecarregamento.

A pesquisa não trará benefícios diretos aos participantes, mas seus resultados poderão futuramente contribuir para a melhor compreensão e uso das ferramentas estudadas, orientar programadores e pesquisadores, direcionar estudos acadêmicos e melhorar o processo de identificação de mudanças no código, tornando-o mais claro e preciso.

O projeto foi submetido e aprovado pelo Comitê de Ética em Pesquisa - CEP (CAAE: 84860524.2.0000.5208).

CONDIÇÕES E ESTIPULAÇÕES

1. Eu compreendo que todas as informações são confidenciais. Não serei identificado(a) pessoalmente. Concordo em completar o questionário para fins de pesquisa e que os dados derivados deste questionário **anônimo** podem ser publicados em revistas, conferências e postagens de blog.

2. Eu compreendo que minha participação neste questionário de pesquisa é **totalmente voluntária** e que se recusar a participar não implica em penalidade ou perda de benefícios.

3. Eu compreendo que posso contatar o pesquisador caso tenha qualquer dúvida sobre o questionário de pesquisa. Estou ciente de que meu consentimento não me trará benefícios diretos. Também estou ciente de que o autor manterá os dados coletados de forma perpétua e poderá utilizá-los em trabalhos acadêmicos futuros.

4. Eu compreendo que ao clicar no botão abaixo, confirmo que sou maior de idade e tenho mais de 18 anos completos.

5. Eu compreendo que os dados não podem ser removidos depois do preenchimento, mesmo que o participante solicite, visto que as informações não são identificáveis.

6. Ao clicar no botão abaixo, forneço livremente meu consentimento e reconheço meus direitos como participante voluntário de pesquisa, conforme descrito acima, e autorizo os pesquisadores a utilizarem minhas informações para a realização da pesquisa. Também confirmo que sou maior de idade para preencher este questionário.

Sua experiência

3. Há quanto tempo você trabalha como programador? * *Marcar apenas uma oval.*

- ☐ 6 meses - 1 ano
- ☐ 2 anos - 5 anos
- ☐ 5 anos ou mais

4. Você já trabalhou com alguma dessas linguagens de programação?

- ☐ Java
- ☐ C
- ☐ Outro:

5. Diariamente, utilizamos o comando "*git diff*" para mostrar a diferença entre versões de arquivos de código-fonte, este comando utiliza do mecanismo de *diff textual* para realizar este processo. No entanto, existem outras formas de realizar a diferenciação de arquivos e diretórios, como o *diff sintático*. Você ja ouviu falar de ferramentas que utilizam o *diff sintático*?

- ☐ Sim
- ☐ Não
- ☐ Já ouvir falar, mas nunca utilizei

Cenário 1

Resultado 1

```
@@ -1,14 +1,14 @@
-void Chunk_copy(Chunk *src, size_t src_start, Chunk *dst, size_t dst_start, size_t n)
+int Chunk_bounds_check(Chunk *chunk, size_t start, size_t n)
 {
-    if (!Chunk_bounds_check(src, src_start, n)) return;
-    if (!Chunk_bounds_check(dst, dst_start, n)) return;
+    if (chunk == NULL) return 0;

-    memcpy(dst->data + dst_start, src->data + src_start, n);
+    return start <= chunk->length && n <= chunk->length - start;
 }

-int Chunk_bounds_check(Chunk *chunk, size_t start, size_t n)
+void Chunk_copy(Chunk *src, size_t src_start, Chunk *dst, size_t dst_start, size_t n)
 {
-    if (chunk == NULL) return 0;
+    if (!Chunk_bounds_check(src, src_start, n)) return;
+    if (!Chunk_bounds_check(dst, dst_start, n)) return;

-    return start <= chunk->length && n <= chunk->length - start;
+    memcpy(dst->data + dst_start, src->data + src_start, n);
 }
\ No newline at end of file
```

