Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Inteligência Artificial como Coautora: A Nova Era da Engenharia de Software

Ricardo Manuel Lopes Pereira

Orientador(es) | Vitor Beires Nogueira

Pedro Salgueiro

Évora 2026

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Inteligência Artificial como Coautora: A Nova Era da Engenharia de Software

Ricardo Manuel Lopes Pereira

Orientador(es) | Vitor Beires Nogueira

Pedro Salgueiro

Évora 2026

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Lígia Maria Ferreira (Universidade de Évora)
Vogais	|	Paulo Miguel Quaresma (Universidade de Évora) (Arguente) Vitor Beires Nogueira (Universidade de Évora) (Orientador)

Évora 2026

Agradecimentos

Tenho de agradecer de uma forma que nunca será totalmente compensatória a toda a minha família por servirem de estrutura para que possa fazer o que faço, seja isso o que for. Como sempre, a crença e o incentivo são peças fundamentais para seguir o meu caminho.
Tenho e preciso de dedicar este trabalho à Carmen, ao Afonso e à Maria, minha esposa e filhos, por todas as cedências e esforços que fazem para permitir que eu possa cumprir com estes desafios.
As pessoas que me acompanham na minha vida são peças essenciais para cumprir com os objetivos. Joaquim, Júlia, Margarida, Ruben, Paulo, Carla, Martim, João, Sónia, Fred, Matilde, Bruno, Vânia e demais filhos e chegados, obrigado por fazerem esta viagem ao meu lado e por estarem aqui quando é preciso.
Nada é possível sem o eco que representa aquilo que pretendo ser e atingir, que é deixado nos tempos e para sempre pelos exemplos dos que partiram fisicamente. Ao meu avô, ao meu tio, à minha sogra, ao meu sogro, todo e qualquer desafio superado é parte de vós por todo o exemplo e motivação que me deram e continuam a dar. Obrigado e um até já.

Resumo

Os paradigmas de engenharia e desenvolvimento de software estão, atualmente, a ser completamente redesenhados derivado da inclusão da Inteligência Artificial nos seus processos. O objetivo desta dissertação centra-se em evidenciar a importância e valor cada vez mais notórios da IA como uma coautora no fluxo de implementação de software, fazendo uma análise cuidada da forma como os modelos de linguagem atuais, agentes e ferramentas de desenvolvimento de software assistido estão a interferir e a mudar o contexto do universo da programação. Com o recurso a uma revisão sistémica do estado da arte, a apresentação de um caso de estudo com a ferramenta Github Copilot, um caso prático com essa mesma ferramenta e um caso prático com a ferramenta OutSystems Mentor App Generator, pretende-se demonstrar que a Inteligência Artificial vai mais longe do que apenas automatização de rotinas e tarefas repetitivas, contribuindo também de forma bastante eficiente e segura para decisões de alguma complexidade técnica, ao ponto de gerar código por si só, gerar modelos de arquitetura para as soluções a implementar e, também, apoiar de forma robusta a componente de suporte e resolução de problemas.
Pretende-se com o trabalho exposto neste documento, salientar uma transição evidente e fundamental no modo como se pensa a programação, centrada no ser humano, na pessoa com inteligência orgânica e natural, para uma abordagem mais colaborativa no par homem-máquina, onde podemos contar com os sistemas inteligentes artificiais para servirem de copilotos com capacidade para compreender e interpretar requisitos transmitidos em linguagem natural e transforma-los em soluções totalmente ou quase totalmente funcionais.

Palavras-chave

Inteligência Artificial, Engenharia de Software, Desenvolvimento Assistido, Automação, Modelos de Linguagem

Abstract

Title EN: *Artificial Intelligence as a Co-author: The New Era of Software Engineering*

The paradigms of software engineering and development are currently being completely redesigned as a result of the inclusion of Artificial Intelligence in their processes. The objective of this dissertation is to highlight the increasingly evident importance and value of AI as a co-author in the implementation flow of digital solutions, providing a careful analysis of how modern language models, agents, and AI-assisted software development tools are influencing and transforming the context of the programming universe. Through a systematic review of the state of the art, the presentation of a case study using the Github Copilot tool, a practical case using the same tool, and a practical case using the OutSystems Mentor App Generator tool, this study aims to demonstrate that Artificial Intelligence goes beyond simply automating routines and repetitive tasks. It also contributes efficiently and securely to decisions of some technical complexity, to the point of generating code on its own, generating architectural models for the solutions to be implemented, and robustly supporting the problem-solving component.

The work presented in this document seeks to emphasize a clear and fundamental transition in the way programming is conceived: from a human-centered perspective, grounded in organic and natural intelligence, to a more collaborative human–machine approach. In this new paradigm, artificial intelligent systems can act as copilots capable of understanding and interpreting requirements expressed in natural language and transforming them into fully or nearly fully functional solutions.

Keywords

Artificial Intelligence, Software Engineering, Computer-Aided Development, Automation, Language Models

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução e Motivação

1.1 Objetivos do trabalho

Este trabalho pretende analisar de forma sistemática e crítica o impacto que a IA tem como coautora nos processos de desenvolvimento de software, salientando os seguintes pontos:

- Demonstrar a transformação dos paradigmas tradicionais da Engenharia de Software com a integração da IA em todo o ciclo de vida da implementação de um dado software.
- Analisar o impacto da IA num conjunto de métricas como produtividade e a qualidade de código de forma a identificar os prós e os contras da solução.
- Extrapolar os desafios éticos e legais que se associam ao processo de coautoria com um agente não humano.
- Analisar e documentar a evolução de competências de um engenheiro de software com base no paradigma tradicional direcionando-se para o paradigma atual.
- Demonstrar casos reais e práticos como o GitHub Copilot e o OutSystems Mentor App Generator de forma a obter uma compreensão empírica do tema em estudo desta dissertação.
- Analisar e explorar o que poderá ser o futuro nesta área, tendo em conta o grau de maturidade e o nível de confiança existente entre humanos e a IA.

1.2 Contexto geral do conceito de Inteligência Artificial nos processos de desenvolvimento de software

A Inteligência Artificial (IA), como qualquer conceito universal que amadurece ao longo do tempo, deixou os limites conhecidos até um passado recente para estravazar a cerca que delimitava a sua própria origem: começamos a ver então, a obra, de certa forma, tornar-se o criador (ou pelo menos parte criador). Deixamos de ter a IA como uma simples ferramenta de apoio e passamos a permitir que a mesma partilhe o lugar do condutor em conjunto com nós mesmos: surge a IA como coautora nos processos de desenvolvimento associados à Engenharia de Software. Esta infiltração da tecnologia na tecnologia redefine por completo o paradigma associado à forma como nascem e crescem as soluções digitais, permitindo a todos aqueles que assumem o perfil de engenheiro de software trabalhar melhor, largando o conceito de único artífice do código permitindo assim assumir uma posição mais centrada na arquitetura e na orquestração de sistemas colaborativos. O crescimento e domínio deste novo paradigma no universo da Engenharia de Software, onde humanos e máquinas cooperam na autoria dos ditos softwares, é o foco da investigação deste trabalho, que, com um estudo detalhado e aprofundado sobre as implicações técnicas, metodológicas, éticas e legais, pretende trazer à luz do dia todas as evidências e detalhes desta nova era.

A evolução da Engenharia de Software tem acontecido de forma contínua, começando por abraçar as metodologias de gestão de projeto em cascata (conhecido geralmente na literatura por modelos *waterfall*) até chegarmos às metodologias iterativas como Agile, funções como o DevOps e, mais recentemente, MLOps. O nascer dos modelos de linguagem de grande escala (os tão conhecidos LLMs) reforçou a influência da IA no universo de Engenharia de Software. Atualmente podem-se contar já com inúmeras ferramentas conhecidas, tal como o GitHub CoPilot, Claude, Amazon CodeWhisperer e o ChatGPT, que estão a transformar o ato de programar em algo muito mais colaborativo e menos dependente do programador humano [Rao+25].

Neste trabalho pretende-se demonstrar e caracterizar uma base de análise fundamental para o leitor compreender de uma forma coerente a transformação que está a ocorrer na disciplina de Engenharia de Software com a introdução da Inteligência Artificial, tal como mencionado anteriormente. Pretende-se que, durante a leitura, rapidamente se identifiquem os paralelismos entre utilização da IA e ajuste às necessidades dos utilizadores, o que confere a característica predominante da

adaptabilidade necessária para que a IA consiga evoluir de um contexto meramente auxiliar para uma vertente muito mais colaborativa e responsável no desenvolvimento de software.

Conforme estudado por Amasanti [AJ25], apesar do aumento de produtividade provocado pela adoção da IA nos processos de desenvolvimento de software, surge uma outra preocupação a ter em conta: a qualidade do produto entregue, principalmente no que diz respeito a geração de código e rastreamento de erros (*debugging*). Além dos problemas de qualidade de código e rastreamento de erros identificáveis numa fase mais inicial, existem problemas que podem ser identificados numa fase posterior e que, apesar de não perturbar o funcionamento atual das soluções após entrada em produção, pode retirar escalabilidade às mesmas: problemas de arquitetura. Quanto mais complexo for o ecossistema, com definição de domínios mais minuciosa e de composição mais detalhada, maior pode ser a tendência para um sistema de IA decompor os componentes arquitetónicos de forma errada. Isto pode levar a um aumento de dívida técnica e, caso não seja gerido devidamente pelo arquiteto/engenheiro de software pode-se tornar catastrófico. Estas preocupações levantam a questão que se irá abordar no trabalho relativamente até que ponto a IA pode ter autonomia em cada tarefa que participa.

Relativamente a competências, compreende-se que existe a necessidade de obtenção de outro tipo das mesmas por parte dos engenheiros de software, sendo que, passam a poder navegar por outros contextos já existentes nos quais não tinham grande preocupação, mas também em novos cenários que contemplam a participação da IA, tal como supervisão e *prompt engineering*. É necessário que um engenheiro de software consiga interpretar os resultados produzidos pelos modelos de IA. Estas características reforçam a confiabilidade, segurança e cumprimento de valores éticos por parte das soluções que são implementadas em parceria com ferramentas inteligentes. Começa a surgir também uma diluição da componente de código para a componente de dados, sendo que serão esses que irão complementar os algoritmos de IA que suportam as ferramentas [AA25].

Conforme já referido, a IA funciona com dois componentes de peso relevante: os algoritmos e os dados. No meio de toda a complexidade do seu funcionamento, em determinado ponto, a equipa de Engenharia de Software vai ter de conseguir compreender o raciocínio técnico extrapolado das operações realizadas pela IA de forma a garantir que faz sentido no contexto real e que cumpre com todos os requisitos. A isso denomina-se transparência, juntamente com a XAI (*eXplainable AI*: uma área de estudo que se foca na compreensão e interpretação das decisões tomadas pelos modelos de aprendizagem automática [Ays+25]) e é algo que nunca se deve perder para se poder manter o controle sobre a implementação [Akh+24].

Quando se fala no impacto da IA na Engenharia de Software não se pode contextualizar essa influência apenas no que à produção de código diz respeito: as próprias metodologias de gestão de projetos de software (ágeis e não ágeis) passam a remeter decisões e tarefas automáticas para o novo parceiro digital, libertando as pessoas para que estas se possam focar na criatividade e na elaboração de soluções que realmente remetem valor para os clientes e utilizadores. Os fluxos de trabalho conhecidos durante as diferentes fases de implementação de projetos de software estão em mutação de forma a incorporar a IA, permitindo assim um ganho de rendimento e produtividade em larga escala.
O ciclo de vida do software tem vindo a sofrer enormes alterações com a penetração da IA, sendo que a mesma não surge apenas para executar, mas também para interpretar e colaborar em todo o processo:

- **Definição de requisitos**: capacidade de interpretar a linguagem natural para gerar as especificações;
- **Design**: sugestão e implementação de diagramas e de artefactos arquitetónicos;
- **Desenvolvimento**: capacidade de geração de código nativo, capacidade de *scaffolding* e de refatorização;
- **Testes**: escrita automática de scripts de testes e validação dos resultados da execução dos mesmos;
- **Manutenção**: capacidade de auto-deteção de bugs e erros, sugestões de melhoria e criação de documentação de forma assistida.

Todas estas capacidades salientam a necessidade de criar ou reforçar técnicas de gestão de processos de implementação mais robustas baseadas em versionamento, rastreabilidade e validação efetuada de forma contínua.
Há sempre que ter cuidado com os contornos potencialmente negativos ou que podem gerar desagradado numa sociedade que se move, atualmente, de forma desigual e com reforço constante para o enviesamento [Nob18]. Existem pesquisas sobre o impacto que a IA pode ter na sociedade, abordando temas como a ética, principalmente no que diz respeito à substituição do ser humano na disciplina de Engenharia de Software. Há quem defenda, como Vaithilingam [Vai22], que não irá ocorrer uma substituição por completo do programador, mas sim uma parceria entre homem e IA que resultará num processo de coautoria, reformatando por completo os processos e paradigmas de implementação de soluções de software. O desenvolvimento de software com recurso à IA deve contemplar os seguintes princípios éticos [Des23]:

- **Capacitação dos humanos**: A Inteligência Artificial deve existir sempre de forma a manter e respeitar os direitos e dignidade humana, além de que deve ser mantido o controle pelos humanos.
- **Responsabilização**: Os responsáveis pelas IAs utilizadas deverão ser as empresas que as utilizam, mesmo que os sistemas sejam desenvolvidos por terceiros, durante todo o ciclo de vida de todo o sistema.
- **Controle humano**: Deve ser garantida uma supervisão completa e adequada por parte de humanos de forma a garantir controle sobre a solução implementada. Em nenhum momento a IA deve ser 100% autónoma.
- **Minimização do enviesamento e justiça**: Deve existir um esforço ao nível de verificação e monitorização de forma a evitar e corrigir qualquer tipo de preconceito que possa gerar descriminação, seja ele ao nível algorítmico ou de dados.
- **Privacidade e *security by design***: As leis de proteção de dados (conhecidas em Portugal por RGPD - Regulamento Geral sobre a Proteção de Dados [Uni16]) e a segurança integrada logo na conceção devem ser levadas em conta para utilização da IA e para a implementação de software com apoio da IA.
- **Transparência e explicabilidade**: Deve ser claro e fácil de compreender de que forma a IA é utilizada, quais os raciocínios que a mesma executa, quais as limitações e dados subjacentes, tal como os resultados obtidos das suas ações.

Essa coautoria levanta outra questão por si só: como deve ser gerida a propriedade intelectual? Quem é que deve ser identificado como autor de um determinado artefacto, seja ele código ou outro componente qualquer? Quem é verdadeiramente o proprietário do produto resultante dos algoritmos de IA? E, porque não, identificar de quem é a responsabilidade sobre o impacto das criações da IA? Quem deve assumir o erro ou a falha caso aconteça? Segundo Ankita Tiwari [Tiw25], essa responsabilidade nunca é atribuída à IA mas sim distribuída por vários intervenientes, dependendo do tipo de erro ou falha:

- **Equipa/empresa de desenvolvimento**: no caso de entrega de um produto defeituoso, com bugs ou falhas no design, a responsabilidade recaia sobre estes.
- **Fornecedores de dados**: a IA aprende com os dados. Se os dados fornecidos forem incorretos ou enviesados, a responsabilidade recai sobre estes.

- **Utilizadores finais**: Caso um utilizador recorra à IA de forma incorreta e com o intuído de prejudicar ou fazer o mal, a responsabilidade recai sobre estes.
- **Entidades reguladoras**: Sempre que se encontre perante uma regulamentação ou lei desajustada ou desatualizada, a responsabilidade recai sobre estas entidades, já que lhes cabe, além da criação das regras e regulamentos, a supervisão e manutenção das mesmas.

Um dos temas que muitas das vezes é esquecido por ser uma corrente recente prende-se com a preocupação de igualdade de acesso. Isso também será contemplado na análise deste trabalho de forma a se conseguir perceber qual o impacto que poderá ter o acesso a ferramentas inteligentes no contexto técnico e, também, económico, tentando realçar que vantagens terão as empresas com acesso à IA em relação às empresas que não possam ou não optem por recorrer a essa capacidade. Segundo Rockall [Roc+25], as empresas que mais investem e adotam a IA nos seus processos demonstram um maior crescimento na produtividade e uma redução de custos relativamente aos seus concorrentes que optam ou não conseguem adotar esta abordagem, levando a um impacto económico e comercial direto, criando assim grandes assimetrias na competição entre as empresas no mercado. Estes pontos exigem uma reflexão ética, legal e organizacional cuidada, além de demonstrar a necessidade de implementação de políticas bem definidas quanto à gestão dos algoritmos.
Relativamente à estrutura desta dissertação, pretende-se que a mesma se apoie em quatro vetores principais:

- Análise crítica das bases técnicas que sustentam o par Homem-IA;
- Avaliação do impacto na qualidade, produtividade e inovação do ciclo de desenvolvimento relativamente à colaboração entre as duas partes no processo criativo;
- Análise da componente ética e legal relativamente às componentes abrangidas pela coautoria entre humanos e IA;
- Exploração de possibilidades quanto ao futuro da profissão de engenheiro de software e quanto ao nível de mudança esperada no paradigma de implementação de soluções digitais.

Com recurso a uma metodologia que combina revisão sistemática de literatura, análise comparativa de ferramentas com um caso de estudo relevante e dois casos práticos, pretende-se que este trabalho possa contribuir de forma eficiente para o

estudo teórico relacionado com a colaboração entre homem e IA na área de Engenharia de Software, sendo capaz de efetuar previsões e criar diretrizes para uma utilização ética, justa e competente da IA nos processos de desenvolvimento de software.

Após esta introdução, o Capítulo 2 – Fundamentação Teórica apresenta os conceitos essenciais que sustentam esta investigação, incluindo a evolução da Inteligência Artificial, os paradigmas tradicionais e emergentes da Engenharia de Software e o enquadramento teórico da IA como coautora. O Capítulo 3 – Estado da Arte aprofunda a análise das principais ferramentas de IA aplicadas ao desenvolvimento de software, organizadas segundo as fases do ciclo de vida e destacando tendências, capacidades e limitações atuais. O Capítulo 4 – Metodologia descreve a estratégia de investigação adotada, os critérios de qualidade, as limitações do estudo e a forma como os resultados foram estruturados. O Capítulo 5 – Avaliação Crítica da Aplicação da IA no Desenvolvimento de Software apresenta a análise detalhada do impacto da IA em cada fase do ciclo de vida do software, complementada por um caso de estudo com o GitHub Copilot e dois casos práticos comparativos envolvendo o GitHub Copilot e o OutSystems Mentor App Generator. Por fim, o Capítulo 6 – Conclusões e Trabalhos Futuros sintetiza os principais contributos da investigação, discute o estado atual de maturidade da IA como coautora e identifica oportunidades de evolução e investigação futura.

Capítulo 2

Fundamentação Teórica

A integração da IA no universo da Engenharia de Software não é um evento recente, sendo que nos anos 80 e 90 já era comum encontrar sistemas especialistas e até sistemas de testes e deteção de erros que recorriam a técnicas de *machine learning* [Cop26]. Este evento abriu uma porta de inovação e disrupção como nunca antecipado, despoletando o início dos processos de transformação na forma como sistemas digitais são implementados, testados e validados. O surgimento dos modelos generativos leva a uma grande revisão dos modelos teóricos de implementação de software, passando a contemplar com a colaboração intrínseca entre agentes humanos e agentes inteligentes artificiais.
Neste Capítulo serão analisados e estudados os pilares deste novo modelo teórico de implementação de software e de que forma esses permitiram chegar ao contexto atual, demonstrando o panorama evolucional teórico com recurso a diversas fontes já existentes de forma a garantir a robustez do conteúdo disponibilizado.

