FACULDADE DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

MIGUEL VIEIRA DE ALMEIDA

Licenciatura em Engenharia Informática

INTEGRAÇÃO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE REVISÃO DE CÓDIGO NO CICLO DE VIDA DE DESENVOLVIMENTO DE SOFTWARE

Plano de Dissertação
MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa

Draft: *27 de janeiro de 2026*

FACULDADE DE
CIÊNCIAS E TECNOLOGIA

INTEGRAÇÃO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE REVISÃO DE CÓDIGO NO CICLO DE VIDA DE DESENVOLVIMENTO DE SOFTWARE

MIGUEL VIEIRA DE ALMEIDA

Licenciatura em Engenharia Informática

Orientador: João Sousa
Consultor Sénior, Processware

Coorientador: Jorge Cruz
Professor, NOVA University Lisbon

Plano de Dissertação
MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Draft: *27 de janeiro de 2026*

Resumo

Por fazer...

Abstract

To be Made

Índice

Índice de Figuras

Índice de Tabelas

Siglas

API Application Programming Interface *(pp. 13, 20–24, 26)*
AST Abstract Syntax Tree *(pp. 8, 15)*

CI Continuous Integration *(pp. 7, 13)*
CR Code Review *(pp. iii, 1–10, 12–14, 17, 18, 27)*

Diff Diferença de código entre versões (*diff*) *(pp. 3, 4, 9–11, 13, 15, 17, 18, 21, 22, 24)*
DL Deep Learning *(pp. 8, 9)*

IDE Integrated Development Environment *(pp. iii, 15)*

LLM Large Language Model *(pp. iii, iv, 1–4, 8–12, 14–18, 20, 21, 23, 25–27)*

ML Machine Learning *(pp. 7, 8)*

NLP Natural Language Processing *(p. 8)*

PR Pull Request *(pp. 1, 3, 4, 6, 9–15, 17–24, 26)*

1

Introdução

Este capítulo tem como objetivo introduzir o tema e objetivos desta dissertação. É apresentada a motivação para a realização do projeto, o contexto em que o problema abordado se insere e os objetivos a serem alcançados.

1.1 Motivação

O aumento da complexidade do software moderno e o ritmo acelerado de desenvolvimento tornam a CR uma etapa essencial no ciclo de vida do software. Este processo garante a qualidade, segurança e conformidade do código com as boas práticas de engenharia e com os padrões internos definidos pelas organizações [0, 0].

Contudo, a CR continua a ser uma tarefa exigente e demorada, que frequentemente exige múltiplos revisores para evitar a introdução de erros [0]. Em ambientes empresariais com grande volume de commits e Pull Requests (PRs), os revisores lidam diariamente com alterações extensas que exigem análise cuidadosa. Isto pode originar atrasos, inconsistências e risco de falhas quando as mudanças chegam ao produto final [0, 0].

No contexto específico desta dissertação, estes desafios tornam-se evidentes. A Processware mantém uma base de código ampla, distribuída por vários repositórios e com múltiplas tecnologias — como C, C++, C#, Vue, SQL — coexistindo com código legacy. Esta diversidade aumenta a complexidade da revisão e exige conhecimentos técnicos específicos por parte dos revisores [0]. Além disso, diferentes equipas podem aplicar práticas distintas, dificultando a uniformização de padrões internos e levando a revisões repetitivas e demoradas [0].

Com o avanço recente dos LLMs, já amplamente demonstrado em ferramentas como ChatGPT, Gemini ou GitHub Copilot, tornou-se evidente que estes modelos têm capacidade para compreender e analisar código-fonte, identificar problemas e sugerir melhorias [0, 0]. Assim, a adoção de LLMs na revisão de código surge como uma oportunidade para reduzir esforço manual, aumentar a consistência e melhorar a eficiência do processo [0, 0].

1.2 Contexto

Esta dissertação enquadra-se no conjunto de soluções desenvolvidas pela Processware para melhorar o processo interno de revisão de código.

A Processware é uma empresa tecnológica com presença internacional, especializada no desenvolvimento de soluções para otimização de processos e gestão operacional em ambientes empresariais complexos. A sua atividade centra-se na transformação digital de operações através da combinação de engenharia de processos e plataformas tecnológicas proprietárias, apoiando organizações na melhoria da eficiência, controlo e escalabilidade das suas operações.

No centro da sua oferta encontra-se a plataforma O2P, uma solução baseada na cloud que permite a digitalização, monitorização e otimização de processos operacionais em tempo real, integrando-se com sistemas existentes e suportando diferentes contextos tecnológicos. A empresa atua em sectores com elevada complexidade operacional, caracterizados por múltiplas tecnologias, elevados volumes de dados e necessidade de elevada fiabilidade.

Neste contexto, a Processware mantém uma base de código extensa e heterogénea, distribuída por vários repositórios e tecnologias, o que coloca desafios significativos ao nível da manutenção e da revisão de código. A necessidade de garantir qualidade, consistência e alinhamento com boas práticas internas torna o processo de CR um elemento crítico no seu ciclo de desenvolvimento de software.

Atualmente, o processo de CR é totalmente manual e depende fortemente da experiência dos revisores. O elevado volume de alterações, a variedade tecnológica e a coexistência de código moderno com código legacy tornam a revisão uma tarefa exigente e pouco escalável [0]. Esta situação resulta em tempos de análise elevados e inconsistências entre equipas, dificultando a padronização de práticas internas [0].

Dado este cenário, torna-se pertinente explorar soluções baseadas em inteligência artificial, capazes de apoiar a análise de código e contribuir para um processo de CR mais eficiente, coerente e alinhado com as necessidades atuais de desenvolvimento de software. Estudos recentes mostram que LLMs podem atuar como revisores assistentes, melhorar a qualidade dos comentários, aumentar a uniformidade e acelerar a deteção de problemas [0, 0].

1.3 Definição do Problema

O processo de CR na Processware é atualmente manual, exigindo que os revisores analisem cada alteração submetida nos diferentes repositórios e tecnologias utilizadas pela empresa. Consequentemente, esta abordagem apresenta várias limitações. Numa primeira instância, a revisão é uma tarefa demorada e dependente das competências técnicas dos revisores, o que pode levar ao aumento da probabilidade de ocorrerem erros e inconsistências [0]. A coexistência de múltiplas tecnologias e de código legacy também dificultam

na identificação de problemas numa forma sistemática [0].

Além disso, revisores poderão adotar critérios distintos, dependendo da PR em questão, resultando em comentários heterogéneos e dificultando a uniformização de práticas internas. Estudos recentes mostram que esta falta de consistência é um problema comum em ambientes empresariais e afeta tanto a qualidade das revisões de código como a eficiência deste processo [0]. Outro problema recorrente é a necessidade de realizar comentários repetitivos, como correções de estilo, validações simples ou alertas sobre riscos conhecidos, que consomem tempo e poderiam ser automatizados [0].

Com os avanços recentes em LLMs, surgiu a oportunidade de automatizar grande parte deste processo. Estes modelos têm demonstrado capacidade para analisar código, identificar problemas e gerar comentários úteis [0]. No entanto, existem registos que apontam limitações: a utilidade dos comentários varia significativamente, alguns modelos tendem a gerar falsos positivos e a eficácia depende da clareza e pertinência das recomendações produzidas [0][0]. Importa ainda referir que LLMs genéricos não compreendem as práticas e padrões internos de cada organização, reduzindo a sua aplicabilidade prática durante o processo de CR.

Assim, o problema central que esta dissertação procura resolver consiste na ausência de um sistema capaz de apoiar a CR na Processware, produzindo comentários automáticos, alinhados com as boas práticas internas, reduzindo esforço repetitivo e contribuindo para um processo mais consistente, eficiente e escalável. Este desafio está alinhado com a necessidade de adaptar LLMs a contextos organizacionais específicos, de forma a tirar partido do seu potencial no processo de CR [0][0].