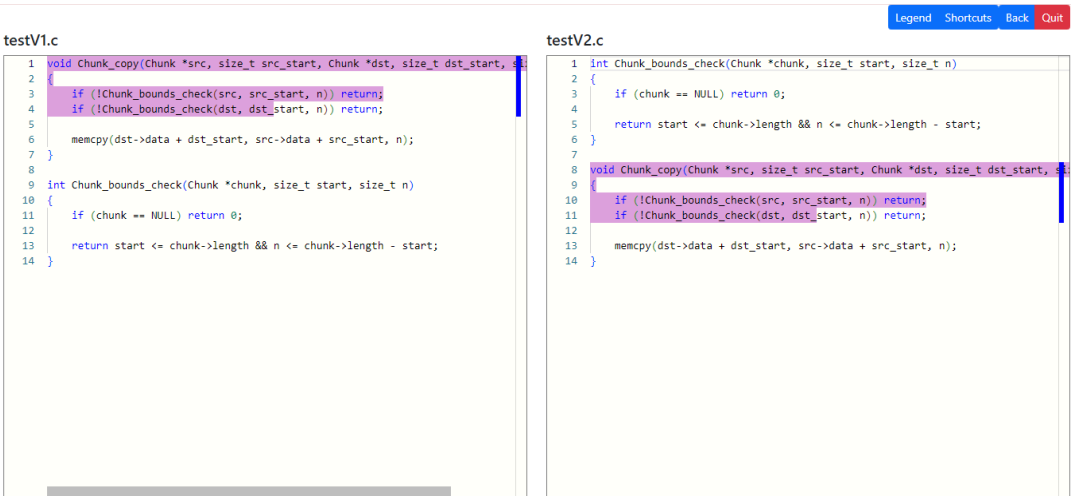
Legenda do Resultado 1

Na imagem, é exibido o resultado da diferenciação entre duas versões, uma mais antiga e outra mais recente, de um mesmo arquivo. Cada linha indica uma ação identificada pela ferramenta:

1) Linha verde: Adição da respectiva linha no arquivo mais recente.

2) Linha vermelha: Deleção da respectiva linha no arquivo mais recente.

Resultado 2



Legenda do Resultado 2

Na imagem, exibe-se as duas versões de um mesmo arquivo, ao lado esquerdo a mais antiga e ao lado direito a mais recente.

Cada cor sinaliza uma ação identificada pela ferramenta:

1) Roxo: Movimentação do trecho de código (código existia em uma posição diferente no arquivo antigo)

2) Amarelo: Modificação de um elemento sintático

3) Verde: Adição de um novo trecho de código

4) Vermelho: Deleção de um trecho de código

6. Em uma situação de desenvolvimento, qual dos dois *diffs* você preferiria?

- ☐ Resultado 1
- ☐ Resultado 2
- ☐ Os dois resultados são equivalentes

7.Com base na sua resposta anterior, por favor, explique o que influenciou sua escolha. Se você marcou um dos dois como superior, o que considera ser a principal vantagem? Caso tenha marcado como equivalente, o que faz você ver as duas opções como similares?

8. Você achou que alguma das imagens era confusa ou difícil de entender?

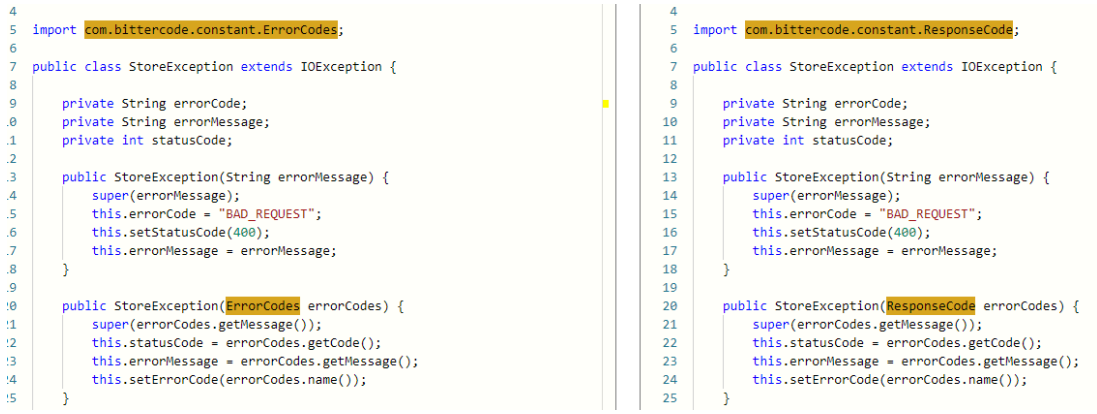
- ☐ Sim, a imagem do Resultado 1
- ☐ Sim, a imagem do Resultado 2
- ☐ Sim, ambas são confusas
- ☐ Não, ambas são claras