2.1 O que é a Engenharia de Software

O contexto de Engenharia de Software refere-se à área de estudo (ou disciplina) que alberga todos os detalhes e processos de implementação de software, iniciando-se no momento de especificação de requisitos até ao momento da retirada desse mesmo software do parque tecnológico, ou seja, ultrapassa a fase de entrada do software em utilização, seguindo pela manutenção e pelas adaptações ao longo da sua vida [Som16].
Em base, como mencionado por Sommerville, existem dois pontos chave na Engenharia de Software:

1. Os engenheiros de software fazem com que tudo funcione, aplicando soluções para os problemas a resolver.

2. Os engenheiros de software levam em conta todos os aspetos da produção de software, mesmo aqueles que não são técnicos.

A Engenharia de Software deve seguir práticas ajustadas à realidade dos problemas que se propõe resolver, passando pelas componentes financeiras e organizacionais das empresas ao qual determinado software vai servir.

2.2 Conceito e evolução da Inteligência Artificial

A Inteligência Artificial é o campo científico e tecnológico dedicado ao desenvolvimento de sistemas capazes de realizar processos de raciocínio, aprendizagem, perceção e tomada de decisão de forma autónoma, através de modelos computacionais que identificam padrões, inferem relações e adaptam o seu comportamento com base em dados. A história da IA surge com os primeiros algoritmos de decisão, de implementação simples, e chega, aos dias de hoje, até modelos de enorme complexidade e de aprendizagem profunda, tal como os Modelos de Linguagem de Larga Escala (LLMs), referindo como exemplo os conhecidos Chat GPT, DeepSeek, CoPilot, entre outros do mesmo tipo. Atualmente, torna-se impressionante a forma como estes modelos conseguem compreender a linguagem natural, sendo treinados com uma volumetria de dados que em outros tempos seria inconcebível, conseguindo gerar conteúdo com um grau de assertividade bastante interessante, permitindo que evoluam para cenários de tomada de decisão assistida.

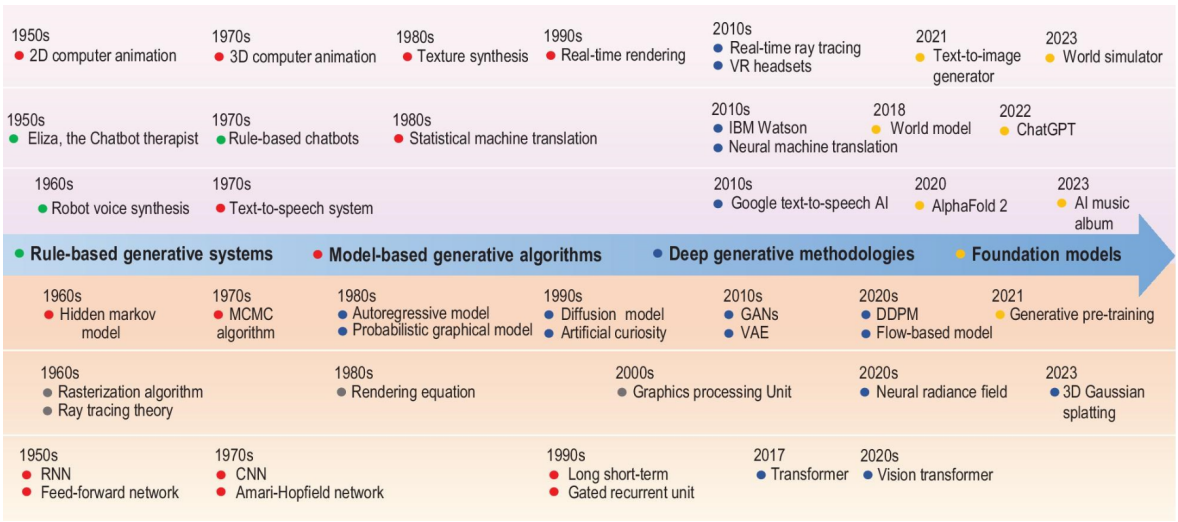

Figura 2.1: Fases de desenvolvimento e evolução da Inteligência Artificial [He+25a]

Na Figura 2.1 podemos encontrar os eventos mais relevantes das fases de evolução

da IA distribuídos numa linha temporal de forma a permitir a compreensão de que acontecimentos levaram ao cenário em que nos encontramos atualmente.
Podem-se identificar quatro vertentes evolutivas mais relevantes da IA e presentes com contexto temporal na Figura 2.1 :

- Sistemas generativos baseados em regras: implementação com algoritmos baseados em regras explícitas.
- Sistemas generativos baseados em modelos: implementação com foco na aprendizagem supervisionada e não supervisionada.
- Metodologias generativas profundas: implementação com recurso a redes neurais profundas com o intuito de geração de conteúdo e tomada de decisão assistida.
- Modelos fundacionais: implementação de modelos massivos, multi-modais e já pré-treinados para execução de tarefas de diferentes contextos.

As vertentes de metodologias generativas profundas e dos modelos fundacionais são, até ao momento, as mais relevantes e as que mais impacto estão a ter na Engenharia de Software.

2.3 Paradigmas tradicionais e emergentes na Engenharia de Software

A disciplina de Engenharia de Software tem como objetivo a aplicação sistemática de princípios matemáticos e científicos no contexto de desenvolvimento, manutenção e operação de software. Seguindo o contexto histórico, podem-se encontrar nos modelos em cascata (*Waterfall*) as primeiras abordagens aos paradigmas de gestão de processos de desenvolvimento de software, passando depois para modelos mais ágeis, que passaram a ser mais recorrentes na Engenharia de Software. Em paralelo com estes modelos surgem abordagens operacionais, como o DevOps e, posteriormente, o MLOps, que servem como práticas organizacionais e técnicas destinadas a melhorar a automação e a integração contínua, incluindo já ferramentas baseadas em *machine learning* no ciclo de vida de implementação de software [Lwa+19].
A introdução da IA nos processos de desenvolvimento de software está a provocar uma alteração drástica no perfil do engenheiro de software, permitindo que esse consiga delegar tarefas em sistemas inteligentes artificiais, levando a que este se

foque em vertentes mais orientadas à orquestração do projeto em si e de gestão dos próprios sistemas de assistência, tal como supervisiona-los.

2.4 Definição e implicações da IA como coautora

Quando se coloca o termo coautor no panorama foge-se ao contexto normalmente previsto para a IA de ferramenta auxiliar. Ser coautor é mais do que apoiar: é criar, inventar, reinventar e, acima de tudo, aprender. Aqui, torna-se imperativo incluir o conceito de agentes de IA. Estes são entidades autónomas ou semi-autónomas que têm a capacidade de compreender o ambiente, tomar decisões e agir de uma forma bastante orientada aos seus objetivos [Rus+21]. É exatamente a capacidade de agente que permite à IA ocupar um papel bastante ativo nas tarefas habituais dos ciclos de vida da Engenharia de Software, desde interpretação de requisitos, passando pela programação, pelos testes e pela manutenção de soluções. Podemos encontrar no agente de IA uma entidade que é parte ativa em tarefas habituais do ciclo de vida de software, tais como:

- Interpretação dos requisitos aplicacionais em linguagem natural;
- Geração de código nativo;
- Refatorização e otimização de soluções;
- Escrita de scripts de teste e respetiva validação;
- Capacidade de deteção de bugs e falhas;
- Criação e interpretação de documentação técnica.

Com esta abordagem de parceria entre humanos e agentes de IA, o objetivo é conseguir encontrar o equilíbrio entre humano e máquina, deixando o engenheiro de software focar-se mais na estratégia e delegando as questões mais operacionais para os sistemas inteligentes artificiais, onde estes conseguem, já nos dias de hoje, atingir índices elevados de eficiência.

2.5 Modelos de linguagem e ferramentas assistidas

A programação assistida tem como motor os já referidos LLMs. Podemos encontrar em ferramentas como GitHub CoPilot, Amazon CodeWhisperer e OutSystems Mentor App Generator grandes exemplos disso: estas ferramentas recorrem a agentes

de IA para gerar código nativo, corrigir bugs, gerar scripts de testes e, em certos casos, gerar até aplicações completas apenas baseando-se nas descrições funcionais submetidas.
Um dos marcos mais importantes que abriu o caminho para todas as ferramentas mencionadas foi o OpenAI Codex, sendo um modelo que deriva do GPT-3 e que foi treinado com enormes quantidades de código fonte de acesso público. A literatura define o Codex como o primeiro sistema capaz de converter as instruções em linguagem natural diretamente em código funcional. Uma característica interessante do Codex é que suporta múltiplas linguagens de programação e podia ser integrado em IDEs como o Visual Studio Code [Ope21]. Este modelo surge como a base do GitHub Copilot (que é a ferramenta estudada no caso de estudo e num dos casos práticos desta dissertação). Posteriormente este modelo evoluiu para versões como o GPT-5.3 Codex, permitindo a introdução de agentes mais poderosos que conseguem ser autónomos na escrita, testes e manutenção de projetos de software [Ner26].
Apesar da integração deste tipo de ferramentas no ciclo de vida do software trazer grandes ganhos ao nível da produtividade e da qualidade, sendo que a IA automatiza tarefas repetitivas, reduz o erro humano e acelera o processo de desenvolvimento [Ame+19], ainda existem preocupações quanto à confiabilidade, explicabilidade e enviesamento dos modelos que estão na base das ferramentas.

2.6 Novas competências com origem na utilização da IA na Engenharia de Software

A partir do momento que existe um novo ator que consegue efetuar um conjunto de tarefas ou apoiar em trabalhos realizados por humanos, faz com que os engenheiros de software passem a ter que obter novas competências tendo em conta a mudança de contexto, obrigando a todos que se queiram manter atualizados a se requalificarem. Podem-se identificar algumas dessas competências necessárias de forma muito simples:

- Supervisão de decisões automáticas;
- Gestão ética e legal de produto gerado pela IA;
- Validação dos resultados e rastreabilidade;
- Interpretação de modelos de IA (conjunto de técnicas que permitem compreender como e porquê um modelo de IA toma as suas decisões);

- *Prompt engineering* - o prompt de sistema define o tom e a orientação do modelo para resolver um dado pedido e o prompt de utilizador define um pedido em específico para resolver. Basicamente estamos perante a "afinação" do funcionamento da ferramenta.

Com a mudança no paradigma de programação para um cenário mais orientado aos dados (conhecimento dos modelos de IA) do que à programação em si demonstra que é muito importante existirem capacidades nos engenheiros de software para a compreensão e manipulação de datasets e garantir a transparência dos algoritmos utilizados.

2.7 Ética, lei e desafios organizacionais

Apesar de todo o potencial que advém da colaboração entre humanos e máquinas no processo de desenvolvimento de software, há certos limites e barreiras que devem ser impostas e constantemente validadas de forma a garantir a maior responsabilidade social e humana possível. Quando se fala de coautoria em Engenharia de Software levantam-se questões como:

- Quem é realmente o autor do código gerado pela IA e de que forma as próprias ferramentas conseguem descrever ou limitar a sua participação no processo de criação?
- De que forma se consegue gerir a propriedade intelectual?
- Quem é considerado responsável no caso de falhas ou impactos negativos?
- De que forma se consegue dar acesso de forma igual às ferramentas inteligentes?

Todas estas questões levantam a necessidade de criação de novas plataformas legais e éticas que possam agilizar e assegurar uma forma mais correta, segura e justa de utilização da IA na Engenharia de Software.
O primeiro esforço realizado no sentido de obter um enquadramento regulatório para a IA na União Europeia é denominado *EU AI Act* e compreendeu um conjunto de regras e obrigações baseadas num paradigma de categorização por risco (p.e.: mínimo, limitado, elevado e inaceitável) de forma a tornar mais percetível o potencial impacto das incidências nessas mesmas categorias. O *EU AI Act* pretende impor requisitos bastante rigorosos de forma a manter a transparência da utilização da IA, garantir a supervisão humana, manter a qualidade dos dados, capacitar a gestão de riscos e manter a rastreabilidade das criações da IA [Par23].

É importante manter uma cultura dentro da comunidade de engenheiros de software que promova o diálogo aberto sobre as considerações éticas de forma a tornar-se um conceito central em qualquer fase do ciclo de vida de um software. É necessário promover e incentivar discussões sobre qualquer tipo de dilema ético e garantir que, de forma colaborativa, se conseguem encontrar soluções eficazes para os problemas neste âmbito.

Capítulo 3

Estado da Arte

O contexto do desenvolvimento de software tem sofrido uma enorme evolução potenciada por todos os fatores que a rodeiam. O meio em que o software existe é de alto desempenho, mudança rápida e onde, de dia para dia, torna-se necessário encontrar soluções de escalabilidade altamente robustas e dinâmicas. Ao longo dos anos refinaram-se processos, paradigmas, e até linguagens de programação. Até ao dia em que algo inteligente e de uma forma não biológica assume o comando com um elevado grau de autonomia que permite acelerar o desenvolvimento do software e escalar soluções de uma forma como nunca se viu. Chegou a era da Inteligência Artificial. Deixamos de ter o programador (que nos últimos anos já passou de um mero codificador para algo mais evoluído, com mais predominância, *soft skills* e mais participativo na definição de requisitos), e passamos a ter um conjunto de linhas de código que associados a um conjunto de dados transformados em conhecimento, assumem as rédeas do universo do software, desde a escrita de código até à manutenção preditiva e corretiva de sistemas [Bri25] .

Os últimos avanços na aplicação de IA e respetivos agentes à Engenharia de Software têm levado a uma ampliação do ecossistema de ferramentas que intervêm nas diferentes fases do ciclo de vida do desenvolvimento de software. Estas novas ferramentas que aumentam o leque de alternativas e soluções disponíveis não se prendem com conceitos de apenas geração de código. Elas conseguem abranger a componente de análise de requisitos, desenho de arquitetura de software, testes, geração de documentação, manutenção, implementação de segurança, otimização e até operações. A melhor forma de conseguir documentar o estado atual é categorizando as ferramentas por tipo de propósito e maturidade.

3.1 Ferramentas orientadas à análise de requisitos

As ferramentas orientadas para a análise de requisitos baseiam-se em modelos de linguagem de forma a interpretar as descrições entregues em linguagem natural e com capacidade para as converter em artefactos mais formais. Podem-se verificar os seguintes exemplos:

- Claude/Gemini/ChatGPT - capacidade de interpretar requisitos em linguagem natural, criar histórias de utilizador e detetar ambiguidades [Sin23].
- Microsoft CoPilot for Azure/GitHub Issues AI - capacidade de gerar automaticamente requisitos técnicos e tarefas [Git26c].
- OutSystems Mentor App Generator - capacidade de conversão de requisitos fornecidos em linguagem natural em modelos de dados e aplicações funcionais [Out25c].

Estas ferramentas têm a capacidade de reduzir o tempo de levantamento de requisitos, além de aumentar a consistência e ajudarem na estruturação dos respetivos requisitos de forma mais eficiente.

3.2 Ferramentas orientadas à geração de código

As ferramentas orientadas à geração de código são as que têm mais visibilidade no mercado, apresentando níveis cada vez mais elevados de maturidade. Os agentes que suportam estas ferramentas são baseados em LLMs especializados para programação, tais como:

- Anthropic Claude - modelo com uma grande capacidade de raciocínio e orientado à programação, sendo considerado o LLM na atualidade, na sua versão Sonnet 4.5, como melhor escolha para o desenvolvimento de software [Ant25].
- GitHub Copilot - geração de código com base em contexto, capaz de efetuar sugestões de implementação inteligentes, refatorizações, scripts de testes e suporta apoio a tarefas de debugging [Git26b].
- Amazon CodeWhisperer (Amazon Q Developer) - modelo bastante focado na geração de código seguro segundo os critérios de conformidade [Ama26].
- Tabnine - capacidade de efetuar o autocomplete de código com base em modelos já treinados em código permissivo [Tab26].

Estas ferramentas têm a capacidade de aumentar drasticamente a produtividade e de reduzir o fator de erro, fazendo com que o engenheiro de software se possa focar em temas mais orientados à arquitetura e à gestão do projeto em si.

3.3 Ferramentas de arquitetura de software e design

As ferramentas de geração de código como as exemplificadas no ponto anterior não se limitam à programação de software, enquadrando-se de forma mais ou menos subtil nos conceitos de arquitetura de software e design. Essas ferramentas conseguem sugerir padrões de arquitetura através do reconhecimento de estruturas existentes e até identificar dependências entre componentes.
Além disso, existe um conjunto de ferramentas que têm como objetivo a geração de diagramas de bases de dados, de modelos de arquitetura e geração de documentação técnica. Como exemplos, podem-se verificar as seguintes ferramentas:

- Mermaid/PlantUML - geradores de diagramas UML com recurso a LLMs [Mer26].
- Figma AI - ferramenta orientada à criação de interfaces e fluxos de navegação [Fig26].
- OutSystems Mentor App Generator - ferramenta com capacidade de geração de modelos de dados, fluxos lógicos e todos os artefactos Low Code necessários para a geração de uma aplicação [Out25c].

O impacto destas ferramentas é notório na capacidade para diminuir a janela temporal entre a entrega dos requisitos e o design, reduzindo a distância entre a prototipagem e a definição das histórias de utilizador finais.

3.4 Ferramentas de testes

Hoje em dia existem ferramentas que se apoiam em agentes e LLMs capazes de gerar scripts de testes eficientes e detetar falhas durante a análise do código e durante a execução dos respetivos testes. Podem-se encontrar as seguintes ferramentas com relevância neste campo:

- Diffblue Cover - ferramenta de geração de testes unitários para a linguagem JAVA [Dif25].
- Testim.io - ferramenta para efetuar testes *end-to-end* com recurso a agentes inteligentes [Tes25].

- Copilot for Pull Requests - ferramenta com capacidade de criação de testes e validação automática de alterações ao código entre versões [Nex23].

Estas ferramentas permitem aumentar a cobertura de testes, levar a uma redução de custos nos processos de *quality assurance* e, ao mesmo tempo, reduzir o tempo necessário para efetuar entregas de software funcional.

3.5 Ferramentas de geração de documentação

As ferramentas de geração de documentação surgem no contexto de automatização de processos tediosos para o humano, libertando o mesmo para outras tarefas mais relevantes e de maior importância. Dentro destas ferramentas de geração de documentação automática, salientam-se as seguintes:

- Mintlify AI - ferramenta focada na geração automática de documentação técnica [Min26].
- Swimm - ferramenta com capacidade para gerar documentação contextualizada e sincronizada com o código [Swi26].
- Sourcegraph Cody - ferramenta com capacidade para explicar o código analisado e efetuar navegação inteligente pelo mesmo [Sou26].

Estas ferramentas têm a capacidade de reduzir a dívida técnica e de melhorar a compreensão e interpretação de sistemas mais complexos.

3.6 Ferramentas de segurança

No panorama atual da Engenharia de Software, a segurança é fator crucial para o sucesso, tendo em conta que os atacantes se tornam cada vez mais agressivos e insistentes. Nessa área a IA torna-se também um fator diferenciador, já que permite uma deteção de vulnerabilidades muito mais eficiente e tem a capacidade de efetuar sugestões de correção. Pode-se enumerar um conjunto de exemplos de ferramentas com a componente de IA, tais como:

- GitHub Advanced Security + CoPilot - ferramenta com a capacidade de efetuar varrimentos inteligentes de dependências e secrets do código implementado [Mic26b].
- Checkmarx AI - ferramenta para análise estática que utiliza modelos treinados com padrões de vulnerabilidades [Che26].

- Snyk AI - ferramenta que utiliza a IA para detetar vulnerabilidades e sugerir correções [Sny26].

Estas ferramentas permitem reduzir riscos e custos de resolução, sendo que, com a sua utilização desde as fases iniciais nos projetos de software leva à abordagem de *security by design*.