1.4 Objetivos

O principal objetivo desta dissertação é desenvolver um sistema capaz de apoiar o processo de CR na Processware, fornecendo comentários automáticos consistentes com as boas práticas internas e ajudando a reduzir o esforço associado a tarefas repetitivas. Pretende-se criar uma solução que aumente a eficiência, a uniformidade e a qualidade das revisões de código realizadas na empresa.

Para atingir este propósito, foram definidos os seguintes objetivos:

1. **Identificação e formalização de boas práticas internas:** analisar o código existente e recolher conhecimento junto da equipa responsável pelo processo de CR para mapear os padrões, convenções e recomendações atualmente aplicados neste processo.

2. **Análise inteligente de alterações de código:** desenvolver um sistema capaz de interpretar Diferença de código entre versões (*diff*)s (Diffs) e compreender o contexto das alterações submetidas, tendo em conta as várias tecnologias utilizadas na Processware.

3. **Geração de comentários consistentes e acionáveis:** criar um mecanismo que produza comentários claros, concisos e alinhados com as boas práticas internas, ajudando os revisores a detetar problemas recorrentes e a manter a consistência entre PRs.

4. **Integração fluida no fluxo de desenvolvimento:** garantir que a solução pode ser incorporada no processo atual de revisão de código, permitindo que os comentários sejam disponibilizados durante a análise de PRs.

5. **Avaliação da eficácia e utilidade prática:** validar o sistema através de métricas objetivas e subjetivas, garantindo que a solução contribui para revisões mais eficientes e uniformes.

Este conjunto de objetivos visa assegurar que o sistema desenvolvido traz valor real ao processo de CR da Processware, promovendo maior qualidade, consistência e produtividade no desenvolvimento de software.

1.5 Solução Proposta

A solução proposta consiste no desenvolvimento de um sistema de apoio à CR baseado em LLMs, capaz de analisar automaticamente alterações submetidas nas PRs e gerar comentários alinhados com as boas práticas internas da Processware. O sistema pretende atuar como um assistente de revisão, reduzindo tarefas repetitivas, promovendo maior consistência entre equipas e aumentando a eficiência do processo.

A proposta assenta nos seguintes componentes principais:

1. **Extração e pré-processamento das alterações de código:** o sistema recebe o Diff da PR e transforma-o numa representação adequada para análise, garantindo que apenas as alterações relevantes são processadas.

2. **Interpretação das mudanças com suporte de LLMs:** um modelo de linguagem é utilizado para compreender o contexto das alterações, identificar possíveis problemas e reconhecer padrões que violam boas práticas previamente definidas.

3. **Geração de comentários automáticos:** com base na análise realizada, o sistema produz comentários claros, concisos e acionáveis, seguindo as recomendações internas da Processware e facilitando o trabalho dos revisores humanos.

4. **Módulo de personalização de boas práticas:** permite adaptar regras e orientações às necessidades da empresa, garantindo que os comentários automáticos refletem padrões internos e preferências específicas de diferentes equipas.

5. **Integração com o fluxo de desenvolvimento existente:** o sistema é incorporado diretamente no processo atual da Processware, permitindo que os comentários sejam adicionados nas PRs através das ferramentas já utilizadas pela equipa.

6. **Monitorização e melhoria contínua:** a solução recolhe métricas e feedback dos utilizadores para identificar oportunidades de melhoria, ajustando o comportamento do sistema de forma progressiva ao longo do tempo.

Em conjunto, estes componentes constituem uma solução prática e extensível, preparada para apoiar revisores humanos e tornar o processo de CR mais eficiente, consistente e alinhado com as necessidades reais da empresa.

No próximo capítulo serão apresentadas as várias tecnologias a serem utilizadas para o desenvolvimento deste projeto, bem como a análise de trabalhos que sejam considerados relevantes para o enquadramento deste tema de dissertação.

2

Conceitos

Este capítulo tem como objetivo apresentar os principais conceitos teóricos e técnicos que sustentam o desenvolvimento desta dissertação. São abordados os fundamentos relacionados com o processo de CR no contexto do desenvolvimento de *software*, bem como a evolução das abordagens de automação aplicadas a este processo.

2.1 *Code Review* no Desenvolvimento de *Software*

CR é uma prática fundamental no desenvolvimento de software que consiste na análise sistemática do código-fonte por um ou mais programadores, com o objetivo de identificar defeitos, melhorar a qualidade do código e garantir a conformidade com boas práticas e padrões definidos pela organização [0, 0]. Esta atividade é normalmente realizada antes da integração de alterações no ramo principal do projeto, assumindo um papel central nos fluxos de desenvolvimento modernos baseados em PRs.

Historicamente, a revisão de código surgiu como uma prática formal, associada a inspeções estruturadas, como as inspeções de Fagan, que demonstraram benefícios significativos na deteção precoce de erros e na redução de custos associados à correção de defeitos em fases avançadas do desenvolvimento [0]. Com a evolução das metodologias ágeis e das plataformas colaborativas de desenvolvimento, a revisão de código tornou-se um processo mais leve e contínuo, integrado no ciclo diário de desenvolvimento de software [0].

Os principais objetivos da revisão de código incluem a deteção de erros lógicos e defeitos funcionais, a melhoria da legibilidade e manutenibilidade do código, a verificação da conformidade com padrões de estilo e boas práticas, bem como a mitigação de riscos relacionados com segurança e desempenho [0]. Para além dos aspetos técnicos, a revisão de código desempenha também um papel relevante na partilha de conhecimento entre membros da equipa, promovendo a disseminação de boas práticas e contribuindo para a uniformização do código ao longo do projeto [0].

Apesar dos seus benefícios comprovados, a revisão de código é uma atividade exigente

do ponto de vista cognitivo e temporal, dependendo fortemente da experiência e disponibilidade dos revisores. Em projetos de grande escala, caracterizados por elevados volumes de alterações e diversidade tecnológica, este processo pode tornar-se difícil de escalar, originando atrasos, inconsistências e variabilidade na qualidade das revisões realizadas [0, 0].

2.2 Automação do Processo de *Code Review*

Com o aumento da complexidade dos sistemas de software e do volume de alterações submetidas diariamente, surgiu a necessidade de melhorar o processo de CR através de mecanismos de automação. O principal objetivo destas abordagens é reduzir o esforço manual associado a tarefas repetitivas, melhorar a deteção precoce de defeitos e apoiar os revisores humanos na análise das alterações de código [0].

As primeiras tentativas de automação do CR basearam-se essencialmente em ferramentas de análise estática de código. Estas ferramentas analisam o código-fonte sem necessidade de execução, recorrendo a regras pré-definidas para identificar problemas como violações de estilo, padrões perigosos, possíveis erros de execução ou más práticas comuns [0]. Linters e analisadores estáticos tornaram-se parte integrante dos fluxos de desenvolvimento modernos, sendo frequentemente integrados em pipelines de Continuous Integration (CI).

Apesar dos benefícios associados à sua utilização, estas abordagens apresentam limitações significativas. Estudos mostram que ferramentas de análise estática tendem a gerar um elevado número de falsos positivos, o que pode levar à sua desvalorização por parte dos programadores [0]. Além disso, estas ferramentas possuem uma compreensão limitada do contexto em que o código é desenvolvido, não conseguindo capturar a intenção do programador, decisões de design ou requisitos específicos do domínio da aplicação [0].

Outra limitação relevante prende-se com a rigidez das regras utilizadas. A adaptação destas ferramentas a práticas internas ou a estilos específicos de uma organização requer frequentemente configurações manuais complexas, dificultando a sua adoção em ambientes empresariais com grande diversidade tecnológica [0]. Como consequência, a automação tradicional do CR tende a ser eficaz na deteção de problemas simples e bem definidos, mas menos adequada para identificar defeitos mais complexos ou fornecer comentários contextualizados e acionáveis.