9. A ação exemplificada na imagem (movimentação de código) costuma ocorrer regularmente em sua rotina de trabalho?

- ☐ Sim
- ☐ Não

Cenário 2

Resultado 3



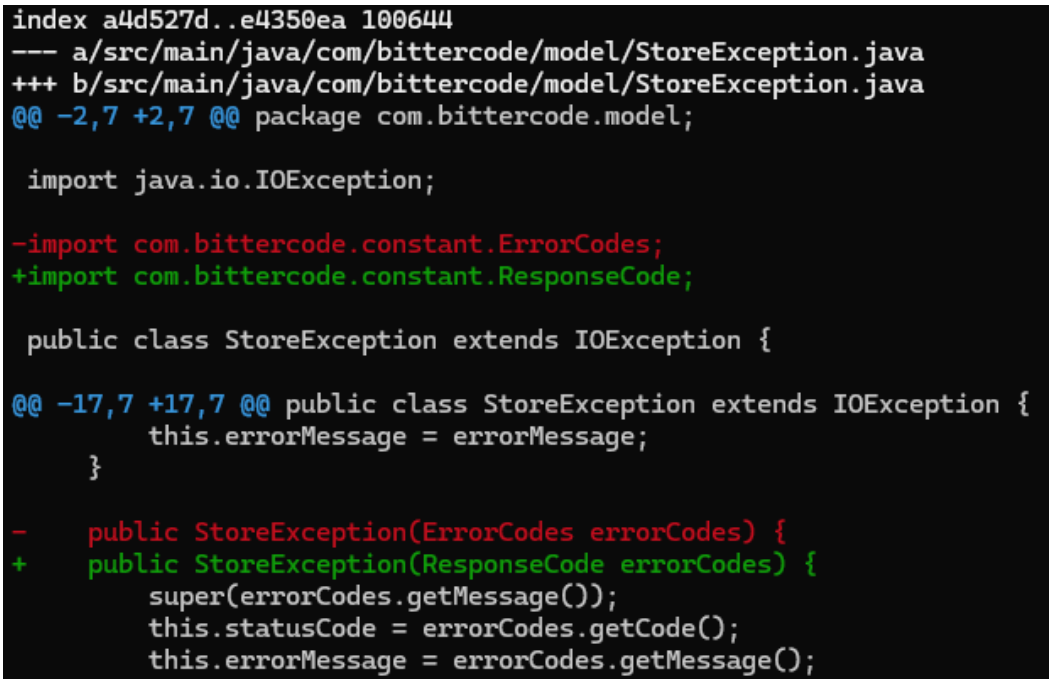
```
4
5  import com.bittercode.constant.ErrorCodes;
6
7  public class StoreException extends IOException {
8
9      private String errorCode;
0      private String errorMessage;
1      private int statusCode;
2
3      public StoreException(String errorMessage) {
4          super(errorMessage);
5          this.errorCode = "BAD_REQUEST";
6          this.setStatusCode(400);
7          this.errorMessage = errorMessage;
8      }
9
0      public StoreException(ErrorCodes errorCodes) {
1          super(errorCodes.getMessage());
2          this.statusCode = errorCodes.getCode();
3          this.errorMessage = errorCodes.getMessage();
4          this.setErrorCode(errorCodes.name());
5      }
```

```
4
5  import com.bittercode.constant.ResponseCode;
6
7  public class StoreException extends IOException {
8
9      private String errorCode;
10     private String errorMessage;
11     private int statusCode;
12
13     public StoreException(String errorMessage) {
14         super(errorMessage);
15         this.errorCode = "BAD_REQUEST";
16         this.setStatusCode(400);
17         this.errorMessage = errorMessage;
18     }
19
20     public StoreException(ResponseCode errorCodes) {
21         super(errorCodes.getMessage());
22         this.statusCode = errorCodes.getCode();
23         this.errorMessage = errorCodes.getMessage();
24         this.setErrorCode(errorCodes.name());
25     }
```

Legenda do Resultado 3

Na imagem, exibe-se as 2 versões de um mesmo arquivo, ao lado esquerdo a mais antiga e ao lado direito a mais recente.
Cada cor sinaliza uma ação identificada pela ferramenta:
1) Roxo: Movimentação do trecho de código
2) Amarelo: Modificação de um elemento sintático
3) Verde: Adição de um novo trecho de código
4) Vermelho: Deleção de um trecho de código