3.7 Ferramentas de otimização operacional

A Engenharia de Software não se resume a programação. A componente operacional é extremamente importante para que um dado projeto de software decorra de forma compreensiva e sem grandes desvios em relação ao tempo estimado de execução como sem grandes desvios em relação ao orçamento previsto. Para tal, existe um conjunto de ferramentas e funções que, com recurso a agentes e modelos de IA, conseguem otimizar e cooperar de forma a otimizar fluxos e a monitorizar processos e operações. Podem-se dar como exemplo de ferramentas inseridas neste contexto as seguintes:

- AI GitHub Action - ferramenta com a capacidade de criação de workflows de forma automática e inteligente [Git26a].
- Azure DevOps MCP Server - ferramenta com a capacidade de gerar pipelines, análise inteligente de falhas e otimização operacional [Mic26a].
- Datadog AI (Internal Developer Portal) / New Relic AI - ferramentas com a capacidade de deteção de anomalias e previsão de falhas durante a implementação de software [Dat26].

Com recurso a estas ferramentas potenciadas pelo uso de agentes e IA, os engenheiros de software conseguem aumentar a fiabilidade operacional e reduzir drasticamente o tempo necessário para resolução de incidentes que decorrem durante a implementação de um dado projeto.

3.8 Ferramentas de agentes autónomos para desenvolvimento de software

Este tipo de ferramentas é o mais emergente no universo da Engenharia de Software e o que está ligado diretamente ao conceito de coautoria da IA com humanos. Estas ferramentas conseguem diferenciar-se das ferramentas tradicionais assistidas por não dependerem de instruções passo a passo, sendo capazes de decidir

e agir de forma autónoma dentro de um conjunto de limites definidos. Podem-se referir as seguintes ferramentas como exemplos:

- OpenAI Agents - estes agentes podem ser especializados na análise de código, debugging e testes, sendo capacitados de um raciocínio avançado e de uma autonomia parcial. Estes agentes são implementados com o AgentKit da OpenAI [Ope26].

- Anthropic Claude - apesar de não ser um agente autónomo por definição, é muitas vezes utilizado como core cognitivo de imensos agentes, tendo em conta todo o seu potencial nas áreas de raciocínio, decomposição de problemas e geração de código. Podemos encontrar o Claude em frameworks como LangChain e AutoGen [Ant25].

- Visual Studio Code com Copilot Agents - A Microsoft tornou o Visual Studio Code num ambiente central para agentes de IA, sendo que passa a ser possível a sua integração com o GitHub Copilot Chat, Copilot Agents e extensões inteligentes de geração de código [Stu26].

- Cursor IDE - esta ferramenta permite incorporar agentes que executam tarefas complexas de forma autónoma e continua. Os agentes incorporados pelo Cursor IDE conseguem explorar o código fonte, executar comandos e corrigir *bugs* apenas baseados em requisitos definidos de forma *high level* pelo engenheiro de software responsável. Podem ser incorporados agentes baseados em diversos modelos, desde modelos da Anthropic, Google, OpenAI, Grok e Kimi. A vertente de coautoria entre a IA e os humanos fica extremamente vincada com ferramentas como o Cursor IDE tendo em conta que o sistema executa iterações automáticas de implementação, testes e correções tendo apenas como base os objetivos definidos pelos respetivos engenheiros de software [Doc24].

Este tipo de ferramentas são as que mais se identificam com o processo de coautoria entre humanos e IA. Estes sistemas permitem que a IA deixe de funcionar apenas como um mero assistente e passe a ser um parceiro muito mais ativo naquilo que é o processo de desenvolvimento de software.
Resumindo as ferramentas mais proeminentes atulamente no mercado para cada uma das fases do SDLC são as da Tabela 3.1.

Tabela 3.1: Ferramentas de IA no SDLC e respetivas vantagens

Fase do SDLC	**Ferramentas de IA**	**Vantagens Principais**
Planeamento e Análise	ChatGPT, Claude, Gemini, Llama 3	Geração de requisitos, análise de riscos, criação de user stories, validação conceptual, aceleração da pesquisa e comparação de fontes.
Design Arquitetural	GitHub Copilot, Cursor, Claude Sonnet, Gemini Advanced	Sugestões de padrões arquiteturais, geração de diagramas, validação de modelos, identificação de dependências e deteção precoce de inconsistências.
Implementação (Coding)	GitHub Copilot, Codeium, Cursor, Amazon CodeWhisperer	Aceleração da escrita de código, redução de erros, geração de testes unitários, completions contextuais, aumento da produtividade entre 20–40%.
Testes e Qualidade	Copilot for Test Generation, Codeium Test Assist, Meta TestGen	Geração automática de casos de teste, maior cobertura, deteção de vulnerabilidades, redução de tempo de QA, testes baseados em cenários reais.
Segurança e Compliance	Microsoft Security Copilot, Sentinel AI, OWASP AI Tools	Identificação de vulnerabilidades, análise de logs, deteção de anomalias, apoio em conformidade (GDPR, ISO 27001), mitigação de riscos.
Deployment e DevOps	GitHub Actions AI Assist, Azure DevOps Copilot, Google Cloud Duet AI	Automação de pipelines, otimização de CI/CD, previsão de falhas, melhoria da observabilidade, redução de tempo de entrega.
Manutenção e Operações	Copilot Chat, Claude Projects, Gemini Ops	Diagnóstico automático, resolução assistida de incidentes, documentação contínua, análise de regressões, suporte ao troubleshooting.

3.9 Contexto geral

Pensar que a Inteligência Artificial é apenas uma ferramenta quanto à escrita do código em si é altamente limitador. Não podemos abordar este tema com a ideia pré-concebida de que a Inteligência Artificial apenas se move no contexto daquelas tarefas repetitivas e mundanas do mundo da programação. Os conceitos de desenvolvimento de software e Engenharia de Software ficam completamente reformatados em relação ao que se conhecia. Passamos a encontrar a Inteligência Artificial integrada na segurança de sistemas, a prever e detetar padrões e a responder aos mesmos de acordo com a aprendizagem que teve, a definir arquiteturas de sistemas e aplicações tendo como base todos os conceitos e caminhos que nós, humanos, lhes ensinamos. Mesmo na componente de programação, pode-se dizer que a Inteligência Artificial altera a definição do que significa o conceito de programa de software [Rib23]. A ideia demonstrada em todas estas palavras escritas até agora neste estado da arte tendem, de uma forma muito leve, demonstrar que o nível de penetração da Inteligência Artificial neste mundo já é de tal modo elevado que já existem artefactos denominados "modelos de linguagem" que já permitem a sistemas de desenvolvimento de software compreender os requisitos escritos em linguagem natural, fazendo sugestões de implementação e, até, escrever o código necessário para o nascimento das respetivas soluções informáticas. Ferramentas amplamente conhecidas de todos os que de alguma forma se envolvem no desenvolvimento de software, como o GitHub, Amazon CodeWhisperer, Claude e Tabnine são um grande exemplo desta nova era de desenvolvimento assistido por Inteligência Artificial [Bri25].

Perante todo este contexto, o pretendido neste estado de arte é, além de demonstrar o grau evolutivo da penetração da Inteligência Artificial no desenvolvimento de software, explorar as principais metodologias, tecnologias e as tendências que estão a levar este movimento em direção a um futuro já bastante presente. Faz todo sentido, nesta fase, analisar-se de que forma é que a IA já está integrada em todas as diferentes fases do ciclo de vida do software em si, onde estão os grandes desafios técnicos que surgem a cada passo que se dá na evolução desta parceria e, claro está, todos os prós e contras que isto tem na evolução e no futuro da Engenharia de Software como disciplina [Sil21].

Um primeiro evento em análise relativamente ao impacto provocado pela Inteligência Artificial no desenvolvimento de software é a interferência na automação do ciclo de vida do desenvolvimento em si. Neste momento, todas as fases do respetivo ciclo de vida estão a ser transformadas pela Inteligência Artificial [Bri25]:

- Na atualidade, ferramentas de gestão de projeto e produto (como o JIRA,

Azure DevOps) já utilizam recursos de *Machine Learning* para apoiar nas fases de planeamento e levantamento de requisitos, efetuando sugestões ao nível de prioridade de tarefas e com sistemas de previsões de atrasos.

- As ferramentas mais modernas de programação, como o GitHub Copilot, Eclipse, Amazon Whisperer e Visual Studio Code já possuem a capacidade de programação assistida com recurso a agentes mencionados nas ferramentas (entre outros não mencionados), onde a Inteligência Artificial que serve de base a essa funcionalidade gera código por si só, além de sugerir correções ao código criado manualmente e consegue, inclusive, escrever os scripts de teste para o respetivo código [Rib23].
- A Inteligência Artificial já consegue, com base no conhecimento obtido nas fases anteriores do ciclo de vida do desenvolvimento de software, escrever testes eficientes para serem executados automaticamente, levando a uma melhoria significativa, tanto na superfície do tecido de código coberto como na redução de falhas. As ferramentas mencionadas anteriormente como ferramentas de testes (Diffblue: gera testes de forma automática, Testim.io e Copilot for Pull Requests: geram testes *end to end* recorrendo a modelos de aprendizagem automática) comprovam exatamente este ponto, permitindo ao engenheiro de software obter software com maior qualidade e fiabilidade.
- Através da interpretação de logs e traces gerados pela utilização e execução das aplicações, a IA consegue, com recurso aos dados e informação disponibilizados para que a mesma adquira conhecimento, identificar anomalias e defeitos que, por norma, passam despercebidos aos métodos tradicionais de verificação, como bugs não determinísticos, erros que apenas ocorrem em condições muito específicas ou defeitos resultantes de interações complexas entre componentes. Estes bugs são difíceis de detetar porque não ocorrem de forma consistente, dependem de combinações raras de estados ou só surgem em sistemas altamente distribuídos, onde a causalidade é difícil de rastrear.
- Com recurso aos dados históricos e às métricas de performance, a Inteligência Artificial consegue efetuar predições de possíveis problemas futuros a vários níveis, sendo as componentes de performance, disponibilidade e segurança grandes exemplos do valor acrescido associado à utilização deste tipo de abordagem.

Pode-se realçar na lista de impactos da Inteligência Artificial no desenvolvimento de software, a já mencionada geração de código e o recurso a modelos para suportar a geração de conteúdo de forma autónoma:

- Atualmente já se utilizam modelos de linguagem como o GPT, DeepSeek, Claude e o modelo base do Copilot para gerar código nas mais diversas linguagens de programação, tendo como base pedidos customizados pelo utilizador ou em documentação técnica submetida e respetivamente interpretada pelos LLMs (*Large Language Models*). Os modelos de linguagem mencionados não operam de forma isolada, mas sim integrados com ferramentas de programação como, por exemplo, o GitHub Copilot, o Amazon CodeWhisperer e o OutSystems Mentor App Generator. Mais recentemente surgiram também os agentes como, por exemplo, o Copilot Agents no Visual Studio Code e agentes baseados no Claude ou GPT, que conseguem efetuar diversas tarefas sequencialmente e de forma bastante autónoma, aumentando a produtividade e permitindo ao engenheiro de software focar-se noutras tarefas mais orientadas à supervisão e à arquitetura de software.

- Cada vez mais a Inteligência Artificial está a tomar conta do desenvolvimento de software com recurso a ferramentas Low Code/No Code. Existe o exemplo do produto "Mentor App Generator" disponibilizado pela OutSystems na versão ODC (versão da plataforma OutSystems totalmente nativa em cloud, baseada em *containers* e *kubernetes*, sendo a sigla uma abreviação de *OutSystems Developer Cloud*) da sua ferramenta de desenvolvimento Low Code, em que, com base na descrição funcional e não funcional submetida no prompt em linguagem natural, esse mesmo produto, em poucos minutos gera a aplicação na sua totalidade (de salientar que não é perfeito, muitas vezes e, com projetos de complexidade média ou alta, é necessário o programador efetuar as customizações mais complexas e os ajustes finais).

Outro impacto que se sente de uma forma bastante massiva é na componente de segurança aplicacional (e rede) e na qualidade de código entregue nas soluções finais:

- Com o conhecimento já obtido ao longo dos últimos tempos, os componentes de IA têm facilidade em detetar padrões defeituosos ou incorretos nas soluções de software e, em muitos casos corrigir os problemas de forma automática ou, no mínimo informar as equipas responsáveis por esse componente de que foi encontrado um defeito.

- Além de ferramentas de código estático, a Inteligência Artificial já penetrou nas ferramentas de análise em tempo real, permitindo uma verificação em tempo real muito mais eficiente em falhas ou anomalias, como o exemplo das ferramentas DeepCode e Snyk [Sil21].

Um impacto bastante notório da interferência da Inteligência Artificial no desenvolvimento de software prende-se com a colaboração entre o ser humano e a máquina, sendo que o programador e/ou operador passa a navegar o desenvolvimento e operação das aplicações de forma assistida:

- Em muitos cenários, não há uma substituição real do engenheiro de software na programação de soluções tecnológicas, mas existe um suporte bastante efetivo por parte da Inteligência Artificial para essas tarefas [Bri25].
- Noutro cenários, o programador é libertado para tarefas relacionadas com arquitetura ou outras mais criativas, deixando as tarefas repetitivas, de análise ou analítica para a Inteligência Artificial.

Toda a capacidade da Inteligência Artificial e o seu nível de intrusão atual e futuro no desenvolvimento de software leva a que existam preocupações aos níveis técnico, ético e legal:

- Um caso muito analisado é o do enviesamento dos dados submetidos para treino de modelos. Este enviesamento pode levar a resultados completamente fora do enquadramento esperado, levando, muitas vezes, a problemas de critérios sociais (por exemplo, decisões que podem ser conotadas como racistas). O código gerado com base nesses modelos tem grande probabilidade de ser inseguro ou ineficiente [Sil21].
- Muitas das vezes não se consegue perceber como é que a Inteligência Artificial chega a determinado resultado ou solução, ou seja, há falta de explicabilidade.
- A existência de Inteligência Artificial, se for mal gerida, pode provocar a perda de competências técnicas por parte dos humanos [Rib23].

Facilmente se consegue perceber que o conceito de desenvolvimento de software como era conhecido está completamente transformado pela penetração da IA. Além da capacidade de apoiar e reforçar a capacidade humana para o desenvolvimento e implementação de soluções tecnológicas, a Inteligência Artificial já tem uma parte muito mais ativa, sendo que é autónoma em muitas tarefas de programação e operação de aplicações e processos do desenvolvimento de software. A fusão entre as duas inteligências (humana e artificial) traz à superfície um novo paradigma, que quebra regras e princípios utilizados e acreditados até então. Mas esta quebra de barreiras levanta outras preocupações, sendo algumas já antigas e que não foram totalmente resolvidas ainda no tempo em que a Inteligência Artificial não pertencia

a este contexto. Levantam-se imensas questões nos âmbitos técnico, éticos e legais, sendo que se tem como objetivo a construção de soluções confiáveis, seguras e socialmente responsáveis. Resumindo, não se pode abordar este tema como se de uma inovação tecnológica se tratasse mas sim com uma redefinição completa de um conceito que transporta o desenvolvimento, a colaboração e a evolução de ecossistemas digitais cada vez mais inteligentes e autónomos.

Capítulo 4

Metodologia

A abordagem de base da investigação que sustenta este trabalho é a revisão sistemática de literatura com acréscimo de valor com a apresentação de um caso de estudo prévio e a apresentação de dois casos práticos. Pretende-se, através deste caminho, conseguir obter uma análise crítica e consolidada sobre o atual estado relativamente à integração da IA na autoria dos processos de Engenharia de Software nas suas demais fases e vertentes e, utilizar os casos de estudo e práticos para o demonstrar efetivamente. A escolha desta abordagem mais no sentido exploratório e qualitativo prende-se com a enorme necessidade de conseguir compreender o fenómeno a estudar no mesmo passo que este evolui e partindo do princípio que ainda não existem dados quantitativos emergentes nem modelos consolidados.

4.1 Estratégia da investigação

A estratégia usada para colocar em prática a investigação baseia-se em quatro grande fases sequenciais:

1. Identificação e obtenção de fontes bibliográficas

    - Os critérios escolhidos para a seleção de documentação e bibliografia centram-se em artigos científicos, teses, livros, artigos particulares de instituições focadas nas áreas de Tecnologia de Informação e Inteligência Artificial e relatórios técnicos. Toda a documentação tem, maioritariamente, datas de publicação entre 2020 e 2026, salvo exceções que fazem sentido no contexto em que se inserem.
    - As palavras-chaves de pesquisa selecionadas para obtenção dos melhores resultados tiveram como base os principais conceitos que servem de estrutura para o tema da IA como coautora no desenvolvimento de soft-

ware. Como tal, as palavras-chaves escolhidas inserem-se nas seguintes expressões: “Inteligência Artificial como coautora”, “IA na Engenharia de Software”, “modelos de linguagem de grande escala”, “ferramentas de programação assistida” (estas demonstram diretamente o conceito em análise desta dissertação), “automação no desenvolvimento de software”, “ética na IA”, “propriedade intelectual em IA” (estas pretendem capturar documentação e literatura que traga contexto sobre os impacto, riscos e desafios éticos e legais) e outras escolhidas pontualmente num contexto mais dinâmico de forma a refinar a pesquisa.

- Um dos pontos em consideração foi garantir a obtenção de documentos relevantes de fontes credíveis como IEEE Xplore, ACM Digital Library, SpringerLink, Scopus e Google Scholar e que todos os documentos obtidos noutras fontes se encontravam em concordância com a documentação obtida nas fontes primárias.

2. Análise do conteúdo da bibliografia obtida e capacidade de síntese

    - Posteriormente à fase de recolha de documentos, os mesmos foram analisados tendo em conta o papel ocupado pela IA nas diferentes fases do ciclo de implementação de software, o impacto em características como a arquitetura, a qualidade de conteúdo produzido e produtividade obtida, as competências emergentes para os envolvidos nos processos e quais as questões éticas e legais associadas ao tema.
    - A síntese segue uma abordagem puramente relacionada com o tema de forma a tentar facilitar a descoberta de padrões, *trends* e conceitos onde há maior falta de conhecimento.

3. Apresentação de caso de estudo prévio

    - O trabalho é valorizado com a apresentação de um caso de estudo (conforme mencionado anteriormente) de uma ferramenta amplamente utilizada no mercado: O GitHub com a sua componente de IA, o Copilot. A escolha para o caso de estudo foi tomada tendo em consideração que, neste momento, o Copilot integra de forma nativa a maior plataforma que serve de repositório de código (o GitHub) e onde se encontra mais investigação empírica publicada, como o caso do artigo de Peng [Pen+23].
    - A análise do caso de estudo pretende fornecer uma base comparativa sólida, já que o GitHub Copilot é uma referência no que diz respeito ao grau de maturidade do mercado.

A apresentação de um caso de estudo seguido de dois casos práticos tem o intuito de disponibilizar uma base sólida de conhecimento através do respetivo caso teórico, servindo depois para comparar com os resultados obtidos nos casos práticos, permitindo assim tornar as conclusões mais robustas e validar as diferentes variáveis em análise nos vários cenários.

4. Validação por análise de ferramentas e apresentação de caso prático na mesma tecnologia do caso de estudo prévio
    - A investigação do caso de estudo prévio é reforçada pela implementação de um caso prático com recurso à mesma tecnologia desse mesmo estudo anterior: o Github Copilot.
    - O objeto de implementação será aproximadamente semelhante ao objeto de implementação do caso prático seguinte de forma a fornecer um conjunto de dados analíticos e métricas mais consistentes e compreensíveis.
5. Validação por análise de ferramentas e apresentação de caso prático numa tecnologia Low Code para uma solução semelhante à do caso prático anterior
    - A investigação é reforçada por um caso prático desenvolvido propositadamente para esta dissertação, recorrendo a um produto orientado ao desenvolvimento RAD (*Rapid Application Development*, sendo uma abordagem de aceleração dos processos de desenvolvimento de software), produzido pela OutSystems [For23], que aposta na utilização da IA para acelerar os processos de implementação de software: o OutSystems Mentor App Generator. A relação com o caso de estudo e o caso prático mencionados anteriormente é de complementação, ou seja, o caso do GitHub Copilot fornece o enquadramento teórico e empírico enquanto o caso prático do OutSystems Mentor App Generator permite validar e aprofundar as conclusões num cenário controlado.
    - A revisão da documentação teórica pretende-se que seja suportada por análise de ferramentas como o GitHub CoPilot e o OutSystems Mentor App Generator, sendo este último associado ao caso prático descriminado neste ponto.