Estas limitações motivaram a investigação de abordagens mais avançadas para a automação do processo de revisão de código, explorando técnicas de Machine Learning (ML) capazes de aprender padrões diretamente a partir de grandes volumes de código-fonte e dados históricos de desenvolvimento [0]. Estas abordagens abriram caminho para soluções mais flexíveis e contextuais, que serão discutidas nas secções seguintes.

2.3 Aprendizagem Automática e *Deep Learning* em *Code Review*

As limitações das abordagens tradicionais de automação do CR motivaram a utilização de técnicas de ML e, posteriormente, de Deep Learning (DL). Ao contrário das ferramentas baseadas em regras fixas, estas abordagens permitem aprender padrões diretamente a partir de grandes volumes de código-fonte e dados históricos de desenvolvimento, reduzindo a necessidade de configuração manual e aumentando a capacidade de generalização [0].

Os primeiros trabalhos nesta área exploraram modelos estatísticos e técnicas de aprendizagem supervisionada para identificar padrões de defeitos, prever a probabilidade de erros e priorizar avisos gerados por ferramentas automáticas [0]. Estas abordagens demonstraram que informação histórica, como alterações anteriores e decisões tomadas em revisões passadas, pode ser utilizada para melhorar a eficácia do processo de CR.

Com o avanço de técnicas de DL, surgiram modelos capazes de representar o código-fonte de forma mais expressiva, explorando a sua estrutura sintática e semântica. Trabalhos baseados em redes neuronais profundas passaram a utilizar representações como Abstract Syntax Tree (AST), grafos de fluxo de controlo e sequências de *tokens*, permitindo capturar relações mais complexas entre diferentes componentes do código base [0]. Estes modelos mostram melhorias significativas em tarefas como deteção de defeitos, sugestões de correções e identificação de padrões recorrentes no código.

Outro avanço relevante foi a introdução do conceito de *naturalness* do software, que assume que o código-fonte apresenta regularidades semelhantes às da linguagem natural. Estudos demonstraram que código que se desvia destes padrões tende a estar mais associado a defeitos, o que abriu caminho à aplicação de Natural Language Processing (NLP) à análise de código [0]. Esta perspetiva reforçou a adequação de técnicas originalmente desenvolvidas para processamento de linguagem natural ao domínio do desenvolvimento de software.

Apesar dos progressos alcançados, as abordagens baseadas em ML e DL apresentavam ainda algumas limitações práticas. Muitos modelos exigiam grandes volumes de dados rotulados, eram dispendiosos do ponto de vista computacional e apresentavam dificuldades em lidar com múltiplas linguagens de programação ou contextos organizacionais específicos [0]. Além disso, a integração destas soluções nos fluxos de desenvolvimento reais permanecia um desafio.

Estas limitações criaram as condições para a adoção de LLMs, capazes de aprender representações genéricas de código e de transferir conhecimento entre diferentes tarefas e linguagens. A utilização de LLMs no processo de CR será abordada na secção seguinte.

2.4 Modelos de Linguagem de Grande Escala

Os LLMs representam a evolução mais recente no campo da aprendizagem profunda aplicada à linguagem. São modelos baseados na arquitetura *Transformer*, treinados com grandes quantidades de dados textuais e, no contexto relevante para esta dissertação,

também em código-fonte, que adquiriram capacidades generativas e de compreensão contextual notáveis [**vaswani2017attention**, 0].

Ao contrário dos modelos anteriores de DL em que é necessário algum tipo de calibração ou ajustes específicos para cada tarefa, os LLMs modernos não necessitam de tanta informação ou exemplos para conseguir devolver uma resposta. Isto significa que podem realizar novas tarefas com base apenas em instruções de linguagem natural, sem necessidade de treino adicional extensivo [**brown2020gpt3**]. Esta característica é fundamental para a sua aplicação prática em ambientes empresariais, onde a diversidade de tarefas e a necessidade de rápida adaptação são elevadas.

Dois aspetos tornam os LLMs particularmente adequados para a análise de código:

1. **Treino Multimodal (Texto e Código):** Modelos como o Codex (base do GitHub Copilot), o GPT-4, ou modelos *open-source* como o CodeLLaMA, foram treinados em vastos *corpora* que incluem tanto linguagem natural como código-fonte de múltiplas linguagens de programação. Isto permitiu-lhes aprender não só a sintaxe, mas também padrões semânticos, *idioms* comuns e até associações entre comentários (especificações em linguagem natural) e a sua implementação [0, 0].

2. **Compreensão de Contexto de Longo Alcance:** A arquitetura *Transformer*, através do seu mecanismo de *attention*, permite que o modelo pondere relações entre *tokens* distantes no texto de entrada. Isto é crucial para compreender um Diff de código, onde uma alteração numa linha pode ter implicações em funções ou módulos referenciados noutras partes do ficheiro [**vaswani2017attention**].

Estas capacidades transformam os LLMs de meros geradores de texto em ferramentas potencialmente capazes de atuar como *assistentes de programação inteligentes*, compreendendo a intenção por trás de alterações de código e avaliando-as num contexto mais amplo do que as ferramentas de análise estática baseadas em regras.

2.5 Modelos de Linguagem de Grande Escala Aplicados à Revisão de Código

A aplicação de LLMs à automação da revisão de código é uma área de investigação e desenvolvimento industrial em rápido crescimento. Estes modelos são utilizados para automatizar ou auxiliar várias sub-tarefas do processo de CR:

- **Geração de Descrições e Sumários:** Automatizar a criação de descrições concisas para PRs, resumindo as alterações realizadas, o que ajuda os revisores a compreender rapidamente o contexto [**codeagent2024**].

- **Deteção de Defeitos e *Code Smells*:** Identificar potenciais *bugs*, vulnerabilidades de segurança, más práticas de desempenho ou violações de princípios de design (como

SOLID) com base no contexto aprendido, superando em alguns casos a precisão de analisadores estáticos tradicionais para certos tipos de problemas [0, 0].

- **Geração de Comentários de Revisão:** Esta é a aplicação mais diretamente relacionada com o objetivo desta dissertação. Os LLMs analisam o Diff e geram comentários que podem variar desde sugestões de estilo (formatação, nomenclatura) até a questões de lógica, sugerindo implementações alternativas ou apontando casos extremos não tratados [0, 0].
- **Resposta a Comentários:** Alguns sistemas exploram a capacidade dos LLMs de atuar como autores, respondendo automaticamente a comentários dos revisores, esclarecendo decisões ou comprometendo-se com correções [**codeagent2024**].

Estudos e Resultados Chave: Trabalhos recentes demonstram o potencial desta abordagem. Por exemplo, estudos que utilizam modelos como o GPT-3.5/4 ou o CodeLLaMA mostraram que os comentários gerados podem ser considerados úteis por programadores numa percentagem significativa dos casos, por vezes equivalentes a comentários de revisores humanos para categorias específicas de problemas [0, 0]. No entanto, a literatura também aponta desafios consistentes: a utilidade é altamente variável, há uma tendência para gerar falsos positivos ou sugestões genéricas, e os modelos podem "alucinar", sugerindo problemas inexistentes ou correções incorretas [0, 0].

A eficácia destes sistemas depende criticamente da formulação do *prompt* (a instrução dada ao modelo), do contexto fornecido (o Diff, mas também potencialmente ficheiros relevantes, descrição do PR, etc.) e da existência de um mecanismo para guiar o modelo com o conhecimento específico da organização - uma lacuna que esta dissertação pretende colmatar.

2.6 Limitações atuais e Desafios

Apesar do potencial demonstrado, a integração prática de LLMs no processo de CR em ambientes empresariais específicos, como o da Processware, enfrenta vários desafios significativos:

1. **Falta de Contexto Organizacional Específico:** LLMs genéricos são treinados em código público e conhecimentos gerais. Não possuem conhecimento inerente sobre as **boas práticas internas**, convenções de *codebase*, padrões de arquitetura específicos ou regras de negócio de uma empresa. Um comentário genericamente correto pode ser irrelevante ou mesmo contrário aos padrões internos da organização [0, 0].
2. **Variabilidade e Inconsistência na Qualidade:** A utilidade dos comentários gerados pode flutuar amplamente. O modelo pode produzir uma análise perspicaz num caso e, no seguinte, gerar comentários vagos, incorretos ou focados em aspetos triviais, minando a confiança dos programadores [0, 0].