Resultado 4



```
index a4d527d..e4350ea 100644
--- a/src/main/java/com/bittercode/model/StoreException.java
+++ b/src/main/java/com/bittercode/model/StoreException.java
@@ -2,7 +2,7 @@ package com.bittercode.model;

 import java.io.IOException;

-import com.bittercode.constant.ErrorCodes;
+import com.bittercode.constant.ResponseCode;

 public class StoreException extends IOException {

@@ -17,7 +17,7 @@ public class StoreException extends IOException {
         this.errorMessage = errorMessage;
     }

-    public StoreException(ErrorCodes errorCodes) {
+    public StoreException(ResponseCode errorCodes) {
         super(errorCodes.getMessage());
         this.statusCode = errorCodes.getCode();
         this.errorMessage = errorCodes.getMessage();
```

Legenda do Resultado 4

Na imagem, é exibido o resultado da diferenciação entre duas versões, uma mais antiga e outra mais recente, de um mesmo arquivo. Cada linha indica uma ação identificada pela ferramenta:

1) Linha verde: Adição da respectiva linha no arquivo mais recente.

2) Linha vermelha: Deleção da respectiva linha no arquivo mais recente.

10. Em uma situação de desenvolvimento, qual dos dois *diffs* você preferiria?

- ☐ Resultado 3
- ☐ Resultado 4
- ☐ Os dois resultados são equivalentes

11. Com base na sua resposta anterior, por favor, explique o que influenciou sua escolha. Se você marcou um dos dois como superior, o que considera ser a principal vantagem? Caso tenha marcado como equivalente, o que faz você ver as duas opções como similares?

12. Você achou que alguma das imagens era confusa ou difícil de entender?

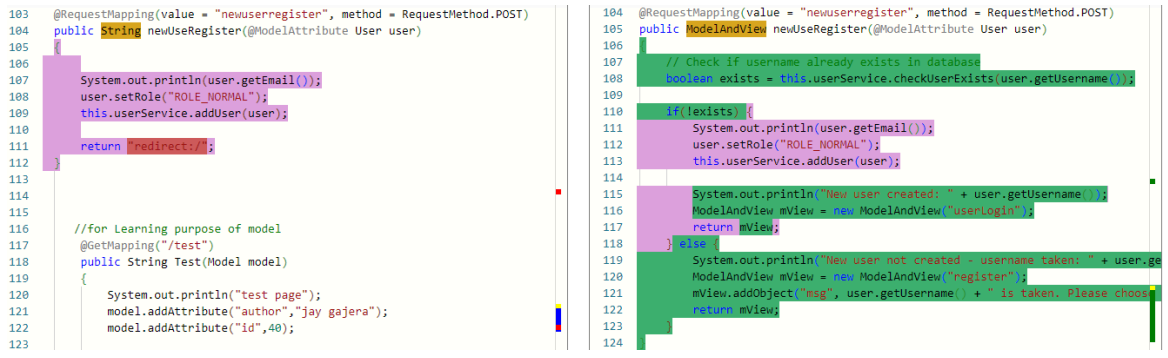
- ☐ Sim, a imagem do Resultado 3
- ☐ Sim, a imagem do Resultado 4
- ☐ Sim, ambas são confusas
- ☐ Não, ambas são claras

13. A ação exemplificada na imagem (mudança no import) costuma ocorrer regularmente em sua rotina de trabalho?

- ☐ Sim
- ☐ Não

Cenário 3

Resultado 5



```
@RequestMapping(value = "newuserregister", method = RequestMethod.POST)
public String newUseRegister(@ModelAttribute User user)
{

    System.out.println(user.getEmail());
    user.setRole("ROLE_NORMAL");
    this.userService.addUser(user);

    return "redirect:/";
}



//for Learning purpose of model
@GetMapping("/test")
public String Test(Model model)
{
    System.out.println("test page");
    model.addAttribute("author","jay gajera");
    model.addAttribute("id",40);
```