4.2 Critérios de qualidade

O trabalho demonstrado neste documento impõe o maior rigor e veracidade possível, transmitindo assim a maior confiança e objetividade possível nos dados e infor-

mação transmitida. Para cumprir com este objetivo de forma plena, são colocadas em prática as seguintes abordagens:

- Importância e relevância do tema: a bibliografia selecionada aborda de forma próxima a IA com a Engenharia de Software.
- Informação atualizada: a bibliografia é priorizada de acordo com a sua idade, dando prioridade a documentos mais recentes de forma a refletir com maior precisão o estado da arte.
- Diversificação bibliográfica: a bibliografia deve seguir fontes de forma abrangente para evitar o mais possível qualquer enviesamento de perspetiva.
- Comparação bibliográfica: a bibliografia deve ser contrastada e comparada de forma a conseguir obter os pontos em comum das diferentes perspetivas.
- Caso de estudo vs. Casos práticos: pretende-se reforçar as evidências do caso de estudo experimentado por terceiros com a execução de dois casos práticos de forma a compreender em primeira instância os prós e os contras do novo paradigma defendido neste documento (a Inteligência Artificial como coautora no processo de desenvolvimento de software).

4.3 Limitações da metodologia

A evolução do contexto da IA por si só é tão rápida que é extremamente difícil garantir que a bibliografia obtida se mantenha atualizada tanto durante a execução do trabalho como nos tempos próximos à sua conclusão. Além disso, não existe um conjunto alargado de estudos empíricos de como a IA impacta a Engenharia de Software, o que reduz a generalidade para algumas conclusões retiradas do trabalho. Este foi um dos argumentos da utilização neste trabalho de casos de estudo e práticos, permitindo reforçar o processo e conclusões obtidos.

4.4 Forma e estrutura dos resultados

O Capítulo 5 - “Avaliação Crítica da Aplicação da IA no Desenvolvimento de Software” apresenta os resultados da revisão sistemática, do caso de estudo e dos casos práticos da forma mais objetiva possível para responder aos objetivos definidos e para apoiar na integra as conclusões finais deste trabalho.

Capítulo 5

Avaliação Crítica da Aplicação da IA no Desenvolvimento de Software

A abordagem na Engenharia de Software com a IA integrada transforma por completo os paradigmas utilizados e conhecidos até há bem pouco tempo. Tendo em conta esta mudança drástica, pretende-se, neste Capítulo, expor uma análise com um elevado grau crítico e aprofundado, apoiado e sustentado tanto por literatura revista como por casos de ferramentas atuais com integração da IA. Para isso serão apresentados um caso de estudo sobre o GitHub Copilot, um caso prático com essa mesma ferramenta e um segundo caso prático, o OutSystems Mentor App Generator.

5.1 Adaptação do ciclo de vida de software para além da automação

Existia um (pre)conceito na sociedade, com mais ou menos contexto, sobre o que é a IA e sobre os vários processos associados ao conceito de software. Pensava-se precipitadamente que a IA apenas automatiza processos e que existe para efetuar aquelas tarefas repetitivas: a típica imagem dos robots numa linha de montagem de uma fábrica que fazem o mesmo procedimento vezes e vezes sem conta. No entanto, atualmente, esse conceito está extremamente afastado da realidade. A IA surge num patamar que obriga à total reconfiguração de cada uma das fases do ciclo de vida de desenvolvimento de software (em inglês, SDLC: *Software Development Life Cycle*).

Figura 5.1: Fases do Ciclo de Vida de Desenvolvimento de Software (SDLC). Baseado na imagem da fonte: [Ide24]

A Figura 5.1 resume as fases do SDLC. Copilotos e agentes estão a transformar a forma como desenvolvemos, testamos e implementamos software. A IA substitui cada vez mais os humanos num conjunto alargado de tarefas de desenvolvimento, tornando a perspetiva dos engenheiro de software bastante diferente, com mais cognição, menos repetição. Os engenheiros precisam agora de orientar, não apenas de programar. As orientações são cruciais: pense em segurança, transparência e rastreabilidade. As organizações com visão de futuro incorporam a IA em todos os departamentos, e não apenas nos projetos.
Como descrito pela Ideas 2 IT [Ide24], a IA interfere nas diferentes fases do SDLC da seguinte forma:

- *Fase 1: Obtenção de requisitos* - a IA consegue obter e construir os requisitos mesmo quando estes são transmitidos com ruído, isto é, com ambiguidades, omissões ou descrições pouco estruturadas que surgem naturalmente na comunicação humana.
- *Fase 2: Design* - desenhar a arquitetura com visão para o futuro, não em retrospetiva.
- *Fase 3: Desenvolvimento* - transformar o código num sistema de aprendizagem contínua.
- *Fase 4: Testes* - cobertura inteligente ao invés de força bruta.
- *Fase 5: Segurança* - passar a ser integrada, não adicionado posteriormente.
- *Fase 6: Entrega* - entrega contínua com inteligência contínua.
- *Fase 7: Manutenção e otimização* - capacidade integrada de auto-regeneração.

Tanto quanto sabemos, os primeiros estudos sistemáticos sobre a produtividade assistida por IA surgiram em 2023, como o caso dos estudos de Cambon e seus pares [Cam+23] representados na Tabela 5.1. É perfeitamente assumido que estes apenas representam o início da adoção em massa de LLMs e agentes para o desenvolvimento de software. Embora estes estudos antecedam, no tempo, o nível de

maturidade atualmente alcançado pelas ferramentas, os mesmos continuam a ter um peso substancial no tema por serem metodologicamente documentados e por permitirem estabelecer as primeiras métricas comparáveis.
No entanto, de forma a dar contexto mais atual, estes dados são complementados com os estudos de Coutinho [Cou+24], sendo estes mais recentes do que os estudos de Cambon e refletindo uma segunda geração de ferramentas e práticas. Desta forma a Tabela 5.1 não apresenta apenas dados históricos, mas sim uma evolução progressiva da produtividade ao longo da adoção da IA, permitindo assim uma análise mais completa e contextualizada.

Tabela 5.1: Ganhos de produtividade com apoio de IA em tarefas de suporte e gestão [Cam+23]

Tipo de Tarefa	**Ganho de Produtividade (%)**	**Impacto na Qualidade**
Documentação técnica	+37%	Qualidade mantida
Redação de emails	+25%	Mais rápido, menos erros
Análise de dados/relatórios	+40%	Maior precisão
Revisão de código	+20%	Menor tempo de supervisão

A IA desempenha, atualmente, um papel transversal a todas as fases do ciclo de vida do desenvolvimento de software, com influência na forma como cada uma dessas fases é concebida, executada e otimizada. A IA e os agentes utilizados como artefacto não se limitam a executar tarefas automatizadas de forma repetitiva, passando a estender-se à interpretação de requisitos, apoio em decisões de arquitetura, geração e validação de código, reforço e implementação de segurança, otimização de processos de entrega e promoção de mecanismos de manutenção automática. Nas subsecções seguintes são descriminados em detalhe os pormenores da influência da IA e agentes em cada uma das fases, seguindo o já descrito sobre a Figura 5.1.

5.1.1 Análise de requisitos

Nesta primeira fase do ciclo de vida do desenvolvimento de software, a IA surge através da figura dos LLMs que funcionam como um conversor de narrativas complexas em linguagem natural, em especificações técnicas devidamente estruturadas. Todo este poder permite a redução de lacunas e tempo na comunicação existente entre os *stakeholders*, que não têm as *hard skills* técnicas e as equipas de desenvolvimento, que muitas vezes não possuem as *soft skills* de comunicação

[Hem+25].
Mas essa capacidade acarreta riscos. O excesso de confiança no trabalho realizado pelos LLMs na tradução referida no parágrafo anterior pode levar a falhas de verificação que levam a falta de clareza e tradução de conceitos de forma errada, provocando erros em fases futuras na conceção de software. Esta fator demonstra que o engenheiro de software é uma peça fundamental na totalidade da máquina produtora de soluções, não devendo de forma alguma menosprezar a possibilidade de falha da IA, garantindo assim que todos os requisitos finais são exequíveis de implementação e que toda a estrutura obtida é integra e robusta [Beg+25].

5.1.2 Design e arquitetura

Quando entramos na fase de design e arquitetura da solução, já com os requisitos bem definidos, aprovados e acordados entre todas as partes, a IA adota outra *persona* no ciclo de vida do desenvolvimento de software. Aqui, a IA recorre aos milhares de padrões que aprendeu sobre arquitetura em imensos projetos para efetuar sugestões e composições arquitetónicas que respondam às nossas necessidades. No entanto, a experiência demonstra que este tipo de abordagem deve ser fortemente supervisionada, já que a IA tende a baixar a qualidade dos resultados quanto maior a complexidade e cruzamento de domínios que uma dada solução exige [He+25b].
Este tipo de ocorrência levanta a necessidade de que as metodologias onde assentam os projetos desenvolvidos com recurso à IA como coautora devem contemplar fases pontuais e intermédias de revisão arquitetural, onde, além da validação dos resultados produzidos pela IA, deve existir espaço para a criatividade humana de forma a permitir uma aferição ou alterações que sejam potencialmente benéficas para os projetos [Rao25].

5.1.3 Implementação e programação

Já é prática comum entre uma grande parte da comunidade de programadores e engenheiros de software, que no passado recorriam a fóruns e sites dedicados para ajudar a remover bloqueios, que agora o façam com o recurso a LLMs como ChatGPT, Copilot, DeepSeek e Claude. Isto dá uma maior visibilidade da utilização da IA na fase de implementação e programação do que nas restantes fases analisadas até este ponto. Caballar [Cab24] defende que o código passa a estar contextualizado dentro de uma caixa de conhecimento gigantesca, permitindo aos sistemas inteligentes artificiais sugerirem algoritmos e soluções para casos conhe-

cidos (identificados regularmente na literatura como *Edge Cases*).
Apesar desta enorme capacidade se refletir em maiores índices de produtividade, é necessário ter atenção ao aumento da dívida técnica silenciosa provocada pela combinação de falta de experiência do engenheiro de software com falta de supervisão dos resultados da IA e com a confiança excessiva no conteúdo gerado automaticamente [Rec+24]. O que é este conceito de dívida técnica silenciosa? É o acumular de problemas e má qualidade que não são imediatamente detetáveis, sendo que, posteriormente, o seu custo de refatorização torna-se elevado, perturbando a produtividade e a capacidade de mudança futura como se pode ver pelo exemplo demonstrado na Figura 5.2. Um pouco como o efeito Icebergue, em que o que se vê fora de água engana a perceção de quem o vê, sendo que o mesmo é gigantesco e nem se consegue imaginar o que está dentro de água. Ou seja, apesar do código (aparentemente) cumprir com os requisitos, não segue os critérios de boas práticas de implementação: modularidade, escalabilidade, robustez, segurança e performance. Amansati e Jasmin [AJ25] alertam para este tipo de situações, evidenciando que o papel do engenheiro de software não fica reduzido, mas transformado: passa de um programador para um supervisor altamente especializado.

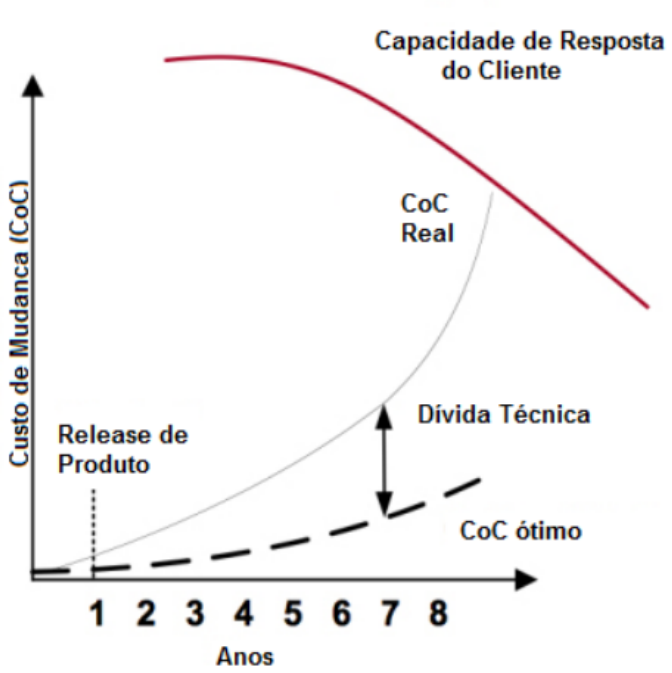

Figura 5.2: Exemplo ilustrativo da interferência da dívida técnica no custo de mudança e produtividade baseado nos gráficos de Jim Highsmith [Hig23]

5.1.4 Testes e manutenção

A componente de testes é extremamente importante para garantir a qualidade da solução a entregar. Testes bem planeados, bem descritos e corretamente efetuados garantem uma solução mais madura e robusta. Isso garante que a própria manutenção da solução digital é muito mais eficiente, reduzida e realmente eficaz. A IA consegue gerar casos de testes com cenários menos óbvios e que facilmente escapam à vista e mente humana, permitindo assim chegar mais longe e abranger uma área maior para testes [Sil21].
Quando se entra na componente de manutenção, o facto de existir inteligência que permite prever eventos com base em padrões comportamentais e métricas é uma mais valia. Além disso, a IA chega ainda mais longe: com o surgimento dos agentes autónomos, além de encontrar potenciais falhas, estes conseguem sugerir correções e alterações que evitem esses eventos futuros. Ainda mais, esses mesmos agentes conseguem despoletar ações preventivas que evitam o escalar dos problemas encontrados. Estes operam de forma contínua, efetuando uma monitorização detalhada de logs, obtendo métricas e detetando comportamentos anómalos, funcionando como mecanismos de auto-regeneração dos sistemas. o Copilot Agents, o Azure DevOps MCP Agents e o Claude Agents são exemplos de ferramentas que permitem configurar agentes para os diversos objetivos nesta área. Apesar de os modelos de IA conseguirem detetar padrões que antecipam erros ou comportamentos não desejados no software produzido, o facto de estes serem opacos impede que seja compreendido com a precisão desejada por que razão essas falhas previstas irão ocorrer. Esta falta de explicabilidade torna-se entrópica para a identificação da raiz do problema e torna muito mais complexa a implementação de correções eficazes e escaláveis. [Akh+24].

5.2 GitHub Copilot: Caso de estudo sobre coautoria no desenvolvimento de software

De forma a lançar uma vista mais realista sobre o contexto da coautoria no desenvolvimento de software entre homem e máquina e de forma a permitir, também, uma comparação com os casos práticos das próximas secções, analisa-se o caso de estudo sobre a ferramenta GitHub Copilot, que utiliza a IA para apoiar na programação de software.
O GitHub Copilot surge de uma colaboração entre a GitHub, a Microsoft e a OpenAI, recorrendo a modelos de linguagem avançados, tornando-se numa referência no mercado de ferramentas de desenvolvimento com suporte por IA. Inicialmente o

GitHub Copilot foi desenvolvido com base no modelo OpenAI Codex (um derivado do GPT-3). Atualmente, o GitHub Copilot é desenvolvido com base em modelos mais avançados da OpenAI como o GPT 4.1 e algumas variantes da família GPT 5 [Git24]. Neste caso de estudo ficará evidenciado que a IA não é apenas utilizada como um instrumento auxiliar, mas que tem parte ativa no desenvolvimento em si, participando dinamicamente na implementação de artefactos de software, apoiando decisões técnicas, criando código e efetuando sugestões para resolução de problemas.

Em perspetiva, pode-se considerar que a componente do Copilot não surge apenas para acelerar tarefas mas também para ser um membro construtivo da equipa, podendo ser considerado como um reforço cognitivo do engenheiro de software. Foi feito um estudo pelo GitHub publicado no seu blog [Kal24], tomando em consideração a perspetiva de programadores e indivíduos envolvidos na componente de programação, onde se verificou que as suas tarefas eram realizadas sensivelmente 55% mais rápido do que sem a intervenção da IA como podemos ver pela Figura 5.3, sendo que esse aumento era obtido pela facilidade com que esta conseguia propor blocos de código completo, antecipar determinados padrões de implementação e um ótimo alinhamento com as abordagens e modelos de arquitetura do tipo de projeto em que trabalhavam.

A Figura 5.3 em particular, demonstra o caso em que foram recrutados 95 programadores e divididos aleatoriamente e cronometrados na tarefa de escrita de um servidor HTTP em JavaScript. Um grupo utilizou o GitHub Copilot para efetuar a tarefa e o outro não. Nesta experiência foi medido o sucesso de cada grupo na conclusão da tarefa e o tempo que cada grupo necessitou para a terminar. O grupo que utilizou o GitHub Copilot teve uma taxa de conclusão da tarefa mais elevada (78%, em comparação com 70% no grupo sem Copilot). A diferença mais marcante foi que os programadores que utilizaram o GitHub Copilot concluíram a tarefa 55% mais rapidamente do que os programadores que não utilizaram o GitHub Copilot. Os programadores que utilizaram o GitHub Copilot demoraram em média 1 hora e 11 minutos a completar a tarefa, enquanto os programadores que não utilizaram o GitHub Copilot demoraram em média 2 horas e 41 minutos.

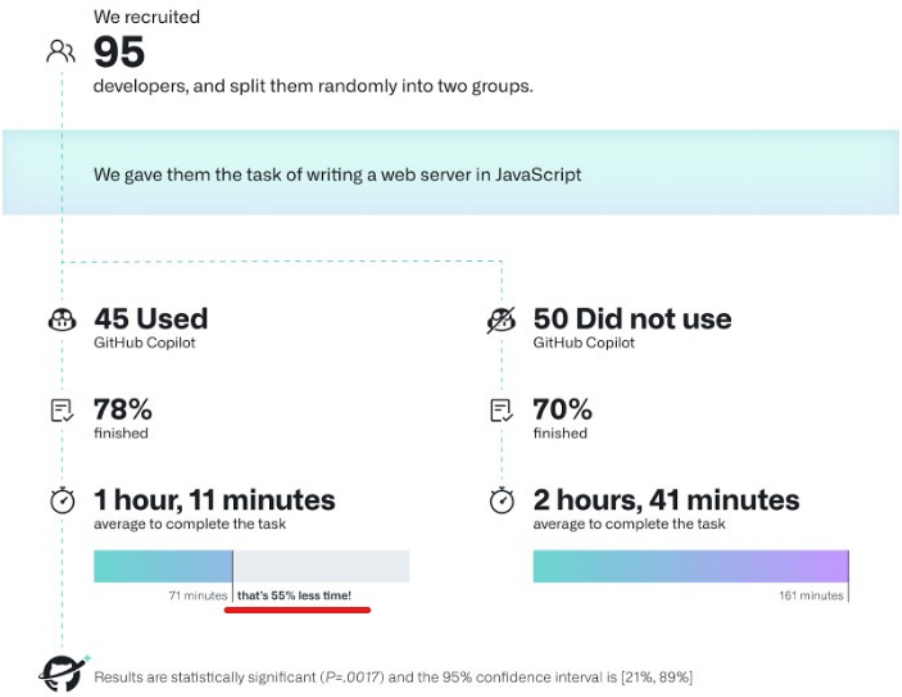

Figura 5.3: Resumo do processo e resultados da experiência do caso de estudo do GitHub Copilot para implementação de um servidor HTTP em JavaScript [Kal24]

Os dados obtidos no caso de estudo ficam reforçados ainda mais com artigos científicos como o de Vaithilingam [Vai22], onde o mesmo e os seus colaboradores defendem que as decisões dos próprios engenheiros de software são diretamente influenciadas pelas ferramentas de geração de código como o Copilot. Este evento acontece, não só porque é mais conveniente aceitar as sugestões da IA para os problemas, mas principalmente porque as soluções apresentadas são bastante plausíveis e alinhadas com as boas práticas. Como resultado consegue-se compreender a aproximação da IA ao papel de coautora no processo de implementação de software, algo como um engenheiro de software digital.
Outra componente do caso de estudo foi o levantamento realizado através de inquéritos aos engenheiros de software quanto a pontos altamente relevantes no processo de desenvolvimento de software. Os dados obtidos dão a perceção de que os indivíduos ficam mais satisfeitos, de forma geral, quanto à sua produtividade com o recurso à IA para coautoria de software, tal como podemos comprovar pela Figura 5.4.