3. **Alucinações e Falsos Positivos:** LLMs podem identificar "problemas" que não existem, sugerir correções sintaticamente inválidas ou semanticamente erradas, ou atribuir incorretamente a causa de um defeito. Esta falta de fiabilidade exige uma supervisão humana cuidadosa, podendo anular os ganhos de eficiência [0, 0].

4. **Custo e Latência:** A execução de LLMs de grande porte para analisar cada PR pode ter custos computacionais e financeiros não triviais, além de introduzir latência no processo, especialmente para Diffs grandes ou análises complexas.

5. **Integração no Fluxo de Trabalho (*Workflow*):** Incorporar uma ferramenta baseada em LLMs de forma não intrusiva e que realmente agregue valor - em vez de gerar ruído - é um desafio de engenharia. A ferramenta deve complementar, e não substituir ou atrapalhar, o julgamento humano e a dinâmica colaborativa da revisão [0].

Este conjunto de limitações define claramente o espaço para a contribuição proposta nesta dissertação. O problema central não é *se* os LLMs podem gerar comentários, mas **como fazê-lo de forma consistente, alinhada com o contexto específico da Processware, e suficientemente fiável para ser integrada no fluxo de desenvolvimento da empresa**. A solução proposta, ao focar-se na personalização com base nas boas práticas internas e na integração fluida, aborda diretamente as lacunas 1 e 5, enquanto os objetivos de avaliação (Objetivo 5) visam mitigar os riscos apontados nas lacunas 2, 3 e 4.

3

Tecnologias

Este capítulo tem como objetivo apresentar as principais tecnologias relevantes para o desenvolvimento da solução proposta nesta dissertação. São abordadas as plataformas de controlo de versões e revisão de código, as ferramentas de revisão automática existentes e as abordagens baseadas em LLMs, bem como as tecnologias utilizadas para a extração, processamento e integração de alterações de código em sistemas reais.

3.1 Plataformas de Controlo de Versões e Revisão de Código

As plataformas de controlo de versões desempenham um papel central no desenvolvimento moderno de software, fornecendo mecanismos para gerir alterações de código, promover a colaboração entre programadores e suportar processos de revisão antes da integração de novas funcionalidades no produto em desenvolvimento. Atualmente, o sistema de controlo de versões distribuído *Git* é amplamente adotado na indústria e serve de base às principais plataformas de desenvolvimento colaborativo, suportando os fluxos de CR característicos do desenvolvimento moderno [0, 0].

Com base no *Git*, têm vindo a surgir plataformas que integram funcionalidades adicionais orientadas para a colaboração e revisão de código, como é o caso do *GitHub*, do *GitLab* e do *Bitbucket*. Estas plataformas introduziram o conceito de PRs (ou *merge request*), que permite submeter um conjunto de alterações para análise, discussão e validação antes da integração das alterações no ramo principal do repositório. Este modelo tornou-se um elemento central dos fluxos de desenvolvimento contemporâneos, particularmente em equipas distribuídas e ambientes empresariais de grande escala, estando associado a melhorias na qualidade do código e na deteção precoce de defeitos [0].

O processo de CR nestas plataformas é suportado por um conjunto de funcionalidades específicas, incluindo comentários *inline* associados a linhas ou blocos de código, mecanismos de aprovação ou rejeição de alterações e integração com sistemas automáticos de validação, como testes e analisadores estáticos. Estas capacidades permitem que os revisores forneçam feedback contextualizado e registem decisões de forma estruturada, promovendo a qualidade e rastreabilidade do processo de desenvolvimento.

Para além da interface de utilizador, estas plataformas disponibilizam interfaces de programação (Application Programming Interface (API)) e mecanismos de notificação, como *webhooks*, que permitem a integração de ferramentas externas no fluxo de CR. Através destas interfaces, é possível aceder programaticamente às alterações submetidas, obter o Diff associado a um PR e adicionar comentários de forma automática. Estas características tornam as plataformas de controlo de versões pontos naturais de integração para sistemas de apoio à CR baseados em inteligência artificial.

Neste contexto, a escolha de uma plataforma de controlo de versões não influencia apenas a gestão do código-fonte, mas também condiciona as possibilidades de automação e integração de soluções inteligentes no processo de CR. Assim, compreender as capacidades oferecidas por estas plataformas é essencial para o desenvolvimento de sistemas que visem apoiar ou automatizar a revisão de código de forma eficaz e alinhada com os fluxos de trabalho existentes.

3.2 Ferramentas de Revisão Automática Tradicionais

Com o aumento da complexidade dos sistemas de software e do volume de alterações submetidas para revisão, surgiram ferramentas de apoio automático ao processo de CR, com o objetivo de reduzir o esforço manual e melhorar a deteção precoce de problemas. Estas ferramentas baseiam-se, maioritariamente, em técnicas de análise estática de código, analisando o código-fonte sem necessidade de execução e recorrendo a um conjunto de regras ou heurísticas pré-definidas, com origem em abordagens clássicas de inspeção de software [0].

Entre as ferramentas mais utilizadas encontram-se os *linters* e analisadores estáticos, como o SonarQube, o ESLint, o PMD ou o Checkstyle, que são capazes de identificar violações de estilo, más práticas recorrentes, potenciais erros de execução e problemas de manutenibilidade. Estas ferramentas são frequentemente integradas em pipelines de CI, permitindo que os problemas detetados sejam sinalizados automaticamente durante o processo de desenvolvimento.

Uma das principais vantagens destas abordagens reside no seu comportamento determinístico e na capacidade de fornecer feedback rápido e consistente. As regras utilizadas são bem definidas, o que facilita a compreensão dos avisos gerados e a sua repetibilidade ao longo do tempo. Além disso, a integração destas ferramentas nos fluxos de desenvolvimento é, em geral, simples e bem suportada pelas plataformas de controlo de versões e CI.

No entanto, as ferramentas de revisão automática tradicionais apresentam limitações relevantes, especialmente em ambientes empresariais complexos. Estudos empíricos demonstram que a utilização de regras fixas dificulta a adaptação a contextos organizacionais específicos e pode originar um número elevado de avisos irrelevantes, levando frequentemente à sua desvalorização por parte dos programadores [0, 0].

Outra limitação significativa prende-se com a compreensão reduzida do contexto em que o código é desenvolvido. As ferramentas de análise estática não conseguem, em geral, captar a intenção do programador, avaliar decisões de design ou interpretar o impacto de uma alteração no contexto mais amplo do sistema. Assim, embora sejam eficazes na deteção de problemas bem definidos e repetitivos, revelam-se menos adequadas para fornecer comentários contextualizados e acionáveis durante o processo de CR.

Estas limitações motivaram a exploração de abordagens mais flexíveis e contextuais, capazes de compreender o código de forma mais abrangente e de gerar feedback alinhado com o contexto específico de cada projeto. As ferramentas baseadas em LLMs, discutidas na secção seguinte, surgem como uma resposta a estas necessidades.

3.3 Ferramentas baseadas em LLMs para Revisão de Código

3.3.1 Enquadramento Geral

Os LLMs baseiam-se em arquiteturas de atenção que permitem capturar dependências complexas em sequências de texto, tendo demonstrado resultados relevantes em múltiplas tarefas de engenharia de software [**vaswani2017attention**, **brown2020gpt3**, 0]. Estes modelos exploram regularidades estatísticas presentes no código-fonte, frequentemente descritas como a sua *naturalidade*, permitindo capturar padrões semânticos e estruturais relevantes [0, 0].