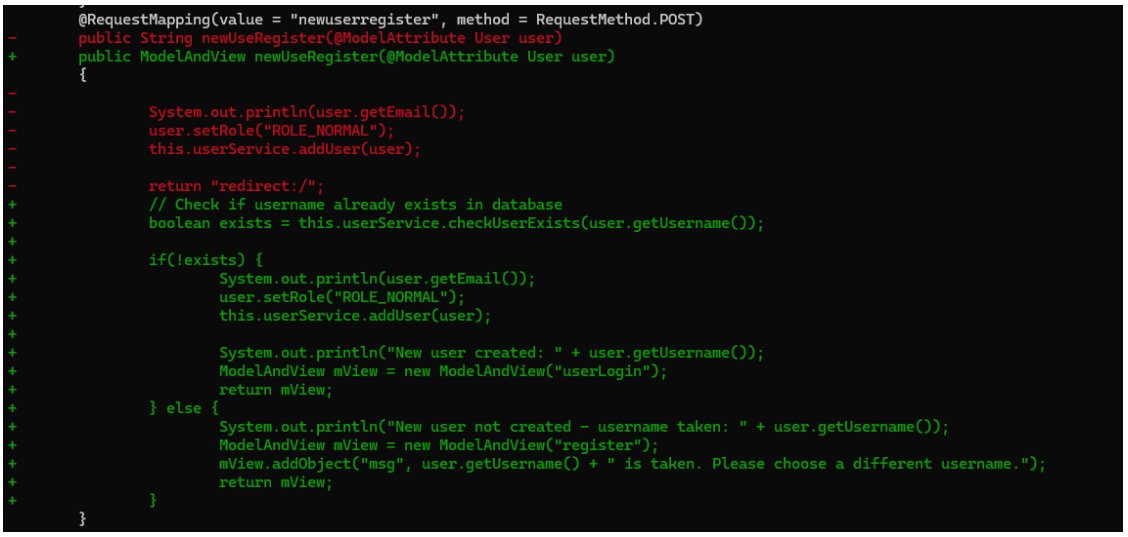
```
@RequestMapping(value = "newuserregister", method = RequestMethod.POST)
public ModelAndView newUseRegister(@ModelAttribute User user)
{
    // Check if username already exists in database
    boolean exists = this.userService.checkUserExists(user.getUsername());

    if(!exists) {
        System.out.println(user.getEmail());
        user.setRole("ROLE_NORMAL");
        this.userService.addUser(user);

        System.out.println("New user created: " + user.getUsername());
        ModelAndView mView = new ModelAndView("userLogin");
        return mView;
    } else {
        System.out.println("New user not created - username taken: " + user.ge
        ModelAndView mView = new ModelAndView("register");
        mView.addObject("msg", user.getUsername() + " is taken. Please choos
        return mView;
    }
}
```

Legenda do Resultado 5

Na imagem, exibe-se as 2 versões de um mesmo arquivo, ao lado esquerdo a mais antiga e ao lado direito a mais recente.
Cada cor sinaliza uma ação identificada pela ferramenta:

1) Roxo: Movimentação do trecho de código
2) Amarelo: Modificação de um elemento sintático
3) Verde: Adição de um novo trecho de código
4) Vermelho: Deleção de um trecho de código

Resultado 6



```
        @RequestMapping(value = "newuserregister", method = RequestMethod.POST)
-       public String newUseRegister(@ModelAttribute User user)
+       public ModelAndView newUseRegister(@ModelAttribute User user)
        {
-
-               System.out.println(user.getEmail());
-               user.setRole("ROLE_NORMAL");
-               this.userService.addUser(user);
-
-               return "redirect:/";
+               // Check if username already exists in database
+               boolean exists = this.userService.checkUserExists(user.getUsername());
+
+               if(!exists) {
+                       System.out.println(user.getEmail());
+                       user.setRole("ROLE_NORMAL");
+                       this.userService.addUser(user);
+
+                       System.out.println("New user created: " + user.getUsername());
+                       ModelAndView mView = new ModelAndView("userLogin");
+                       return mView;
+               } else {
+                       System.out.println("New user not created - username taken: " + user.getUsername());
+                       ModelAndView mView = new ModelAndView("register");
+                       mView.addObject("msg", user.getUsername() + " is taken. Please choose a different username.");
+                       return mView;
+               }
        }
```

Legenda do Resultado 6

Na imagem, é exibido o resultado da diferenciação entre duas versões, uma mais antiga e outra mais recente, de um mesmo arquivo. Cada linha indica uma ação identificada pela ferramenta:

1) Linha verde: Adição da respectiva linha no arquivo mais recente.

2) Linha vermelha: Deleção da respectiva linha no arquivo mais recente.