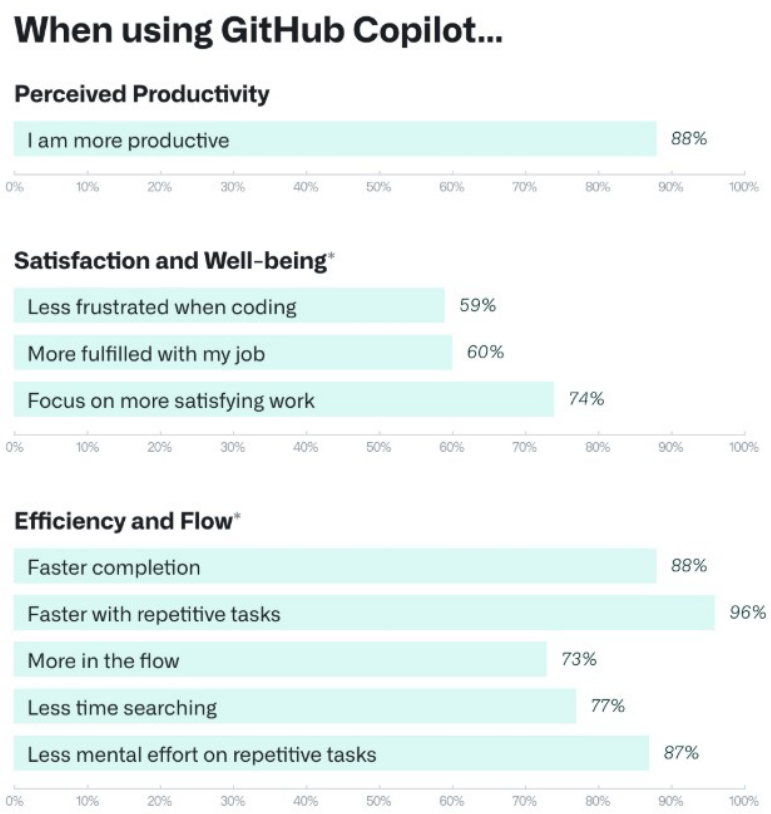

Figura 5.4: Respostas do inquérito que medem as dimensões da produtividade do engenheiro de software ao utilizar o GitHub Copilot [Kal24]

Uma outra fonte de informação relativamente a este caso de estudo, a Microsoft Research, pelas mãos de Chen e seus colaboradores [Che+21], relata que os modelos de linguagem treinados em tarefas de programação, modelos esses utilizados na base do GitHub Copilot, conseguem ser bastante eficientes quanto aos resultados produzidos e que, em determinados momentos, esses mesmos resultados são em todo semelhantes a resultados obtidos por humanos. Como Chen menciona, a IA não tem um papel apenas mecânico e repetitivo no processo de desenvolvimento de software, mas passa a ter funções mais associadas à criatividade que influenciam de uma forma muito direta o design do software desenvolvido.

O caso de estudo exposto neste trabalho comprova, de uma forma sólida, que a IA permite às ferramentas de desenvolvimento de software ultrapassarem os seus contextos tradicionais através da contribuição na criação de código e na influência que exercem sobre as decisões técnicas envolvidas nos processos de Engenharia de Software. Passa a existir uma perspetiva muito mais colaborativa e criativa, o que permite atribuir a conotação de “coautora” à IA nesses mesmos processos.

Em última análise do caso de estudo do GitHub Copilot, pode-se concluir que esta ferramenta comprova que a IA pode (e deve) ser utilizada como um parceiro criativo

e fazer parte das decisões nos processos de Engenharia de Software. Torna-se óbvio e devidamente comprovado que esta abordagem transporta um grande conjunto de benefícios, desde que exista a devida responsabilidade e ética por parte dos engenheiros de software na forma como gerem as ações e comportamento da IA no decorrer dos respetivos processos.

5.3 GitHub Copilot: Caso prático comparativo para desenvolvimento de software

O caso teórico discutido na secção anterior apresenta sustentação documental, mas, de forma a se conseguir obter uma experiência mais realista e propositada, segue nesta secção o desenvolvimento de um caso prático com a mesma ferramenta do caso de estudo: o GitHub Copilot.

5.3.1 Descrição funcional do projeto a implementar no caso prático

A implementação a efetuar segue a seguinte descrição:

- Aplicação de gestão de pedidos (tickets) com:
  - **Backend:** ASP.NET Core Web API
  - **Frontend:** ReactJS
  - **Linguagem de Base de Dados:** SQL
  - **IA Coautora:** GitHub Copilot
- O modelo funcional da aplicação de gestão de pedidos deve respeitar a seguinte descrição:
  - **Campos típicos:**
    - Id (GUID ou Int)
    - Título
    - Descrição
    - Estado (Pendente, Aprovado, Rejeitado)
    - DataDeCriação
    - DataDeAtualização
    - CriadoPor (Nome ou email)

- **Funcionalidades:**
  - Criar pedido
  - Listar pedidos
  - Atualizar estado

5.3.2 Implementação do caso prático com o IDE Visual Studio Code + extensão GitHub Copilot

O IDE escolhido para o caso prático é o Microsoft Visual Studio Code. Para se ter acesso ao GitHub Copilot, o mesmo foi instalado via Extensões de forma a ficar disponível para o desenvolvimento, tal como o GitHub Copilot Chat. Atualmente esta ferramenta já se encontra integrado nativamente no IDE. As requisições serão alternadas entre comentários no código para autogeração e prompts no respetivo chat. O IDE deve também ter instaladas as extensões de *ReactJS* e *ASP.NET Core* para se poder criar o código necessário.

Para que o Github Copilot funcione é necessário que o utilizador tenha uma conta no GitHub. As contas recomendadas para este tipo de exercício são as contas gratuitas denominadas de *Student Pack*.

O primeiro passo é criar a solução de backend. Para isso cria-se um novo ficheiro *Program.cs* numa pasta dedicada ao projeto, e começa-se por colocar o seguinte comentário no ficheiro:

// Configure a basic Web API for managing requests with CRUD endpoints

De seguida, deixa-se o Copilot implementar a sua sugestão mínima (controllers, endpoints, etc). A implementação deste primeiro passo gera de imediato o código da solução backend mínima como se pode verificar no ficheiro *Program.cs* presente no Anexo A.1.1.

O passo seguinte consiste em criar o modelo de entidades para os pedidos (requests). Para isso cria-se o ficheiro no caminho do projeto *Models/Request.cs* e coloca-se o seguinte comentário:

// Create a request entity with Id, Titulo, Descricao, Estado, DataCriacao, DataAtualizacao, CriadoPor public class Request

Tal como no passo anterior, deixa-se o Copilot implementar a sua sugestão. A implementação do segundo passo gera de imediato o código do modelo para a entidade de Request e cria o ficheiro *Models/Request.cs* contendo este o código disponibilizado no Anexo A.1.2.

Além disso, nesta fase já existe o ficheiro de contexto da base de dados *Data/RequestDBContext.cs* com o código disponibilizado no Anexo A.1.3.

Existindo já o contexto do Base de Dados, procede-se ao pedido da criação do con-

troller para a API. Para isso, insere-se o seguinte comentário:
// Create an API controller for Request with CRUD endpoints [ApiController] [Route("api/[controller]")] public class RequestsController : ControllerBase
Permite-se então ao Copilot implementar a sua sugestão. A implementação deste passo leva a que determinado código pré-gerado seja reescrito e tenha que ser verificado. Após a conclusão deste passo passa a existir o ficheiro *RequestsController.cs*.
De seguida, irá ser feito o pedido ao GitHub Copilot para lidar com a mudança de estado, ou seja, efetuar a criação de um endpoint para esse efeito. Para tal introduz-se o seguinte comentário:
// Endpoint to update the status of a request [HttpPatch("id/status")] public async Task UpdateStatus(int id, [FromBody] string newStatus)
Após o pedido, o GitHub Copilot cria o novo método com o verbo PATCH no ficheiro *RequestController.cs* como podemos verificar no Anexo A.1.4.
Nesta fase o backend encontra-se implementado. O projeto consegue ser compilado com sucesso e sem avisos. A próxima fase é a implementação do frontend.
A componente de frontend será desenvolvida em ReactJS, ou seja, o ficheiro principal será adicionado à raiz do projeto com o nome *src/App.js*. No chat do Visual Studio Code, inserimos o seguinte prompt:
// Create a React app to list, create and update requests from the ASP.NET Core API function App() . It needs to be in root.src/App.js file
Durante o processo de implementação pedido, o Visual Studio Code, com o controle do GitHub Copilot, mediante a aceitação do utilizador, instala todos os pacotes necessários e procede à respetiva programação. No final, a implementação disponibiliza as funcionalidades de listar, criar e atualizar pedidos e com um histórico de todos os passos dados pelo GitHub Copilot no processo.
Para finalizar a implementação do projeto, pede-se ao GitHub Copilot para gerar os componentes necessários no frontend. O pedido será baseado em criar um componente para cada funcionalidade descrita inicialmente, ou seja, criar pedido, listar pedidos e alterar estado dos pedidos.
De forma a obter o pretendido, é inserido o seguinte prompt no Visual Studio Code necessário para a criação dos componentes:
//React component to display a list of requests with actions to approve or reject and a React component to create requests
Durante o processo de implementação, o GitHub Copilot faz os ajustes necessários ao código pré-existente de forma a obter o resultado esperado. Durante o processo foi necessário aprovar uma mudança de código relativamente à forma como estavam a ser chamados os métodos da API do backend e bloquear uma alteração que

iria tornar-se menos bem conseguida, apesar de funcional, alterando a composição do controller da API. Nesta fase, após os diversos pedidos, torna-se notório o benefício do GitHub Copilot questionar e permitir ao engenheiro de software tomar decisões críticas, avançando apenas em conformidade com o que é selecionado. Este tipo de comportamento permite manter o controle e a responsabilidade do lado do engenheiro de software.
O resultado da implementação dos componentes fica disponível no ficheiro *src/App.js* e pode ser verificado no Anexo A.1.5.

Nesta fase pode-se proceder aos testes da aplicação gerada de forma a comprovar que a mesma cumpre com os requisitos esperados. Para tal, coloca-se em execução a API ASP.NET criada na fase de backend no *localhost*, executando o comando *dot net run* no terminal do Visual Studio Code, dentro da pasta de origem da componente de backend do projeto.
De seguida, abrindo a pasta de origem da componente de frontend, executou-se o comando *npm run dev* para executar o frontend, pelo que se obteve um erro relativamente ao endpoint configurado. O GitHub Copilot, tendo em conta os pedidos em inglês e as terminologias em português, colocou o endpoint configurado no frontend em português (http://localhost:5000/api/pedidos) em vez de inglês (http://localhost:5000/api/requests). Após a correção do endpoint no código, a aplicação abriu corretamente no browser como se pode ver na Figura 5.5.

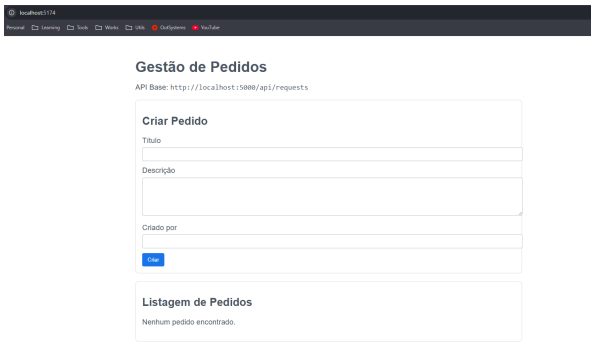

Figura 5.5: Execução da aplicação gerada com o apoio do GitHub Copilot

Após a iniciação da aplicação, testou-se a criação de um pedido com o preenchimento dos campos necessários. O sistema tanto criou o pedido como o colocou na lista de pedidos criados de forma correta como se pode verificar pela Figura 5.6.

Figura 5.6: Criação de um pedido na aplicação e validação de entrada do mesmo na listagem dos pedidos

No passo seguinte testou-se a funcionalidade de edição de um pedido, verificando-se que o mesmo ocorre de forma correta e é apresentado na listagem com as edições submetidas tal como se pode ver pelas Figura 5.7 e Figura 5.8.

Figura 5.7: Edição de um pedido na aplicação

Figura 5.8: Apresentação correta de pedido editado na aplicação

A funcionalidade de eliminar pedidos também foi testada com sucesso, sendo que o pedido existente desaparece da listagem conforme o esperado e demonstrado na Figura 5.9.

Figura 5.9: Eliminação de pedido na aplicação

Por fim, testou-se a criação de múltiplos pedidos e verificou-se que eram apresentados de forma correta na listagem, obtendo sucesso na sua totalidade conforme comprovado pela Figura 5.10.

Figura 5.10: Criação de múltiplos pedidos na aplicação com correta visualização na listagem

De forma a comprovar que os dados estavam bem gerados no backend, invocou-se diretamente a API de requests para comprovar os dados provenientes da mesma. Os dados estavam em concordância com o frontend conforme se verifica na Figura 5.11.

Figura 5.11: Verificação dos dados existentes na aplicação com a chamada à API de backend

No final, verificou-se que tudo o que foi pedido funciona devidamente.

5.3.3 Avaliação dos resultados do caso prático com GitHub Copilot

Pode-se comprovar que o GitHub Copilot é, de facto, um copiloto na implementação de software, sendo que valida qualquer ação mais crítica com o engenheiro de software de forma a ser este a tomar as decisões. Torna-se um acelerador do processo, sendo que, todo este projeto foi implementado em apenas 16h já com as pequenas correções e ajustes necessários. Para base de comparação foram pedidas as estimativas para a execução de um projeto semelhante a um programador pleno e um programador sénior, neste caso, ambos de ASP.NET + React. As diferenças são bastante elevadas conforme se pode verificar pela Figura 5.12.

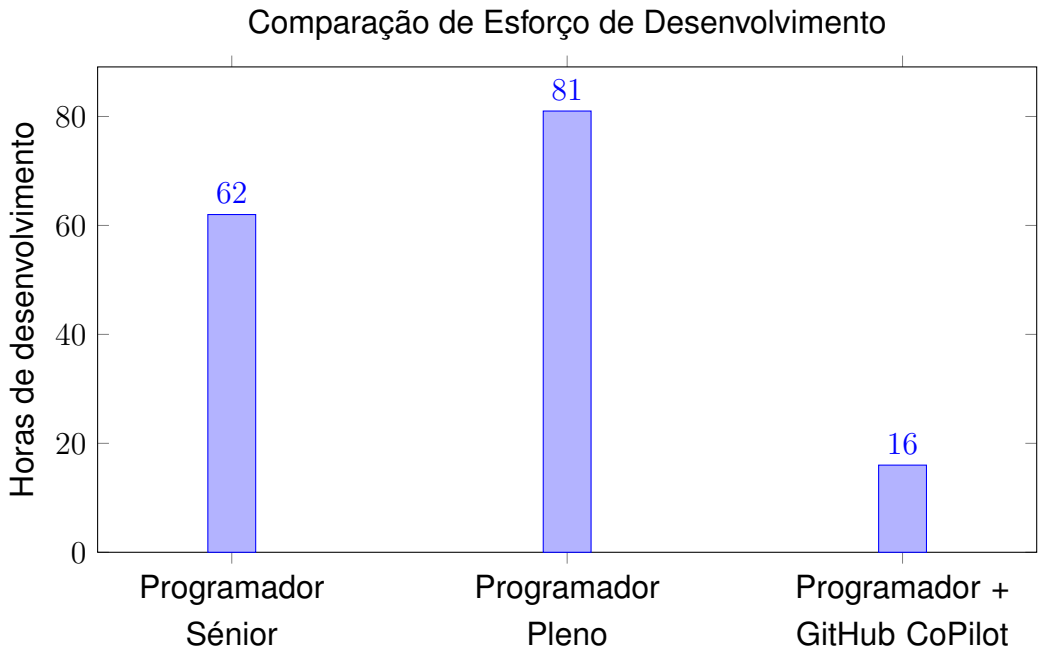

Figura 5.12: Esforço estimado: programador Sénior, programador Pleno e programador + GitHub Copilot

O GitHub Copilot permite ganhos de produtividade de cerca de 80% em relação a um programador pleno e de cerca de 74% relativamente a um programador sénior. Estes valores suportam de uma forma bastante robusta a capacidade que a IA tem de, no papel de coautora, aumentar a produtividade de forma significativa. A aplicação é simples, mas demonstra a capacidade de geração de código e sugestão que o GitHub Copilot tem para projetos que contemplam mais do que uma linguagem. Neste caso, foi capaz de gerar o código para o backend (ASP.NET) e o frontend (React) de forma minimamente organizada e compreensível. De notar que o código gerado, apesar de minimamente organizado, não tem qualquer comentário, o que, em casos de maior complexidade pode reduzir a sua compreensão. Para análise,

podem ser consultados o exemplo do código de frontend no Anexo A.1.5, onde se pode inspecionar o ficheiro *App.jsx* e os exemplos de código de backend nos Anexos A.1.1, A.1.2, A.1.3 e A.1.4 onde se podem verificar os ficheiros *Program.cs*, *Request.cs*, *RequestDBContext.cs* e *RequestController.cs* respetivamente.
As restantes conclusões serão apresentadas em contexto com todo o restante trabalho no Capítulo 6 - “Conclusões e Trabalhos Futuros”.

5.4 OutSystems Mentor: Caso prático sobre desenvolvimento de software assistido por IA

De forma a tornar o trabalho mais robusto, apresenta-se um caso prático sobre um produto desenvolvido por uma empresa com origem portuguesa, a OutSystems. Apesar de ser sediada atualmente em Boston, EUA, os três fundadores e mentores da empresa são portugueses e esta surgiu inicialmente em Portugal, sendo um dos unicórnios nacionais.
O OutSystems Mentor App Generator surge a comprimir todas as fases do SDLC refletindo-se nas seguintes características:

- Uma prototipagem rápida e eficiente.
- Um desenvolvimento muito mais célere do que no paradigma tradicional.
- Testes muito mais eficientes com a criação de utilizadores que representam as personas.
- Uma capacidade de correção e *fine tunning* acima do normal com a capacidade de modelação da aplicação desenvolvida diretamente no IDE da OutSystems (ODC Studio).
- Um *Go Live* extremamente ágil, já que existe o ODC Portal que permite efetuar um controle total de deploys entre ambientes.
- Versionamento, validação de estabilidade e uma velocidade de deployment acima da média com zero downtime [Out25b] (potenciado pela tecnologia baseada em *kubernetes*).

Todas estas características sugerem um caminho robusto para a redução dos tempos de implementação de software e para uma maior robustez e qualidade dos produtos finais.

5.4.1 Motivo de escolha da plataforma OutSystems

A Gartner já evidenciou que, até 2026, as ferramentas Low Code irão representar 75% do desenvolvimento de novas aplicações, sendo este valor potenciado pela pressão que as organizações e empresas sofrem para evoluírem e se adaptarem às novas tendências [Inf22], em que a OutSystems é líder há nove anos consecutivos [Out25a]. Além disso, a comunidade de programadores OutSystems já ultrapassa o meio milhão de indivíduos [Wir23]. Pode ser algo recente (nasceu em meados de 2000, quando ainda nem se tinha cunhado o termo Cloud), mas já é mais uma *trend* do que um nicho. Através do OutSystems Mentor App Generator, cada vez mais vem a ser reforçado o conceito de *Citizen Developer* (elemento de uma organização que, sem conhecimento nem formação na área de software, consegue, com recurso a plataformas Low Code e No Code, criar e modificar aplicações [Man25]), o que ainda vinca mais o papel da IA em toda esta revolução, visto ser esta a capacitadora deste conceito.
O facto de estarmos perante um produto *Enterprise* em que as licenças para empresas é pago, não podemos negar que as características e capacidades potenciadas pela IA estão presentes. Muitas das grandes evoluções tecnológicas são patenteadas e de pouco acesso, sendo que, neste caso, basta ter uma conta *Free* na OutSystems que permite ao engenheiro de software ter acesso à ferramenta OutSystems Mentor App Generator.