No contexto de CR, a utilização de LLMs permite ultrapassar algumas das limitações das abordagens tradicionais baseadas em regras fixas. Estes modelos são capazes de analisar alterações de código de forma mais flexível, considerando o contexto envolvente, a intenção do programador e a estrutura global do código. Como resultado, torna-se possível gerar comentários mais expressivos, próximos do feedback fornecido por revisores humanos, e apoiar tarefas como a deteção de erros lógicos, a identificação de problemas de legibilidade e a sugestão de melhorias de design.

A aplicação de LLMs à revisão de código tem sido explorada tanto em ambientes de investigação como em soluções comerciais, refletindo o interesse crescente em integrar estas tecnologias nos fluxos de desenvolvimento existentes [0]. No entanto, a diversidade de abordagens adotadas evidencia que o domínio se encontra ainda em evolução.

3.3.2 Assistentes de CR Comerciais

Nos últimos anos, várias soluções comerciais têm incorporado LLMs como forma de apoiar tarefas relacionadas com a revisão de código. Estas ferramentas são, em geral, integradas em plataformas de controlo de versões ou disponibilizadas como serviços externos, sendo capazes de analisar alterações submetidas em PRs e gerar comentários automáticos.

Estudos recentes indicam que estas ferramentas podem reduzir o esforço manual dos revisores e melhorar a eficiência do processo de CR, embora a qualidade e relevância

dos comentários gerados varie significativamente entre soluções [0, 0]. Em contextos industriais, observa-se que a falta de adaptação ao contexto organizacional e às práticas específicas de cada projeto constitui uma limitação relevante [0, 0].

3.3.3 Ferramentas Integradas em IDEs

Para além das soluções integradas em plataformas de controlo de versões, os LLMs têm sido amplamente explorados em ferramentas integradas em IDEs. Estas ferramentas fornecem feedback em tempo real durante o processo de escrita de código, incluindo sugestões de melhoria, explicações de código e deteção de potenciais problemas.

Apesar do impacto positivo na produtividade individual, estas soluções não substituem o processo colaborativo de revisão formal, uma vez que o seu foco se encontra predominantemente no apoio ao programador individual e não nos fluxos estruturados de validação coletiva [0].

3.3.4 Limitações das Soluções Existentes

Apesar do potencial demonstrado pelas ferramentas baseadas em LLMs, estudos empíricos evidenciam limitações relacionadas com a consistência do feedback, a previsibilidade do comportamento dos modelos e a sua integração nos fluxos de desenvolvimento existentes [0, 0]. Adicionalmente, questões de privacidade, custos computacionais e controlo do comportamento dos modelos continuam a constituir desafios relevantes em ambientes empresariais.

3.4 Tecnologias para Extração e Processamento de Alterações de Código

A análise automática de alterações de código requer o acesso estruturado às modificações submetidas pelos programadores. Este acesso é geralmente realizado através das interfaces disponibilizadas pelas plataformas de controlo de versões, que permitem obter informação detalhada sobre *commits*, ficheiros modificados e diferenças associadas a PRs.

As alterações são frequentemente representadas sob a forma de Diffs, embora possam ser transformadas em representações sintáticas, como ASTs, ou híbridas, dependendo dos objetivos da análise. A escolha da representação influencia diretamente a eficácia das técnicas aplicadas, conforme discutido na literatura sobre análise automática de código [0, 0].

3.5 Tecnologias para Integração de LLMs

A integração de LLMs em sistemas de apoio à revisão de código é normalmente realizada através de arquiteturas baseadas em serviços intermediários, responsáveis pela orquestração da extração de dados, processamento das alterações e geração de feedback automático.

Estas arquiteturas permitem maior flexibilidade e controlo sobre o comportamento do sistema [**codeagent2024**, 0].

No entanto, esta integração levanta desafios técnicos relacionados com latência, custos computacionais e robustez do sistema, exigindo mecanismos adicionais de controlo e monitorização [0].

3.6 Síntese e Considerações Finais

Neste capítulo foram apresentadas as principais tecnologias relevantes para o desenvolvimento de sistemas de apoio à revisão de código. Foram analisadas as plataformas de controlo de versões, as ferramentas tradicionais de revisão automática e as abordagens baseadas em LLMs.

Adicionalmente, foram discutidas as tecnologias necessárias para a extração, processamento e integração de alterações de código em sistemas reais. Este enquadramento tecnológico permitiu identificar limitações nas soluções existentes e fundamentar as decisões técnicas subjacentes à solução proposta, cuja arquitetura e implementação são apresentadas no capítulo seguinte.

4

Solução Proposta

Este capítulo apresenta a solução proposta para a integração de inteligência artificial no processo de revisão de código na Processware. É descrita a abordagem adotada, bem como a arquitetura geral do sistema, os seus principais componentes e a forma como a solução se integra no ciclo de vida de desenvolvimento do software existente.

O capítulo começa por apresentar uma visão geral da solução, seguida da definição dos requisitos e premissas que orientaram o seu desenho. São depois introduzidos os princípios arquiteturais e as decisões de design que sustentam a proposta, preparando o enquadramento para os capítulos seguintes, onde serão abordados os detalhes de implementação e a avaliação da solução desenvolvida.

4.1 Visão Geral

A solução proposta consiste num sistema de apoio à revisão de código baseado em modelos de linguagem de grande escala, concebido para analisar automaticamente alterações submetidas em PRs e gerar comentários de revisão alinhados com as boas práticas internas da Processware. O sistema atua como um assistente ao revisor humano, com o objetivo de reduzir o esforço associado a tarefas repetitivas, aumentar a consistência do feedback e melhorar a eficiência do processo de CR.

A arquitetura da solução baseia-se num serviço intermediário integrado com a plataforma de controlo de versões utilizada pela empresa, o Bitbucket. O sistema é acionado automaticamente aquando da criação ou atualização de uma PR, extraindo as alterações submetidas e aplicando um processo de pré-processamento que filtra e organiza o Diff de acordo com as tecnologias envolvidas.

As alterações processadas são analisadas por um motor baseado em LLMs, que combina a informação do código alterado com um conjunto formalizado de boas práticas internas. Este mecanismo permite gerar comentários de revisão em linguagem natural, focados em potenciais problemas, desvios a padrões definidos e oportunidades de melhoria. De forma a mitigar limitações associadas à utilização de LLMs, como a geração de falsos positivos, os comentários produzidos são sujeitos a uma fase de validação e filtragem antes da sua

publicação.

Os comentários finais são automaticamente publicados na PR, de forma integrada com o fluxo normal de revisão de código. Importa salientar que o sistema não substitui o julgamento humano nem interfere com o processo de aprovação das alterações, funcionando como um mecanismo complementar às ferramentas de análise estática existentes e à experiência dos revisores.

4.2 Requisitos e Condições de Utilização

O desenvolvimento da solução proposta tem por base um conjunto de requisitos funcionais e não funcionais, bem como um conjunto de condições de utilização associadas ao contexto organizacional e tecnológico da Processware. A definição destes elementos é fundamental para garantir que o sistema desenvolvido responde com eficácia ao problema identificado, mantendo a compatibilidade com os fluxos de desenvolvimento já existentes na empresa.

4.2.1 Requisitos Funcionais

Os requisitos funcionais descrevem as capacidades que o sistema deve com o objetivo de apoiar o processo de CR. Foram definidos os seguintes requisitos funcionais:

- **RF1 – Deteção automática de eventos de revisão:** O sistema deve ser capaz de detetar automaticamente a criação ou atualização de PRs nos repositórios da Processware, recorrendo aos mecanismos de notificação disponibilizados pela plataforma de controlo de versões.
- **RF2 – Extração de alterações de código:** O sistema deve obter programaticamente as alterações submetidas numa PR, incluindo o Diff, a lista de ficheiros modificados e informação contextual relevante, como a linguagem de programação utilizada.
- **RF3 – Análise automática das alterações:** O sistema deve analisar as alterações de código utilizando LLMs, com o objetivo de identificar potenciais problemas, desvios a boas práticas e oportunidades de melhoria.
- **RF4 – Integração de boas práticas internas:** O processo de análise deve ser guiado por um conjunto de boas práticas internas da Processware, garantindo que os comentários gerados estão alinhados com os padrões e convenções adotados pela organização.
- **RF5 – Geração de comentários de revisão:** O sistema deve gerar comentários de revisão claros, concisos e acionáveis, em linguagem natural, associados às alterações analisadas.
- **RF6 – Publicação automática de comentários:** Os comentários considerados relevantes devem ser automaticamente publicados na PR, de forma integrada com a interface da plataforma de controlo de versões.