14. Em uma situação de desenvolvimento, qual dos dois *diffs* você preferiria?

- ☐ Resultado 5
- ☐ Resultado 6
- ☐ Os dois resultados são equivalentes

15. Com base na sua resposta anterior, por favor, explique o que influenciou sua escolha. Se você marcou um dos dois como superior, o que considera ser a principal vantagem? Caso tenha marcado como equivalente, o que faz você ver as duas opções como similares?

16. Você achou que alguma das imagens era confusa ou difícil de entender?

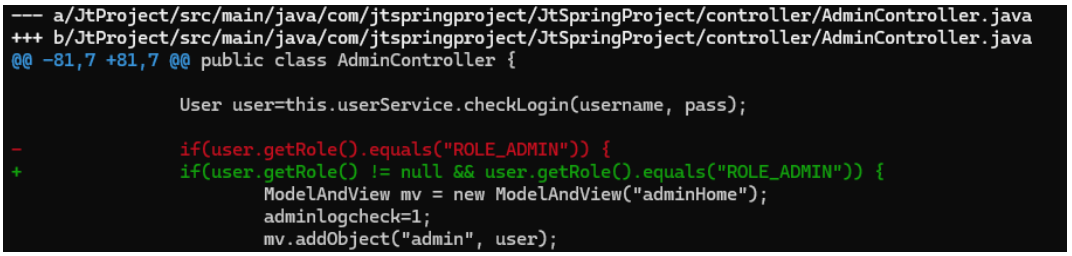
- ☐ Sim, a imagem do Resultado 5
- ☐ Sim, a imagem do Resultado 6
- ☐ Sim, ambas são confusas
- ☐ Não, ambas são claras

17. A ação exemplificada na imagem (mudanças internas em um método) costuma ocorrer regularmente em sua rotina de trabalho?

- ☐ Sim
- ☐ Não

Cenário 4

Resultado 7



Legenda do Resultado 7

Na imagem, é exibido o resultado da diferenciação entre duas versões, uma mais antiga e outra mais recente, de um mesmo arquivo. Cada linha indica uma ação identificada pela ferramenta:

1) Linha verde: Adição da respectiva linha no arquivo mais recente.

2) Linha vermelha: Deleção da respectiva linha no arquivo mais recente.

Resultado 8

```
if(user.getRole().equals("ROLE_ADMIN")) {
    ModelAndView mv = new ModelAndView("adminHome");
    adminlogcheck=1;
    mv.addObject("admin", user);
    return mv;
}
else {
    ModelAndView mv = new ModelAndView("adminlogin");
    mv.addObject("msg", "Please enter correct username and password");
    return mv;
}
```

```
if(user.getRole() != null && user.getRole().equals("ROLE_ADMIN")) {
    ModelAndView mv = new ModelAndView("adminHome");
    adminlogcheck=1;
    mv.addObject("admin", user);
    return mv;
}
else {
    ModelAndView mv = new ModelAndView("adminlogin");
    mv.addObject("msg", "Please enter correct username and password");
    return mv;
}
```

Legenda do Resultado 8

Na imagem, exibe-se as 2 versões de um mesmo arquivo, ao lado esquerdo a mais antiga e ao lado direito a mais recente.
Cada cor sinaliza uma ação identificada pela ferramenta:
1) Roxo: Movimentação do trecho de código
2) Amarelo: Modificação de um elemento sintático
3) Verde: Adição de um novo trecho de código
4) Vermelho: Deleção de um trecho de código

18. Em uma situação de desenvolvimento, qual dos dois *diffs* você preferiria?

- ☐ Resultado 7
- ☐ Resultado 8
- ☐ Os dois resultados são equivalentes

19. Com base na sua resposta anterior, por favor, explique o que influenciou sua escolha. Se você marcou um dos dois como superior, o que considera ser a principal vantagem? Caso tenha marcado como equivalente, o que faz você ver as duas opções como similares?

---

20. Você achou que alguma das imagens era confusa ou difícil de entender?

- ☐ Sim, a imagem do Resultado 7
- ☐ Sim, a imagem do Resultado 8
- ☐ Não, ambas são claras

21. A ação exemplificada na imagem (alteração de um condicional) costuma ocorrer regularmente em sua rotina de trabalho?

- ☐ Sim
- ☐ Não

22. Após analisar os casos, você acredita que o uso de diff sintático poderia impactar de alguma forma a sua rotina de trabalho? Comente sobre:

---

23. Você estaria interessado em adotar ferramentas de diff sintático em seus projetos futuros?

- ☐ Sim
- ☐ Não
- ☐ Talvez