5.4.2 Introdução à plataforma OutSystems

A OutSystems disponibiliza, ao nível do mercado empresarial (espectro de médias e grandes empresas), uma plataforma de desenvolvimento de software no paradigma RAD (*Rapid Application Development*) seguindo a vertente Low Code e uma ferramenta em específico que é utilizada para este caso prático: o OutSystems Mentor App Generator. Esta ferramenta consegue gerar aplicações na plataforma baseadas apenas nas descrições funcionais das mesmas. O OutSystems Mentor App Generator é uma ferramenta web com uma interface muito simples e intuitiva, conforme se pode ver na Figura 5.13.

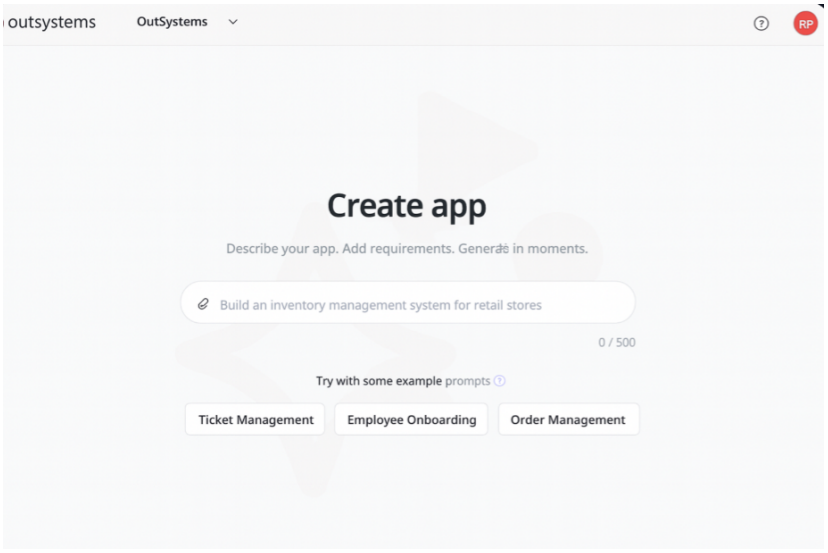

Figura 5.13: Página principal da ferramenta OutSystems Mentor App Generator

Durante a apresentação deste caso prático serão reforçadas diversas características discutidas nesta dissertação. Será documentado que o engenheiro de software necessita, além de supervisionar, alterar partes do programa implementado de forma a ajustar-se corretamente ao ecossistema digital do cliente, garantindo assim que não gera dívida técnica escondida. Este caso prático tenta demonstrar com clareza que a IA ainda não é totalmente autónoma mas que já tem um grau de autonomia bastante relevante, além de demonstrar que as competências do engenheiro de software estão a mudar.

5.4.3 Utilização do OutSystems Mentor App Generator com prompt para geração de aplicação

O processo de utilização da ferramenta OutSystems Mentor App Generator demonstra uma abordagem muito mais abrangente do que uma simples submissão de um prompt em linguagem natural. O processo é muito mais estruturado e dividido em etapas, começando na definição de requisitos, e seguindo de forma conduzida por diversos passos intermédios que asseguram a coerência e a totalidade da aplicação a gerar. O engenheiro de software começa por disponibilizar uma descrição dos requisitos funcionais e não funcionais da solução. De seguida segue-se a fase de validação dos conceitos, onde a ferramenta verifica a consistência dos requisitos inseridos e solicita esclarecimentos no caso de identificação de ambiguidades. Após esta validação o OutSystems Mentor App Generator avança para a geração automática do modelo de dados, mantendo a comunicação com o engenheiro de software. Por fim, a ferramenta disponibiliza a solução na sua totalidade, desde

modelo de dados, lógica, ecrãs e todas as validações de utilização. Este processo demonstra que o OutSystems Mentor App Generator não é uma ferramenta que se limita a gerar código, sendo que tem incorporados mecanismos de interpretação, validação e modelação que tornar a IA num recurso bastante colaborativo no processo de implementação de software. A arquitetura de informação que suporta estas capacidades do OutSystems Mentor App Generator pode ser verificada na Figura 5.14.

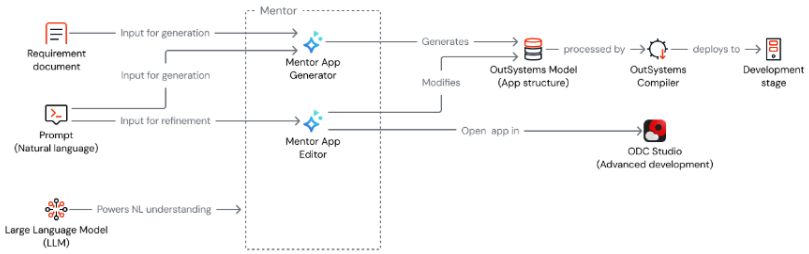

Figura 5.14: Detalhes do funcionamento da ferramenta OutSystems Mentor App Generator [Out25b]

A forma de interagir com esta ferramenta é muito semelhante a ferramentas como o ChatGPT, Microsoft Copilot, DeepSeek e outras: é disponibilizado um prompt, com um conjunto de sugestões, onde o utilizador insere o contexto funcional (até 500 caracteres) e onde pode submeter ficheiros com os requisitos nos formatos txt, docx ou pdf, sendo que os respetivos ficheiros não devem ultrapassar 4MB. Esta informação é disponibilizada nos requisitos de escrita da ferramenta conforme a Figura 5.15.

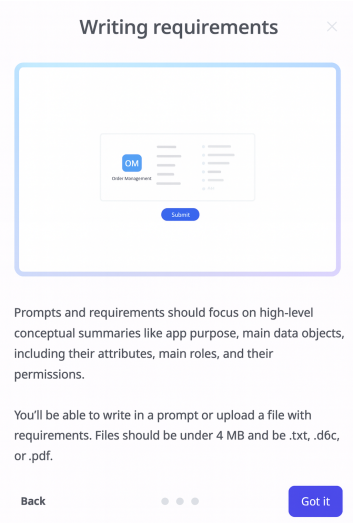

Figura 5.15: Detalhes dos critérios de inserção de requisitos da aplicação a gerar no OutSystems Mentor App Generator

Os prompts devem seguir um conjunto de regras de boas práticas para garantir que o OutSystems Mentor App Generator consegue obter uma solução o mais refinada possível e de acordo com o pretendido. São recomendadas as seguintes boas práticas [Out26]:

- Ser claro e conciso - incluir o nome da aplicação, a finalidade e a funcionalidade principal numa ou duas frases.
- Estruturar o modelo de dados - mencionar entidades e atributos sempre que forem conhecidos.
- Adicionar detalhes específicos das funções - mencionar as funções e as permissões sempre que forem conhecidas.
- Incluir preferências de estilo - mencionar temas, cores ou requisitos visuais sempre que forem conhecidos.
- Especificar as fontes de dados e integrações - mencionar sistemas ou fontes de dados externos quando relevantes.

Tendo isso em conta, para este caso prático, foi gerado um prompt com recurso a uma ferramenta de IA, posteriormente validado por um engenheiro de software, para criar uma aplicação de pedidos de suporte.
O texto a seguir foi utilizado como prompt para a geração da aplicação de pedidos de suporte do caso prático apresentado neste trabalho:

> *A modern, scalable Helpdesk web app where authenticated customers submit support tickets with title, detailed description, category, priority and attachments. The system timestamps tickets, links them to users, and assigns technicians or teams. It provides technician and admin dashboards, metrics, email notifications, advanced search and filters, multilingual responsive interface, integrated knowledge base, GDPR - compliant security, API extensibility and satisfaction feedback.*

No caso prático anterior do GitHub Copilot as instruções também foram em Inglês mas os nomes dos artefactos foram produzidos em Português. Neste caso prático do Mentor isso não ocorreu porque a plataforma OutSystems já tem *built in* o conceito de multi-idioma, podendo ser facilmente inseridas as traduções para qualquer idioma do interesse do cliente. Inserindo o prompt na ferramenta e avançando para o passo seguinte, o OutSystems Mentor App Generator demora apenas alguns segundos a mostrar ao utilizador os conceitos que irão surgir no contexto da aplicação, tal como o *birds-eye-view* de toda a solução conforme a Figura 5.16.

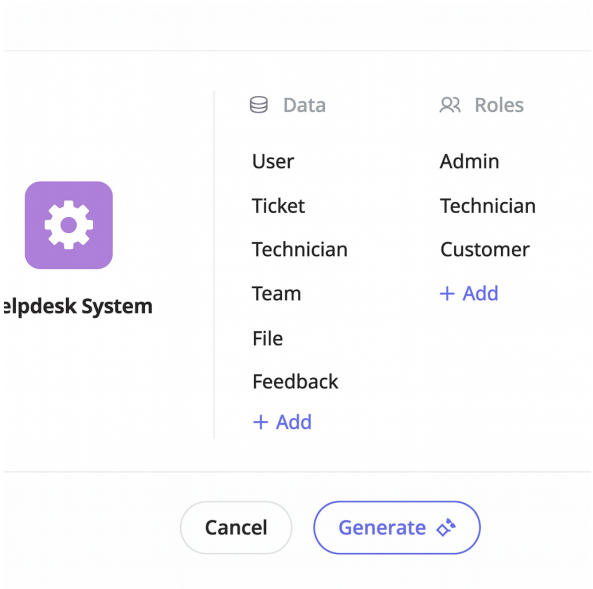

Figura 5.16: Passo de validação de conceitos de uma aplicação a gerar pelo OutSystems Mentor App Generator

Clicando no botão "Generate", o OutSystems Mentor App Generator dá início à geração da aplicação, dando feedback em tempo real no ecrã web em que está a ser executado sobre em que passo da geração se encontra. Pode-se ver um exemplo desse feedback na Figura 5.17.

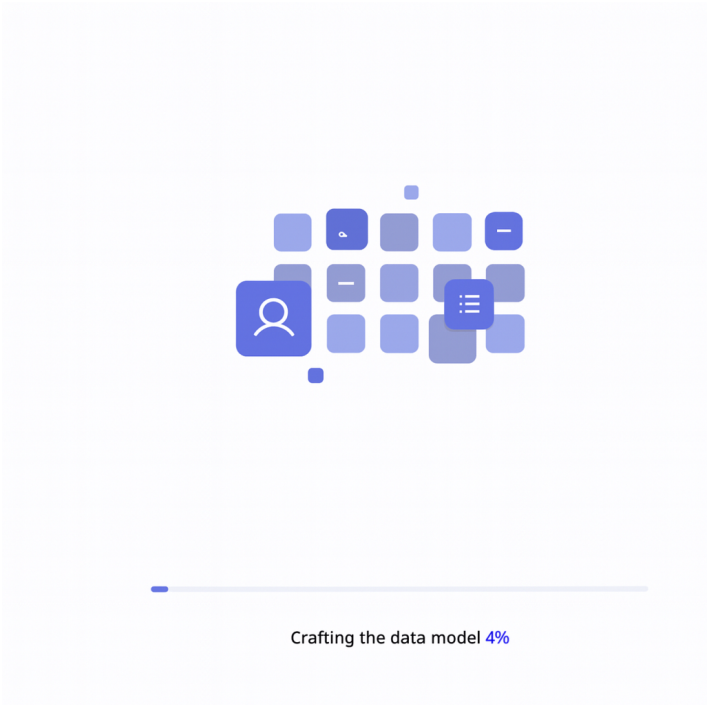

Figura 5.17: Feedback do OutSystems Mentor App Generator no passo de geração do modelo de dados

Em 3 minutos, surge no ecrã web o overview aplicacional gerado, (como o exemplo da Figura 5.18) sendo que, em caso de cumprir com o desejado, basta apenas clicar no botão "Publish", publicando a aplicação no stage de desenvolvimento da instância OutSystems ODC.

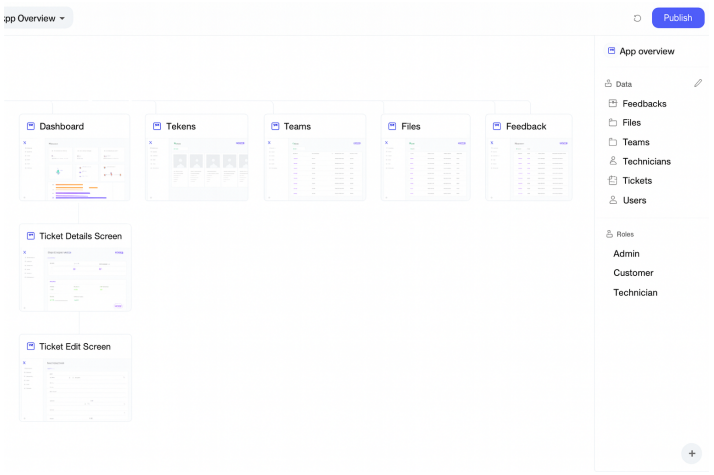

Figura 5.18: Overview da aplicação gerada pelo OutSystems Mentor App Generator a partir do script disponibilizado

Quando a publicação terminar, o botão premido anteriormente é substituído pelo botão de "Preview". Este novo botão deve ser premido para se poder efetuar uma primeira navegação pela aplicação e verificar as suas capacidades funcionais. Rapidamente se verifica que o OutSystems Mentor App Generator cria, além da aplicação, acessos para testes: Na página de login surgem os botões para aceder à aplicação com utilizadores fictícios associados aos diferentes perfis que a aplicação contempla. Pode-se ver o exemplo da aplicação gerada neste caso prático na Figura 5.19.

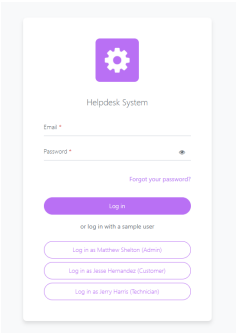

Figura 5.19: Página de login da aplicação gerada pelo OutSystems Mentor App Generator

Da mesma forma que se pode efetuar o overview funcional, o mesmo se pode fazer

do ponto de vista técnico. Acedendo ao IDE da plataforma OutSystems ODC (o ODC Studio), conectado à instância onde a aplicação foi gerada, pode-se verificar que a mesma encontra-se lá disponibilizada da mesma forma que as restantes aplicações. Tal é exemplificado na Figura 5.20.

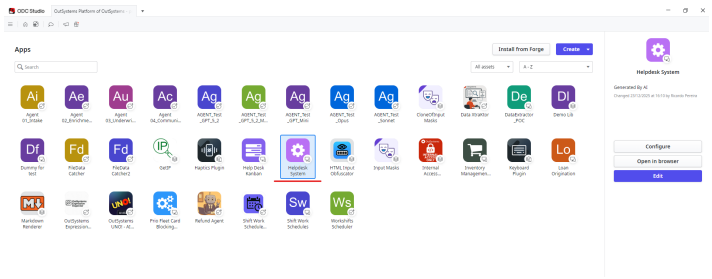

Figura 5.20: Aplicação gerada pelo OutSystems Mentor App Generator no IDE OutSystems ODC Studio

Pode-se verificar, abrindo a aplicação no IDE, que todo o código foi gerado, desde Entidades (abstração de tabelas das base de dados - Figura 5.21), funções (tanto *server side* como *client side* - Figura 5.22) e ecrãs web - Figura 5.23.

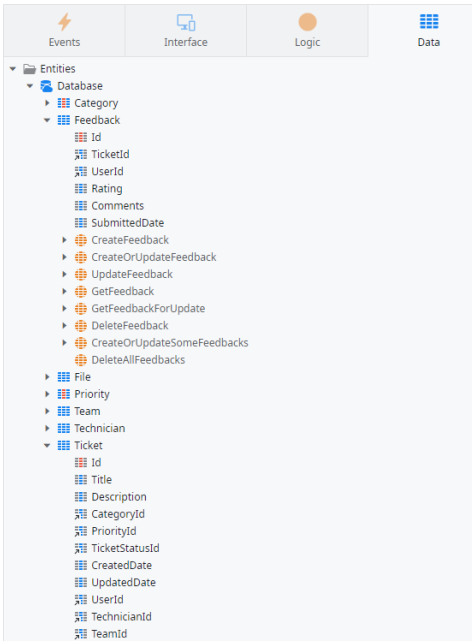

Figura 5.21: Representação das Entidades da base de dados criadas na aplicação gerada pelo OutSystems Mentor App Generator

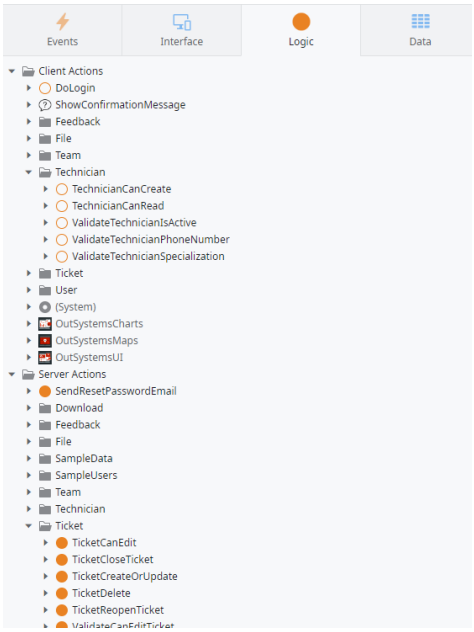

Figura 5.22: Representação das funções (conhecidas em OutSystems como ações) *server side* e *client side* criadas na aplicação gerada pelo OutSystems Mentor App Generator

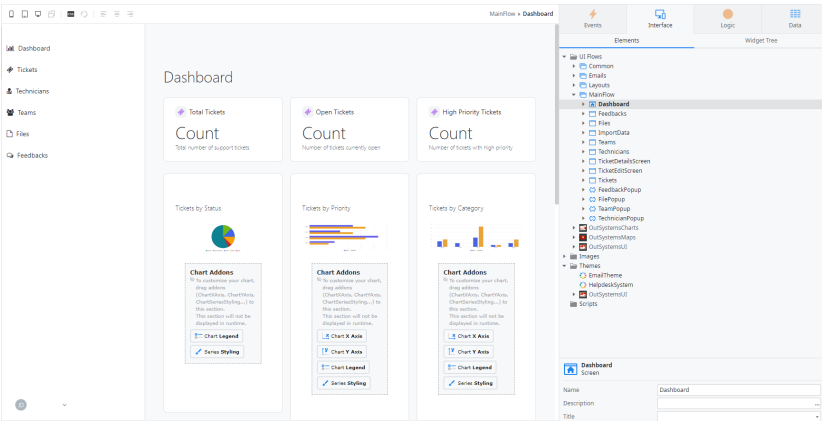

Figura 5.23: Representação dos ecrãs web criados na aplicação gerada pelo OutSystems Mentor App Generator

5.4.4 Avaliação dos resultados com o OutSystems Mentor App Generator

Apesar da aplicação gerada seguir o contexto pedido do script inserido no Mentor App Generator e de ter criado já acesso aos perfis de testes, posteriormente é necessário efetuar o denominado *fine tuning* de forma a garantir que todos os pormenores dos requisitos funcionais e não funcionais são cumpridos (além de que

é necessário remover os acessos das personas para testes na página de login de forma à aplicação poder ser executada em ambientes produtivos). Há pequenos ajustes que podem ser feitos diretamente no OutSystems Mentor App Generator antes da sua publicação, permitindo assim uma geração mais ajustada com o esperado.