- **RF7 – Apoio ao revisor humano:** O sistema deve atuar como um mecanismo de apoio, não substituindo o julgamento humano nem interferindo com o processo de aprovação final das PRs.

4.2.2 Requisitos Não Funcionais

Para além das funcionalidades descritas, a solução deve respeitar um conjunto de requisitos não funcionais, essenciais para a sua adoção num contexto empresarial. Assim sendo, foram definidos os seguintes requisitos não funcionais:

- **RNF1 – Integração transparente no fluxo de desenvolvimento:** A solução deve integrar-se de forma não intrusiva no processo existente de revisão de código.
- **RNF2 – Escalabilidade:** O sistema deve ser capaz de lidar com múltiplas PRs em paralelo.
- **RNF3 – Latência aceitável:** O tempo de resposta do sistema deve ser compatível com o processo de revisão de código.
- **RNF4 – Fiabilidade dos comentários gerados:** O sistema deve minimizar a geração de comentários irrelevantes ou incorretos.
- **RNF5 – Compatibilidade com ferramentas existentes:** A solução deve complementar ferramentas de análise estática já utilizadas, como o SonarLint.
- **RNF6 – Manutenibilidade e extensibilidade:** A arquitetura do sistema deve permitir a evolução futura da solução.

4.2.3 Condições de Utilização

O funcionamento adequado da solução proposta pressupõe um conjunto de condições de utilização que refletem no contexto organizacional e tecnológico em que o sistema irá ser aplicado:

- **CU1 – Existência de um processo formal de revisão de código:** Assume-se a utilização sistemática de PRs como mecanismo de revisão de código, com participação ativa de revisores humanos.
- **CU2 – Utilização de ferramentas de análise estática:** Parte-se do princípio de que ferramentas como o SonarLint são utilizadas durante o desenvolvimento, sendo responsáveis pela deteção de problemas sintáticos e violações de regras bem definidas.
- **CU3 – Supervisão humana dos comentários automáticos:** Os comentários gerados pelo sistema são sempre sujeitos à avaliação dos revisores humanos, que mantêm a responsabilidade final pela aceitação ou rejeição das alterações propostas.

- **CU4 – Disponibilidade de boas práticas internas:** Assume-se que as boas práticas da organização podem ser identificadas, formalizadas e mantidas de forma estruturada, permitindo a sua integração no sistema de análise.
- **CU5 – Ambiente tecnológico heterogéneo:** A solução deve operar num contexto caracterizado pela coexistência de múltiplas linguagens de programação e tecnologias, incluindo código moderno e código *legacy*.

4.3 Arquitetura Geral da Solução

A solução proposta adota uma arquitetura baseada numa aplicação externa, desenvolvida de forma independente da plataforma de controlo de versões, mas integrada com esta através de interfaces bem definidas. Esta abordagem permite um maior controlo sobre o fluxo de execução, facilita a evolução futura do sistema e reduz o acoplamento a tecnologias específicas.

A aplicação externa é responsável por orquestrar todo o processo de análise automática das PRs, desde a receção dos eventos até à publicação dos comentários de revisão. A integração com o Bitbucket é realizada exclusivamente através de mecanismos standard, como *webhooks* e interfaces de programação (APIs), garantindo uma comunicação desacoplada e robusta.

Esta decisão arquitetural permite que a solução seja executada como um serviço autónomo no ecossistema tecnológico da Processware, reutilizando componentes existentes, práticas de desenvolvimento consolidadas e mecanismos internos de monitorização e manutenção.

4.3.1 Visão Geral da Arquitetura

A arquitetura da solução pode ser conceptualizada como um conjunto de módulos cooperantes, organizados em torno de uma aplicação central de análise. O fluxo de execução inicia-se com a deteção de eventos associados a PRs, prossegue com a análise automática das alterações de código e termina com a publicação de comentários de revisão diretamente na plataforma de controlo de versões.

De forma resumida, a arquitetura é composta pelos seguintes elementos principais:

- Plataforma de controlo de versões (Bitbucket);
- Aplicação externa de análise de revisões;
- Motor de análise baseado em LLMs;
- Base de conhecimento de boas práticas internas.

Cada um destes elementos desempenha um papel específico no sistema, sendo descritos em maior detalhe nas subsecções seguintes.

4.3.2 Integração com a Plataforma de Controlo de Versões

A integração com o Bitbucket é realizada de forma indireta, através da configuração de *webhooks* que notificam a aplicação externa sempre que uma PR é criada ou atualizada. Estes eventos desencadeiam o processo de análise automática, sem necessidade de intervenção manual.

Após a receção do evento, a aplicação externa utiliza as APIs disponibilizadas pelo Bitbucket para obter informação detalhada sobre a PR, incluindo o Diff, os ficheiros modificados e metadados relevantes. Da mesma forma, a publicação dos comentários de revisão é efetuada através destas interfaces, assegurando uma integração transparente para os utilizadores finais.

Esta abordagem permite manter a plataforma de controlo de versões desacoplada da lógica de análise, reduzindo dependências e facilitando a manutenção do sistema.

4.3.3 Aplicação Externa de Análise

A aplicação externa constitui o núcleo da solução proposta, sendo responsável por coordenar todas as etapas do processo de análise. Esta aplicação é desenvolvida com tecnologias já utilizadas pela Processware, assegurando consistência com o ecossistema tecnológico existente e facilitando a sua adoção interna.

Entre as principais responsabilidades da aplicação externa destacam-se:

- Gestão dos eventos recebidos da plataforma de controlo de versões;
- Extração e pré-processamento das alterações de código;
- Orquestração do processo de análise baseado em LLMs;
- Aplicação de mecanismos de validação e filtragem dos resultados;
- Publicação dos comentários finais nas PRs.

A separação destas responsabilidades permite uma maior modularidade e contribui para a manutenibilidade e extensibilidade da solução.

4.3.4 Motor de Análise e Base de Conhecimento

O motor de análise é responsável por avaliar as alterações de código submetidas numa PR, combinando o conteúdo do Diff com um conjunto formalizado de boas práticas internas da Processware. Esta base de conhecimento inclui convenções de codificação, regras arquiteturais e recomendações específicas do contexto organizacional.

A utilização de LLMs permite interpretar as alterações de forma contextual, indo além da deteção de problemas puramente sintáticos. No entanto, reconhecendo as limitações inerentes a estes modelos, os resultados produzidos são sujeitos a mecanismos adicionais de validação, com o objetivo de reduzir a geração de comentários irrelevantes ou incorretos.

Esta combinação entre análise baseada em IA e conhecimento organizacional constitui um dos principais contributos da solução proposta.

4.3.5 Considerações Arquiteturais

A adoção de uma aplicação externa como elemento central da arquitetura permite satisfazer os requisitos definidos na Secção 4.2, nomeadamente no que respeita à escalabilidade, manutenibilidade e integração com ferramentas existentes. Adicionalmente, esta abordagem facilita a evolução futura da solução, permitindo a sua adaptação a novas plataformas de controlo de versões ou a diferentes contextos organizacionais sem alterações significativas à arquitetura base.

4.4 Fluxo de Execução do Sistema

Esta secção descreve o fluxo de execução da solução proposta, detalhando as etapas envolvidas desde a criação ou atualização de uma PR até à disponibilização dos comentários de revisão gerados automaticamente. A apresentação sequencial do fluxo permite clarificar a interação entre os diferentes componentes da arquitetura e evidenciar o papel de cada um no processo global de análise.