Analisando a aplicação com atenção e pensando num contexto empresarial, surge uma preocupação: a integração com sistemas já existentes ou com componentes já implementados na fábrica de software. Isto refere-se à forma como o OutSystems Mentor App Generator consegue criar soluções que se complementem e complementem um ecossistema já existente.

A forma como o OutSystems Mentor App Generator gera aplicações segue um princípio de desenvolvimento *standalone*, ou seja, segue as boas práticas de arquitetura, implementa o código de forma adequada, mas como se a aplicação vivesse sozinha e sem que se tivesse que integrar com partes ou domínios já existentes. Por exemplo, para a aplicação gerada no caso prático, um cenário em que já existem as entidades de Técnicos e Equipas noutra aplicação do ecossistema, devem-se reaproveitar os desenvolvimentos pré-existentes, evitando assim replicação de dados, conceitos, domínios e um aumento proporcional de complexidade da solução geral. No caso da plataforma OutSystems, os reajustes e adaptações são rápidos e facilmente escaláveis, já que a mesma assenta no paradigma Low Code, onde existe uma grande transformação da complexidade textual da programação num contexto mais visual.

Este tipo de comportamento evidencia, que, pelo menos na atualidade, a IA pode não obter uma total autonomia no processo de programação. Mais uma vez, fica salientado que a IA deve ser contemplada como uma coautora e sempre como um parceiro do ser humano, já que este deve sempre verificar o resultado e ajustar o que for necessário.

Este tipo de capacidade demonstra, que, apesar da não total autonomia, se consegue ter um aumento de produtividade acentuado. Além de permitir acelerar o desenvolvimento, o OutSystems Mentor App Generator pode ser utilizado como uma ferramenta de prototipagem, levando ao conceito de "falhar rápido para corrigir rápido", munindo assim as equipas de projeto de capacidade para obter soluções mais objetivas e que cumpram com a necessidade dos clientes de forma mais rápida. A OutSystems declara que é possível efetuar uma redução de até 60% no tempo de desenvolvimento de aplicações na plataforma OutSystems com o recurso ao OutSystems Mentor App Generator através da automatização de tarefas tediosas e capacitando a equipa para entregar consistentemente aplicações de alta qualidade com maior eficiência [Out25c]. Para validar empiricamente as afirmações de

produtividade divulgadas pela OutSystems, foi conduzida uma recolha estruturada de estimativas junto de profissionais com experiência comprovada na plataforma. Foram selecionados dois programadores OutSystems, um sénior e um intermédio, de forma intencional, garantindo diversidade de maturidade técnica e exposição prática ao desenvolvimento em ODC. O objetivo não foi produzir valores estatisticamente generalizáveis, mas sim obter uma referência comparativa credível, baseada em conhecimento tácito acumulado, que permitisse confrontar as métricas divulgadas pela plataforma com a perceção de especialistas no terreno. Esta abordagem é comum em estudos de validação exploratória, onde a experiência profissional funciona como proxy para estimativas realistas de esforço e complexidade. As respetivas estimativas podem ser consultadas na Tabela A.3 e na Tabela A.4 presentes em anexo na secção de Dados Complementares.
Podem-se analisar as comparações entre as estimativas dadas pelos programadores e a média prevista para o OutSystems Mentor App Generator no gráfico da Figura 5.24.

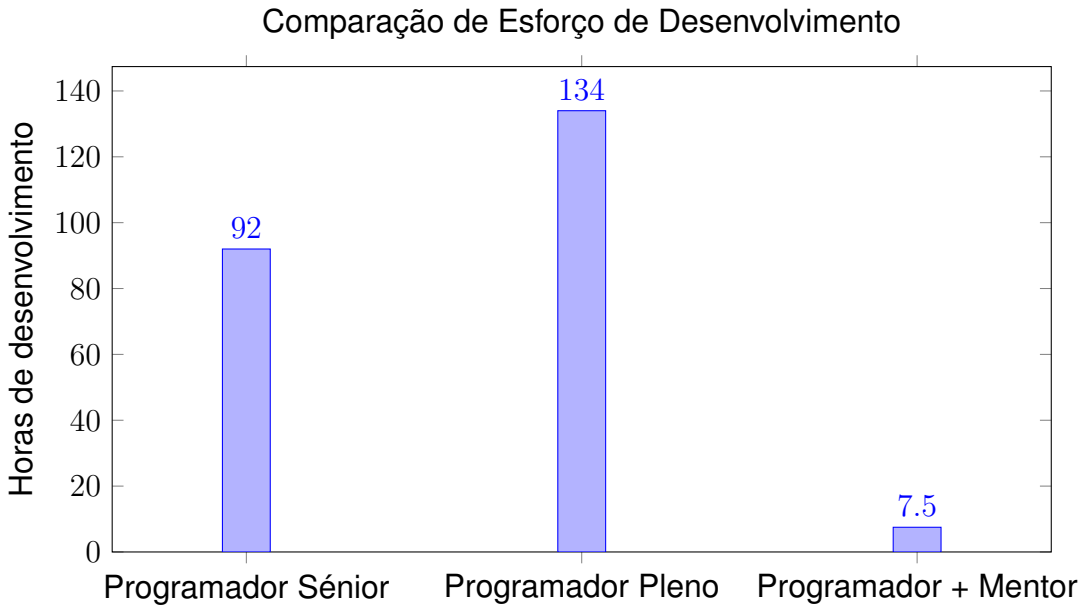

Figura 5.24: Esforço estimado: programador Sénior, programador Pleno e programador + OutSystems Mentor App Generator

Rapidamente se conclui que, para este caso prático, os ganhos de produtividade são superiores ao que é declarado pela OutSystems. Esse desvio é compreensível tendo em conta o grau de complexidade da aplicação do caso prático, sendo que o valor de 60% declarado pela OutSystems contempla uma média para diferentes graus de complexidade de implementação e variados casos de diferentes mercados. Relativamente ao caso prático, temos um ganho de 91.85% relativamente a

um programador sénior e de 94.40% em relação ao programador pleno se usarmos o OutSystems Mentor App Generator para gerar aplicações na plataforma OutSystems em vez de programar tudo manualmente no ODC Studio. Estes dados podem ser analisados com detalhe na Tabela 5.2.

Tabela 5.2: Vantagem percentual do Mentor App Generator face ao desenvolvimento manual

Perfil	**Sem IA (h)**	**Mentor (h)**	**Redução / Aceleração**
Sénior	92	7.5	91.85% (12.27×)
Pleno	134	7.5	94.40% (17.87×)

Segundo os dados da OutSystems provenientes das suas análises [Out25c], os valores comparativos obtidos entre desenvolvimento tradicional (sem IA) e o OutSystems Mentor App Generator não são muito diferentes dos valores que se obtiveram no caso prático como se pode verificar pela Tabela 5.3.

Tabela 5.3: Comparação de produtividade entre desenvolvimento tradicional e o OutSystems Mentor App Generator [Out25c]

Critério	**Desenvolvimento Tradicional**	**Mentor App Generator**	**Diferença (%)**
Tempo para criar modelo de dados	3 horas	30 minutos	-83%
Tempo para gerar o UI	4 horas	45 minutos	-81%
Tempo para lógica de negócio básica	5 horas	1 hora	-80%
Integração com sistemas externos	6 horas	1,5 horas	-75%
Tempo total para app funcional	18 horas	3,75 horas	-79%

Este caso prático consegue facilmente demonstrar o avanço tecnológico e de processos que a IA consegue alavancar, permitindo executar tarefas no âmbito da Engenharia de Software de forma muito mais rápida do que antes da sua existência neste paradigma. A IA deve ser tida em conta como um participante ativo neste ecossistema de desenvolvimento de software totalmente reinventado. A forma como a IA potencia a velocidade de implementação, a acessibilidade e o potencial de inovação são extremamente profundos e tangíveis, sendo o OutSystems Mentor App Generator um grande exemplo desse cenário.

5.5 Análise do grau de autonomia da IA e agentes nos processos de desenvolvimento de software

Existe um grande conjunto de investigadores e líderes tecnológicos que enriquecem com muitos contributos a reflexão sobre a posição da IA nos processos de

implementação de soluções digitais. Uma via muito proeminente que expõe vários destes investigadores e líderes é o canal de *podcasts* de Lex Fridman, onde se discutem diversos temas relacionados com a IA e de que forma esta se posiciona no presente e no futuro. Aqui, vários dos seus entrevistados reforçam a fase transitiva atual em que a IA deixa de ser apenas um assistente e passa a ser coautora inteligente com capacidade de compreender o contexto em que está inserida e antecipar as necessidades, conseguindo propor soluções interessantes para os problemas a que são sujeitas. No *podcast* de Alex Fridman em que entrevista Sam Altman, este reforça que os atuais modelos demonstram uma competência emergente na decomposição de problemas complexos em passos lógicos, mesmo que mantendo a necessidade de supervisão humana [Alt23]. Além disso, Demis Hassabis defende e reforça que o desenvolvimento de software é cada vez mais como uma conversa entre humanos e IA, sendo que a autonomia desta deve ser compreendida como um processo contínuo, no qual os sistemas são muito eficazes em tarefas curtas e bem definidas, mas que ainda apresentam alguma dificuldade nas tarefas mais longas e complexas [Has23].

Os casos práticos revelaram alguns pontos a ter cuidado: as ferramentas de IA ainda não conseguem ser completamente autónomas, podendo gerar problemas, muitas das vezes ocultos à primeira vista. O facto de não conseguirem gerar conteúdo perfeitamente alinhado com um ecossistema digital já existente ou porque não conseguem extrapolar corretamente todos os requisitos funcionais e não funcionais do que lhes é transmitido em linguagem natural, leva a que seja necessária uma supervisão humana de forma a que não exista corrupção do objetivo final e para evitar a geração de dívida técnica. Isto salienta o facto de que a IA deve ser coautora, mas nunca totalmente autónoma e também evidencia a necessidade de reajuste nas competência do engenheiro de software, tendo este de ser capaz de lidar com este tipo de comportamentos da sua ajudante, a IA [Far24], surgindo novas competências associadas à sua compreensão e supervisão.

No estudo lançado pela METR [MET25] facilmente se compreende que, quanto mais longas e complexas as tarefas delegadas na IA forem, maior é a necessidade de supervisão humana, já que a taxa de sucesso reduz drasticamente, passando de limites de cerca de 80% de sucesso para limites de sucesso de cerca de 50% conforme se pode ver nas Figura 5.25 e Figura 5.26 comparativamente. Estas figuras pretendem demonstrar a capacidade que as IAs têm para executar tarefas no âmbito da Engenharia de Software com diferentes níveis de fiabilidade, permitindo ao leitor compreender de uma forma objetiva os limites da autonomia das respetivas IAs. Em maior detalhe, podemos ver na Figura 5.25 o horizonte temporal para o qual diferentes modelos conseguem finalizar 50% das tarefas necessárias com

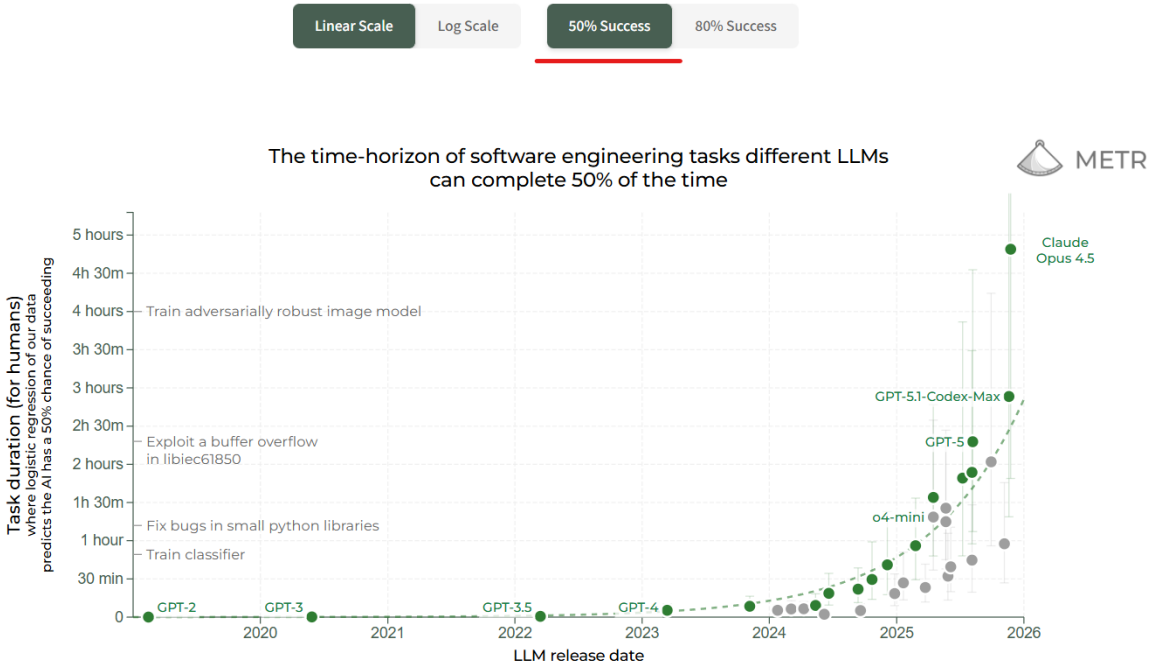

Figura 5.25: Linha de 50% de sucesso de uma IA para as duração prevista das tarefas

sucesso, disponibilizando a linha de base que serve para indicar o ponto em que a IA tem apenas uma probabilidade moderada de sucesso. No eixo horizontal está representada a evolução temporal dos modelos (desde o GPT-2 até aos modelos de 2026) e no eixo vertical encontra-se a duração estimada (em horas) para a execução da mesma tarefa mas por humanos. Este gráfico demonstra que, com o aumento da complexidade e tempo de execução das tarefas o grau de autonomia da IA desce. Na Figura 5.26 o tipo de análise é o mesmo da Figura anterior mas considerando o limite de 80% de sucesso, onde se consegue compreender que só as tarefas mais simples e curtas e com menor ambiguidade conseguem obter altos níveis de fiabilidade. Analisando as duas Figuras consegue-se compreender que, apesar da evolução dos modelos, a eficácia e autonomia da IA ainda é fortemente dependente da duração, estrutura e complexidade das tarefas, reforçando a ideia da necessidade de manter uma supervisão cuidada e contínua por parte de engenheiros de software relativamente aos raciocínios e resultados da IA.
Tudo isto evidencia a necessidade em investir agressivamente na reeducação dos engenheiros, evitando assim travar o progresso da IA no universo da Engenharia de Software. Isso pode ser obtido com recurso a *frameworks* de gestão de processos de implementação de software mais robustas e com investigação contínua para garantir que a IA se torna o mais transparente possível, altamente confiável e totalmente alinhada com os valores legais e éticos esperados pelos humanos.

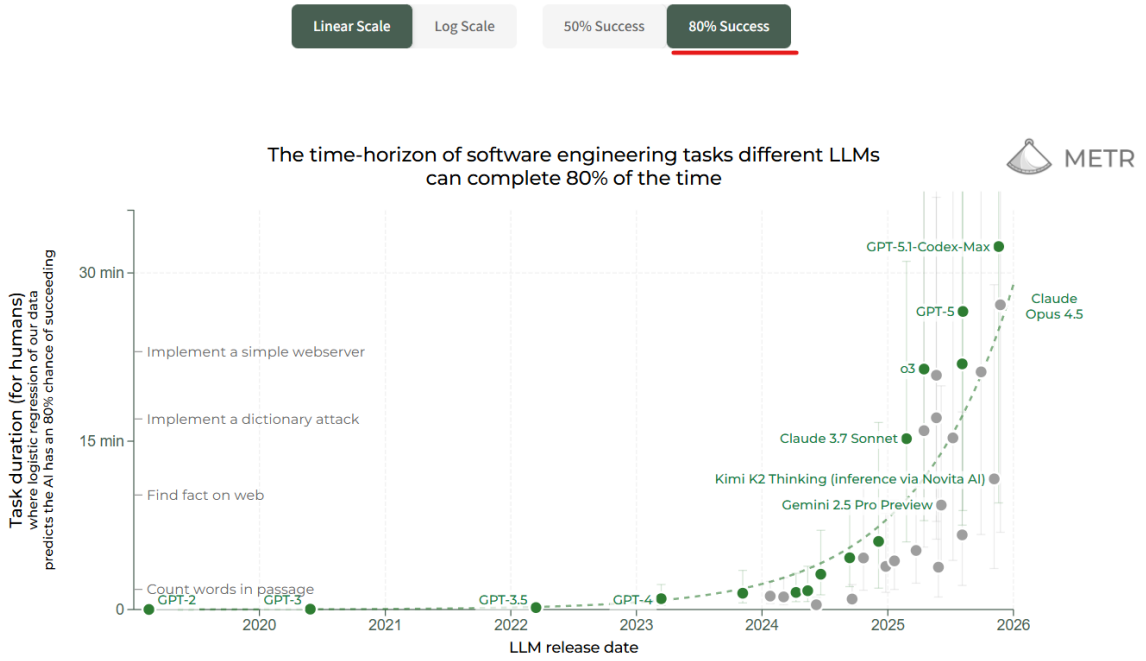

Figura 5.26: Linha de 80% de sucesso de uma IA para as duração prevista das tarefas

Os artigos, os documentos analisados, o caso de estudo e os casos práticos apresentados demonstram que já não há volta ao passado. O futuro segue com o apoio da IA como um copiloto incansável, sempre com enorme performance e em constante aprendizagem para nos ajudar cada vez mais. No entanto, o cenário ainda necessita de amadurecimento, já que não se consegue obter uma total autonomia para resultados perfeitos.

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Conclusões

Neste trabalho o objetivo foi conseguir analisar de uma forma sistemática e crítica a evolução da penetração da Inteligência Artificial nos processos de desenvolvimento de software e de que forma esta se tornou vista como coautora de sistemas digitais e tecnológicos de software. Pretendeu-se dar visibilidade à transformação, desde a base até ao mais alto nível, que a IA está a provocar nos paradigmas conhecidos da Engenharia de Software. Durante o processo de investigação e análise documental realizado, a par com o desenvolvimento do caso de estudo e dos casos práticos, mostrou-se que a IA já não é apenas uma ferramenta auxiliar para redução de tempos de execução e passa a ter um papel muito mais incisivo, com tomadas de decisão, com geração de conteúdo com um elevado grau de autonomia em diversas etapas diferentes do ciclo de vida de implementação de software, desde a interpretação inicial de requisitos até à componente de manutenção de soluções já implementadas. Pode-se comparar esta mudança de paradigma à mudança inserida no passado pelo surgimento das metodologias ágeis ou, até com o nascimento do POO (Programação Orientada a Objetos), tendo qualquer um destes eventos tido a capacidade de mudar por completo as regras do jogo e trazer escalabilidade para o universo do desenvolvimento de software. Espera-se que o impacto da IA seja ainda maior que os eventos antecessores aqui enumerados, até porque esta não interfere apenas com os processos, mas também com competências e perspetivas, alterando por completo determinadas responsabilidades e estruturas empresariais.

Durante o processo de análise efetuado para a realização deste trabalho encontraram-se três vetores referenciais para a caracterização da IA como coautora em processos de desenvolvimento de software:

- A capacidade que a Inteligência Artificial tem para interpretação da linguagem natural e transformação de discursos humanos em artefactos concretos.
- A autonomia cada vez mais incremental na geração de código de programação.
- A capacidade cada vez mais transversal da Inteligência Artificial em se integrar em todos os ciclos de vida de um projeto/produto de software, apoiando e interferindo tanto nas tarefas mais rudimentares como em tomadas de decisão.