4.4.1 Início do Processo

O fluxo de execução inicia-se quando um programador cria ou atualiza uma PR na plataforma de controlo de versões. Este evento é automaticamente comunicado à aplicação externa através de um *webhook* previamente configurado. O *webhook* contém informação essencial sobre a PR, permitindo à aplicação identificar o repositório, o identificador da revisão e o tipo de evento ocorrido.

Após a receção do evento, a aplicação externa valida a sua autenticidade e verifica se a PR cumpre os critérios necessários para ser analisada, como o tipo de projeto ou as tecnologias envolvidas. Caso estas condições não sejam satisfeitas, o processo é interrompido.

4.4.2 Extração e Pré-processamento das Alterações

Uma vez validado o evento, a aplicação externa utiliza as APIs do Bitbucket para obter as alterações de código associadas à PR. Esta informação inclui o Diff completo, a lista de ficheiros modificados e metadados relevantes.

As alterações obtidas são submetidas a uma fase de pré-processamento, cujo objetivo é preparar o código para análise automática. Nesta fase são aplicadas operações como a filtragem de ficheiros irrelevantes, a normalização do formato do Diff e a identificação das linguagens de programação envolvidas. Este passo é particularmente relevante em projetos heterogéneos, onde coexistem múltiplas tecnologias.

4.4.3 Análise Baseada em Inteligência Artificial

Após o pré-processamento, as alterações estruturadas são encaminhadas para o motor de análise baseado em LLMs. Nesta etapa, o sistema constrói um contexto de análise que combina o código alterado com um conjunto de boas práticas internas da Processware, previamente formalizadas.

O motor de análise avalia as alterações com base neste contexto, identificando potenciais problemas, desvios a padrões definidos e oportunidades de melhoria. O resultado deste processo consiste num conjunto de comentários de revisão em linguagem natural, associados a locais específicos do código sempre que possível.

4.4.4 Validação e Filtragem dos Resultados

Reconhecendo as limitações inerentes à utilização de LLMs, os comentários gerados são submetidos a uma fase adicional de validação e filtragem. Esta etapa tem como objetivo reduzir a ocorrência de falsos positivos, eliminar comentários redundantes e garantir que apenas feedback relevante e acionável é apresentado aos revisores.

Os comentários podem ser classificados de acordo com a sua natureza, por exemplo, problemas funcionais, questões de legibilidade ou sugestões de melhoria, permitindo uma apresentação mais estruturada do feedback final.

4.4.5 Publicação dos Comentários

Após a validação, os comentários finais são publicados automaticamente na PR através das APIs do Bitbucket. Dependendo da natureza do comentário, este pode ser associado diretamente a uma linha específica do código ou apresentado como um comentário geral da revisão.

A publicação dos comentários ocorre de forma transparente para os utilizadores, integrando-se no fluxo normal de revisão de código e permitindo que os revisores humanos avaliem, aceitem ou rejeitem as sugestões apresentadas.

4.4.6 Interação com o Revisor Humano

O fluxo de execução termina com a interação do revisor humano com os comentários gerados pelo sistema. O revisor mantém sempre a responsabilidade final sobre a aceitação das alterações submetidas, podendo utilizar os comentários automáticos como apoio à sua análise.

É importante salientar que o sistema não impõe bloqueios nem decisões automáticas, funcionando como um mecanismo complementar às ferramentas de análise estática e à experiência dos revisores. Esta abordagem garante a adoção gradual da solução e contribui para a manutenção da confiança dos utilizadores no processo de revisão de código.

4.5 Desenho Detalhado dos Componentes

Esta secção descreve de forma detalhada os principais componentes da solução proposta, bem como as respetivas responsabilidades e interações. O objetivo é clarificar a decomposição interna do sistema, evidenciando como cada componente contribui para o fluxo de execução apresentado na Secção 4.4.

A solução foi desenhada segundo princípios de modularidade e separação de responsabilidades, permitindo facilitar a manutenção, a evolução futura e a reutilização de componentes em diferentes contextos.

4.5.1 Componente de Integração com a Plataforma de Controlo de Versões

O componente de integração com a plataforma de controlo de versões é responsável por toda a comunicação entre a aplicação externa e o Bitbucket. Este componente gere a receção de eventos provenientes de *webhooks*, assegurando a validação da sua origem e a correta interpretação dos dados recebidos.

Adicionalmente, este componente encapsula o acesso às APIs do Bitbucket, sendo responsável pela obtenção das alterações associadas às PRs e pela publicação dos comentários de revisão gerados pelo sistema. Esta abordagem permite isolar dependências externas, reduzindo o impacto de alterações futuras na plataforma de controlo de versões.

4.5.2 Componente de Gestão de Eventos

O componente de gestão de eventos atua como ponto de entrada do sistema, sendo responsável por coordenar a execução do fluxo de análise após a receção de um evento válido. Este componente determina se uma PR deve ser analisada, com base em critérios previamente definidos, como o tipo de projeto, o estado da revisão ou as tecnologias envolvidas.

Ao centralizar esta lógica, o sistema garante um controlo consistente sobre quando e como a análise automática é desencadeada, evitando execuções desnecessárias e contribuindo para a eficiência global da solução.

4.5.3 Componente de Pré-processamento

O componente de pré-processamento tem como principal função preparar as alterações de código para análise automática. Este componente aplica operações como a filtragem de ficheiros irrelevantes, a normalização do formato do Diff e a identificação das linguagens de programação presentes na PR.

Este passo é particularmente relevante em projetos empresariais complexos, onde coexistem múltiplas tecnologias e diferentes estilos de codificação. O pré-processamento contribui para melhorar a qualidade da análise subsequente e para reduzir ruído nos resultados gerados pelo sistema.

4.5.4 Componente de Orquestração da Análise

O componente de orquestração da análise é responsável por coordenar a interação entre os dados pré-processados, a base de conhecimento de boas práticas internas e o motor de análise baseado em LLMs. Este componente constrói o contexto de análise, garantindo que a informação relevante é corretamente estruturada e apresentada ao motor de IA.

Para além disso, este componente gere aspetos como a segmentação das alterações em unidades analisáveis, o controlo do volume de informação enviado ao modelo e a consolidação dos resultados obtidos. Esta responsabilidade é essencial para assegurar uma utilização eficiente e controlada dos LLMs.

4.5.5 Motor de Análise Baseado em LLMs

O motor de análise constitui o núcleo inteligente da solução proposta. Este componente utiliza LLMs para interpretar as alterações de código de forma contextual, indo além da análise sintática tradicional. Através da combinação entre o código alterado e as boas práticas internas da Processware, o motor é capaz de gerar comentários de revisão em linguagem natural.

Apesar das capacidades avançadas deste componente, o sistema foi desenhado de forma a não depender exclusivamente dos resultados produzidos pelo modelo, reconhecendo as limitações associadas à utilização de IA generativa em contextos críticos.

4.5.6 Componente de Validação e Filtragem

O componente de validação e filtragem tem como objetivo garantir a qualidade dos comentários gerados pelo motor de análise. Este componente aplica regras adicionais para eliminar comentários redundantes, identificar possíveis falsos positivos e priorizar feedback com maior relevância para os revisores humanos.

A existência deste componente é fundamental para manter a confiança dos utilizadores na solução, evitando a sobrecarga de informação e assegurando que os comentários apresentados são claros, úteis e acionáveis.

4.5.7 Base de Conhecimento de Boas Práticas Internas

A base de conhecimento de boas práticas internas armazena um conjunto formalizado de regras, convenções e recomendações específicas da Processware. Este componente permite capturar conhecimento organizacional que, de outra forma, estaria disperso ou implícito na experiência dos programadores mais experientes.

A separação desta base de conhecimento do motor de análise permite a sua evolução independente, facilitando a atualização das boas práticas sem necessidade de alterações significativas na lógica do sistema.

4.5.8 Componente de Publicação de Resultados

O componente de publicação de resultados é responsável por disponibilizar os comentários finais aos utilizadores, através da plataforma de controlo de versões. Este componente assegura que os comentários são corretamente associados às linhas de código relevantes ou apresentados como comentários gerais da PR.

Ao encapsular esta funcionalidade num componente dedicado, o sistema garante uma apresentação consistente do feedback e facilita a adaptação futura a diferentes plataformas de controlo de versões.

4.5.9 Síntese do Desenho de Componentes

A decomposição da solução em componentes bem definidos permite satisfazer os requisitos identificados na Secção 4.2, promovendo a modularidade, a manutenibilidade e a extensibilidade do sistema. Esta abordagem facilita também a transição para a fase de implementação, descrita nos capítulos seguintes, onde cada componente poderá ser mapeado para tecnologias e ferramentas concretas utilizadas pela Processware.

4.6 Considerações de Implementação e Decisões Técnicas

O desenho da solução proposta foi fortemente influenciado por um conjunto de decisões técnicas e considerações práticas, decorrentes tanto das características do problema como do contexto tecnológico da Processware. Esta secção apresenta as principais opções tomadas ao nível da implementação, bem como a sua fundamentação.

Uma das decisões estruturais mais relevantes foi a adoção de uma aplicação externa e independente da plataforma de controlo de versões. Em vez de integrar diretamente a lógica de análise baseada em inteligência artificial no Bitbucket, optou-se por desenvolver um serviço autónomo, capaz de interagir com a plataforma através de *webhooks* e APIs. Esta abordagem permite reduzir o acoplamento com sistemas externos, facilitar a evolução futura da solução e reutilizar o sistema em diferentes contextos ou plataformas de controlo de versões.

Do ponto de vista arquitetural, esta separação contribui também para uma maior flexibilidade na escolha das tecnologias utilizadas, permitindo alinhar a implementação com o *stack* tecnológico já adotado pela Processware, nomeadamente no desenvolvimento de serviços backend em C# e na construção de interfaces de apoio através de tecnologias web. Além disso, a existência de uma aplicação dedicada facilita a monitorização, o controlo de custos associados à utilização de LLMs e a introdução de mecanismos de tolerância a falhas.

Outra consideração relevante prende-se com a integração de ferramentas de análise estática já utilizadas no processo de desenvolvimento, como o SonarLint. A solução proposta não pretende substituir estas ferramentas, mas sim complementá-las. Enquanto o SonarLint é eficaz na deteção de problemas bem definidos e determinísticos, o sistema

baseado em LLMs foca-se na análise contextual e na geração de comentários mais expressivos, alinhados com as boas práticas internas. Esta complementaridade permite maximizar os benefícios de ambas as abordagens.

No que diz respeito à utilização de LLMs, foi considerada a necessidade de mitigar limitações conhecidas, como a geração de falsos positivos ou comentários genéricos. Por esse motivo, a arquitetura inclui componentes específicos para pré-processamento, validação e filtragem dos resultados, assegurando que apenas comentários relevantes e acionáveis são apresentados aos revisores humanos. Esta decisão reflete a preocupação em preservar a confiança dos utilizadores e evitar a introdução de ruído no processo de CR.

Adicionalmente, foram tidas em conta questões relacionadas com desempenho e escalabilidade. A segmentação das alterações de código, o controlo do volume de informação enviado ao motor de análise e a possibilidade de executar análises de forma assíncrona permitem adaptar a solução a diferentes cargas de trabalho, evitando impactos negativos no fluxo de desenvolvimento.

Por fim, a solução foi concebida com foco na extensibilidade e manutenção a longo prazo. A separação entre a base de conhecimento de boas práticas internas e o motor de análise permite atualizar regras e orientações sem necessidade de alterações profundas no sistema. Esta característica é particularmente relevante em ambientes empresariais dinâmicos, onde padrões e convenções evoluem ao longo do tempo.

Em conjunto, estas considerações e decisões técnicas asseguram que a solução proposta é não apenas funcional do ponto de vista conceptual, mas também viável, sustentável e alinhada com as necessidades reais da Processware. No capítulo seguinte será apresentada a implementação concreta da solução, detalhando as tecnologias utilizadas e a forma como cada componente foi materializado.

Bibliografia

[0] M. Allamanis et al. «A Survey of Machine Learning for Big Code and Naturalness». Em: *ACM Computing Surveys* 51.4 (2018), pp. 1–37 (ver pp. 7, 8, 14, 15).

[0] A. Bacchelli e C. Bird. «Expectations, Outcomes, and Challenges of Modern Code Review». Em: *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), pp. 712–721 (ver pp. 6, 7, 12).

[0] X. Chen et al. «Large Language Models for Software Engineering: A Survey». Em: *arXiv* (2023). URL: https://arxiv.org/abs/2312.15223 (ver pp. 1, 3, 9, 14).

[0] M. E. Fagan. «Design and Code Inspections to Reduce Errors in Program Development». Em: *IBM Systems Journal* 15.3 (1976), pp. 182–211 (ver pp. 6, 13).

[0] A. Hindle et al. «On the Naturalness of Software». Em: *Proceedings of the 34th International Conference on Software Engineering (ICSE)* (2012), pp. 837–847 (ver pp. 8, 14).

[0] B. Johnson et al. «Why Don't Software Developers Use Static Analysis Tools to Find Bugs?» Em: *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (2013), pp. 672–681 (ver pp. 7, 13).

[0] S. Kim e M. D. Ernst. «Which Warnings Should I Fix First?» Em: *Proceedings of the 6th Joint Meeting on Foundations of Software Engineering (FSE)* (2016), pp. 45–55 (ver pp. 7, 8, 13).

[0] D. Liang et al. «Code Review Automation: Strengths and Weaknesses». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2401.05136 (ver pp. 1–3, 7, 10, 15, 16).

[0] S. McIntosh et al. «An Empirical Study of the Impact of Modern Code Review Practices on Software Quality». Em: *Empirical Software Engineering* 19.6 (2014), pp. 1819–1850 (ver pp. 6, 7, 12).

[0] T. Nguyen et al. «A Comparison of LLM-Assisted Development Tools». Em: *arXiv* (2025). URL: https://arxiv.org/abs/2501.00000 (ver pp. 1, 11).

[0] F. Rahman et al. «An Empirical Study of LLM-Based Automated Code Review». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2412.18531 (ver pp. 1–3, 10, 15).

[0] G. Research. «AI-Assisted Assessment of Coding Practices in Industrial Code Review». Em: *Google Publications* (2024). URL: https://research.google/pubs/ai-assisted-assessment-of-coding-practices-in-industrial-code-review/ (ver pp. 3, 10, 15).

[0] P. C. Rigby e C. Bird. «Convergent Contemporary Software Peer Review Practices». Em: *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)* (2013), pp. 202–212 (ver pp. 6, 12).

[0] M. Tufano et al. «Deep Learning Similarities from Different Representations of Source Code». Em: *Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2019) (ver pp. 8, 15).

[0] M. Tufano et al. «Large Language Models in Software Engineering: A Systematic Literature Review». Em: *ACM TOSEM* (2024). URL: https://nzjohng.github.io/publications/papers/tosem2024_5.pdf (ver pp. 1–3, 9, 10, 14).

[0] Y. Xie et al. «Rethinking Code Review Workflows with LLM Assistance». Em: *arXiv* (2025). URL: https://arxiv.org/html/2505.16339v1 (ver pp. 2, 3, 11, 15, 16).

[0] L. Zhang et al. «AI-powered Code Review with LLMs: Early Results». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2404.18496 (ver pp. 3, 10, 15).

[0] K. Zhong et al. «Evaluating AI-Assisted Code Review in Industry». Em: *arXiv* (2024). URL: https://arxiv.org/abs/2508.18771 (ver pp. 1–3, 10, 11).

A

Appendix 1

B

Appendix 2

I

Anexo 1