Estes vetores identificados acima conseguem provar que a IA está a transformar-se em algo mais do que um acelerador de produtividade, passando a ser um agente muito mais presente e efetivo na colaboração com a disciplina de Engenharia de Software, demonstrando cada vez mais capacidade para participar em implementações de maior complexidade.
O facto de se delegar algo com um elevado grau de autonomia numa ferramenta que não pode ser diretamente responsabilizada levanta enormes questões nos panoramas ético e legal. Durante este trabalho procurou-se demonstrar as quatro áreas de maior preocupação neste âmbito relativamente à utilização da Inteligência Artificial como coautora, sendo elas a propriedade intelectual, responsabilidade nas falhas, enviesamento de resultados e igualdade de acesso [Roc+25]. Toda esta complexidade comprova que não é possível avançar para este novo paradigma de utilização da IA como coautora sem ter uma *framework* robusta e bem sustentada ao nível ético e legal.
Quando se compara o caso prático do GitHub Copilot ao caso prático do OutSystems Mentor App Generator, compreende-se de imediato que existe uma diferença grande ao nível de abstração. O Copilot opera numa vertente muito mais próxima do código, enquanto que o Mentor opera a um nível mais de modelo e aplicação, conseguindo, apoiando-se na sua vertente Low Code, com os imensos aceleradores, *templates* e padrões auto-gerados, comprimir as várias fases do ciclo de vida de implementação do software (SDLC) num fluxo único e assistido. Esta grande diferença permite uma implementação no OutSystems Mentor App Generator mais completa e mais abstraída do que no caso do GitHub Copilot. A perceção é no sentido do GitHub Copilot ser um par de programação e o OutSystems Mentor App Generator ser um coautor de alto nível, com um impacto mais direto na arquitetura. Resumindo, o GitHub Copilot é um coautor mais flexível, mais granular e mais dependente da maturidade do programador, enquanto que o OutSystems Mentor App Generator é mais opinativo, mais guiado e mais forte na compressão do SDLC.
Tal como referido nos pontos anteriores deste trabalho quando associados ao caso

de estudo e aos casos práticos, a maturidade da IA no processo de autoria de software ainda não garante resultados 100% confiáveis e que permitam total autonomia.

A Inteligência Artificial sofreu grandes avanços desde meados do século passado, mas o impacto tem sido mais notório na última década. É fácil prever que num futuro próximo, apesar de atualmente a IA não ter uma maturidade suficiente para ser totalmente autónoma tendo em conta cenários de complexidade mais avançados, rapidamente vai conseguir obter essa mesma maturidade. A evolução esperada reforça as competências mais no âmbito de conhecimento e *know-how* da Inteligência Artificial, com forte componente estratégica e de arquitetura global do que propriamente na execução de tarefas repetitivas e escrita de código.

A mudança de paradigma com a IA a assumir papeis de coautoria é tão grande que impacta as equipas de desenvolvimento como um todo e de forma transversal. Essas conclusões conseguem ser comprovadas através da análise feita aos diversos documentos de base deste trabalho, tal como o caso de estudo presente e os casos práticos, sendo que todos demonstram uma redução do tempo de prototipagem, redução do tempo de desenvolvimento, uma reinvenção dos perfis necessários nas equipa de desenvolvimento e uma reconstrução de políticas de segurança, ética, privacidade e *compliance*.

Para Rockall [Roc+25] é possível a uma empresa conseguir aumentar a sua produtividade, sendo que é extremamente relevante para esses valores a maturidade dos processos que adotam a IA e que eles só se materializam se existir formação adequada, uma cultura organizacional incentivada à experimentação, com processo validados continuamente e, acima de tudo, que seja possível escalar tecnicamente com recurso a infraestruturas capazes para tal. Ou seja, não se pode esperar que o facto de existir à disposição tal ferramenta faça com que tudo cresça e melhore. É preciso integrar essas soluções, aprender a trabalhar com elas e potencia-las o máximo possível.

A análise dos ganhos de produtividade associados à adoção de Inteligência Artificial evidencia impactos financeiros positivos, ainda que moderados na fase atual de maturidade tecnológica. De acordo com o AI Index Report 2025 da Stanford University, 49% das organizações que utilizam IA em operações de serviço reportam reduções de custos, enquanto 41% observam poupanças em engenharia de software — embora a maioria destes ganhos se situe abaixo dos 10% [25].

É muito importante compreender que as ferramentas atuais, que se servem da IA como coautora nos processos de desenvolvimento, permitem que pessoas com um menor grau de conhecimento e experiência se aventurem no desenvolvimento de software. Isto articula um exercício bastante curioso, que é o facto de abrir a porta

a que quem desenhe e pense nas soluções as consiga implementar sem ter que recorrer a um programador. Isso tem o nome de *citizen development*.
Além disso, a simplificação do processo de desenvolvimento torna-se motivador para que mais indivíduos entrem no mercado de software, o que permite colmatar uma necessidade emergente de engenheiros de software.
Esta tendência foi observada por Caballar [Cab24], tendo em conta que o mesmo defende que a introdução da IA nos processos complexos do ciclo de vida do desenvolvimento de software torna mais acessível esse mesmo desenvolvimento, passando a ser algo muito mais colaborativo e com foco na intenção do que propriamente na execução técnica, sendo este último uma imagem presente na retina de todos os que eram alheios a este universo quando confrontado com ele.
Com todos estes factos conclui-se que a democratização da IA tem de, obrigatoriamente, ser acompanhada de sustentabilidade, ou seja, têm que existir processos de formação, supervisão e gestão que permitam um crescimento sustentável dos paradigmas e com equipas bem formadas e capazes para funcionarem neste novo contexto.
Tendo em conta todas as conclusões mencionadas até esta parte do trabalho, baseadas em análise de documentação, no caso de estudo e nos casos práticos, pode-se resumir e compactar tudo em cinco pontos chave:

1. Atualmente a IA já é uma coautora bastante efetiva no processo de desenvolvimento de software. Já existe um conjunto alargado de ferramentas utilizadas nas várias fases de desenvolvimento de software que utilizam a IA para mais do que tarefas repetitivas, passando a exercer a capacidade de geração de conteúdo.

2. O engenheiro de software da atualidade representa um conjunto de valências bastante diferentes do engenheiro de software pré-IA. Deixa-se de ter um elemento fundamentalmente técnico no âmbito de programação e passa-se a ter alguém mais orientado para a supervisão comportamental da IA, capaz de interpretar os seus resultados, acompanhando todo o processo de desenvolvimento de software garantindo que o trabalho é feito com a maior qualidade, ética, segurança e performance possíveis.

3. A IA está presente em todas as fases do ciclo de vida de desenvolvimento de software. Desde a componente de análise de requisitos, passando pela programação, testes e metodologias de entrega, já encontramos a IA a efetuar todo o tipo de tarefas e mesmo a tomar decisões.

4. Há consciência que existem grandes desafios nas vertentes éticas e legais, onde se continuam a amadurecer regras e guias de boas práticas, sendo que

o tema ainda é muito recente na história. O objetivo é obter o máximo de transparência e responsabilização.

5. A disciplina da Engenharia de Software passa para um contexto de criação híbrida, ou seja, passa-se a ter a IA a participar imenso na componente de interpretação e geração e passa-se a ter o engenheiro de software numa perspetiva mais de supervisão, compreensão de resultados gerados pela IA e otimização/*tweaking*.

Devem-se ter em conta os pontos enumerados acima para criar a base dos trabalhos futuros, compreendendo que existe um conjunto de oportunidades a investigar e a desenvolver dentro deste novo paradigma de desenvolvimento de software.

6.2 Trabalhos futuros

Neste trabalho conseguem-se identificar pontos e conceitos que merecem foco e atenção de forma a permitir a sua maturação mais acelerada. Com isso, pretendem-se descrever os caminhos possíveis para trabalhos de continuidade e que podem trazer valor para o tema em estudo, sugerindo as seguintes propostas:

- Resolução e clarificação dos problemas éticos e legais gerados pelo posicionamento da IA como coautora nos processos de desenvolvimento de software.
- Formas evolutivas dos modelos híbridos de desenvolvimento de software entre humanos e IA.
- Avaliação da qualidade dos resultados gerados pela IA nos processos de desenvolvimento de software.
- A explicabilidade e transparência dos resultados do trabalho executado pela IA nas diferentes fases de desenvolvimento de software.
- Criação de métricas específicas para avaliar o desempenho da IA nos processos de coautoria.
- O impacto da IA coautora de software na área de formação de engenheiros de software.
- Evolução para autonomia controlada da IA nos processos de desenvolvimento de software.

O fundamento dos trabalhos propostos centra-se na obtenção de um comportamento mais controlado por parte da IA e na obtenção de resultados mais confiáveis e alinhados com as expectativas dos humanos, permitindo assim à IA trabalhar cada vez com mais autonomia.

6.3 Resumo final das conclusões e trabalhos futuros

Todo o trabalho realizado ao longo desta dissertação teve como objetivo primário evidenciar o ponto atual da história e avanço tecnológico no core daquilo que é a Engenharia de Software: o ciclo de vida do desenvolvimento de software (SDLC).
Tornou-se percetível e comprovado que o papel da IA nos processos de desenvolvimento de software é tremendamente disruptivo, assumindo, além da capacidade para tarefas mais básicas, o papel de criação e geração dos mais diferentes entregáveis e resultados esperados, sejam eles documentais ou programação de software. Encontramos a IA a interpretar requisitos aplicacionais em linguagem natural, e, consequentemente, a gerar modelos de dados e arquitetura, código de software, scripts de testes e até preparar o próprio software para ser testado. Já é recorrente encontrar a IA a fazer a manutenção de software já criado e refatorizações necessárias. A IA é, neste momento, realmente e de forma comprovada, uma coautora de software, já que, para uma autoria completa e autónoma, ainda carece de amadurecimento de ferramentas e de implementação de normas, regras e leis que permitam que os resultados por si obtidos não comprometam nenhum vetor que seja aplicado aos processo de desenvolvimento de software realizado apenas por humanos.
Todo o estudo teórico realizado apoiado pela bibliografia partilhada e reforçado pelo caso de estudo do GitHub Copilot, pelo caso prático dessa mesma ferramenta e pelo caso prático da ferramenta OutSystems Mentor App Generator, permitiram chegar a um conjunto de conclusões devidamente suportadas pelo conteúdo deste trabalho e descritas anteriormente neste Capítulo.
Resumindo, não se contempla, pelo menos num futuro próximo, o desaparecimento do engenheiro de software, mas sim uma reconfiguração da personagem, com um novo conjunto de competências e aptidões.
Ao contrário do que intuitivamente possa ser sugerido, não temos uma castração da criatividade humana, mas sim uma amplificação que ainda nem se consegue medir corretamente.
Apesar da grande capacidade de decisão que os sistemas inteligentes artificiais possuem, não se pode menosprezar a sua supervisão.
A disciplina de Engenharia de Software dificilmente irá retroceder e voltar a ser novamente um trabalho apenas de humanos. Estamos perante uma fase de revolução

digital de grande escala, sendo que a IA surge logo no início da cadeia, presente, não apenas como uma ajudante para tarefas secundárias, mas sim como coautora de diferentes conteúdos e resultados que levam à entrega de soluções digitais de qualidade e para o futuro.

No decorrer desta investigação, a Inteligência Artificial foi utilizada exclusivamente como auxiliar de pesquisa e confrontação de dados, permitindo acelerar a recolha de informação, comparar fontes distintas e validar coerência conceptual entre diferentes secções do trabalho. Todas as decisões analíticas, interpretações e conclusões apresentadas permanecem integralmente da responsabilidade do autor, garantindo que a IA desempenhou apenas um papel instrumental de apoio metodológico, sem interferir na autonomia científica nem na integridade crítica da dissertação.

Bibliografia

[25] AI Index Report 2025. Economy and Productivity Chapter. Stanford Human-Centered Artificial Intelligence, 2025. https://aiindex.stanford.edu.

[Akh+24] M. A. K. Akhtar et al.. “Transparency and Accountability in Explainable AI: Best Practices”. *Springer Nature Link* , 2024.

[AA25] M. Alenezi e M. Akour. “AI-Driven Innovations in Software Engineering: A Review of Current Practices and Future Directions”. *MDPI* , 2025.

[Alt23] Sam Altman. Lex Fridman Podcast 419: OpenAI, GPT-5, Sora, Board Saga, Elon Musk, Ilya, Power and AGI. https://lexfridman.com/sam-altman-2/.

[AJ25] G. Amasanti e J. Jasmin. “The Impact of AI-Generated Solutions on Software Architecture and Productivity: Results from a Survey Study”. *arXiv* , 2025.

[Ama26] Amazon. CodeWhisperer is becoming a part of Amazon Q Developer. https://docs.aws.amazon.com/codewhisperer/latest/userguide/whisper-legacy.html.

[Ame+19] S. Amershi et al.. “Software Engineering for Machine Learning: A Case Study”. *IEEE Xplore* , 2019.

[Ant25] Anthropic. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5.

[Ays+25] H. I. Aysel et al.. “Explainable Artificial Intelligence: Advancements and limitations”. *MDPI* , 2025.

[Beg+25] A. Beg et al.. “Leveraging LLMs for Formal Software Requirements – Challenges and Prospects”. *arXiv* , 2025.

[Bri25] Mozart T. Brito. “Abordagens ao desenvolvimento de software usando ferramentas de inteligência artificial”. Accessed on 2026-02-28. Tese de mestrado. Instituto Politécnico de Viana do Castelo, 2025. http://repositorio.ipvc.pt/bitstream/20.500.11960/4477/1/Mozart_Brito.pdf.

[Cab24] R. D. Caballar. “AI copilots are changing how coding is taught”. *IEEE Spectrum* , 2024.

[Cam+23] A. Cambon et al.. “AI and Productivity Report – First Edition”. *Microsoft Research* , 2023.

[Che26] Checkmarx. AI Secure Coding Assistant (ASCA) for Visual Studio. https://docs.checkmarx.com/en/34965-314153-ai-secure-coding-assistant--asca--for-visual-studio.html.

[Che+21] M. Chen et al.. “Evaluating Large Language Models Trained on Code”. *arXiv* , 2021.

[Cop26] B. J. Copeland. “Artificial Inteligence”. *Britannica* , 2026.

[Cou+24] M. Coutinho et al.. “The Role of Generative AI in Software Development Productivity: A Pilot Case Study”. *arXiv* , 2024.

[Dat26] Datadog. Internal Developer Portal - Ship quickly and confidently with developer self-service, delivery guardrails, and live system data. https://www.datadoghq.com/product/internal-developer-portal/.

[Des23] Web Desk. Software Development Ethics: AI Bias, Privacy Concerns, and Responsible Coding. https://www.digitalinformationworld.com/2023/10/software-development-ethics-ai-bias.html.

[Dif25] Diffblue. Discover Diffblue Cover. https://docs.diffblue.com/.

[Doc24] Cursor Docs. Agent Modes. https://cursor.com/docs.

[Far24] S. Farmer. Upskilling engineering teams for the AI era. https://thenewstack.io/upskilling-engineering-teams-for-the-ai-era/.

[Fig26] Figma. Figma AI é o seu colaborador criativo. https://www.figma.com/pt-br/ai/.

[For23] Forbes. A história do jovem que construiu o unicórnio da Outsystems. https://www.forbespt.com/a-historia-do-jovem-que-construiu-o-unicornio-da-outsystems/.

[Git24] GitHub. GitHub Copilot X: The AI-powered developer experience. https://github.blog/news-insights/product-news/github-copilot-x-the-ai-powered-developer-experience/.

[Git26a] GitHub. AI GitHub Action - Supercharge your GitHub workflow with AI-powered automation. https://github.com/marketplace/actions/ai-github-action.

[Git26b] GitHub. O que é o GitHub Copilot? Saiba o que é Copilot e o que você pode fazer com isso. https://docs.github.com/pt/copilot/get-started/what-is-github-copilot.

[Git26c] GitHub. Triaging an issue with AI. https://docs.github.com/en/issues/tracking-your-work-with-issues/administering-issues/triaging-an-issue-with-ai.

[Has23] Demis Hassabis. Lex Fridman Podcast 475: Future of AI, Simulating Reality, Physics and Video Games. https://lexfridman.com/demis-hassabis-2/.

[He+25a] J. He et al.. "Generative artificial intelligence: a historical perspective". *National Science Review* , 2025.

[He+25b] J. He et al.. "LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead". *arXiv* , 2025.

[Hem+25] M. Hemmat et al.. "Research directions for using LLM in software requirement engineering: a systematic review". *Frontiers in Computer Science* , 2025.

[Hig23] Jim Highsmith. AI in Software Development Lifecycle (SDLC). https://jimhighsmith.com/technical-debt-is-rust-you-cant-see/.

[Ide24] Ideas2IT. AI in Software Development Lifecycle (SDLC). https://www.ideas2it.com/blogs/ai-in-software-development-sdlc.

[Inf22] InfoWork. Low-code development technologies market forecast to hit 44.5 billion dollars by 2026. https://www.infoworld.com/article/2337677/low-code-development-technologies-market-forecast-to-hit-445-billion-by-2026.html.

[Kal24] E. Kalliamvakou. "Research: quantifying GitHub Copilot's impact on developer productivity and happiness". *GitHub Blog* , 2024.

[Lwa+19] L. E. Lwakatare et al.. "A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation". *Research Gate* , 2019.

[Man25] Manube. Citizen Development - What is a Citizen Developer?. https://www.manubes.com/what-is-a-citizen-developer/.

[Mer26] Mermaid. About Mermaid. https://mermaid.ai/open-source/intro/.

[MET25] METR. Measuring AI ability to complete long tasks. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/.

[Mic26a] Microsoft. Enable AI assistance with Azure DevOps MCP Server. https://learn.microsoft.com/en-us/azure/devops/mcp-server/mcp-server-overview?view=azure-devops.

[Mic26b] Microsoft. GitHub Advanced Security. https://azure.microsoft.com/pt-pt/products/github/Advanced-Security.

[Min26] Mintlify. Mintlify - Get started. https://www.mintlify.com/docs/quickstart.

[Ner26] All Tech Nerd. What is GPT-5.3-Codex? OpenAI's new coding model explained. https://www.alltechnerd.com/what-is-gpt-5-3-codex/.

[Nex23] GitHub Next. Copilot for Pull Requests. https://githubnext.com/projects/copilot-for-pull-requests.

[Nob18] S. U. Noble. Algorithms of Oppression. https://nyupress.org/9781479837243/algorithms-of-oppression/.

[Ope21] OpenAI. OpenAI Codex. https://openai.com/codex.

[Ope26] OpenAI. Agents - Learn how to build, deploy, and optimize agent workflows with AgentKit. https://developers.openai.com/api/docs/guides/agents/.

[Out25a] OutSystems. Gartner Magic Quadrant 2025: OutSystems recognized as a 9X Leader. https://www.outsystems.com/1/low-code-application-platforms-gartner-/.

[Out25b] OutSystems. How Mentor works. https://success.outsystems.com/documentation/outsystems_developer_cloud/building_apps/build_apps_with_ai/how_mentor_works/.

[Out25c] OutSystems. Mentor App Generator and App Editor: Setting a new standard in app development with AI. https://www.outsystems.com/product-updates/mentor-ga/.

[Out26] OutSystems. Prompting guide. https://success.outsystems.com/documentation/outsystems_developer_cloud/building_apps/build_apps_with_ai/prompting_guide/.

[Par23] European Parliament. EU AI Act: First regulation on artificial intelligence. https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence.

[Pen+23] H. Peng et al.. "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot". *arXiv* , 2023.

[Rao+25] Jagannadha D. B. Rao et al.. “A Comparative Analysis of Software Development Models: Waterfall, Agile and DevOps”. *Springer Nature Link* , 2025.

[Rao25] Jagannadha D. B. Rao. “Agile System Development Lifecycle for AI Systems: Decision Architecture”. *arXiv* , 2025.

[Rec+24] G. Recupito et al.. “Technical Debt in AI-Enabled Systems: On the Prevalence, Severity, Impact, and Management Strategies for Code and Architecture”. *GitHub Pages* , 2024.

[Rib23] Marcos de Souza Ribeiro. “Inteligência artificial e suas contribuições para a área: Análise de aplicações no desenvolvimento de sistemas”. *Revista Multivisões AESA*, 1 :pp. 1–15 , 2023.

[Roc+25] E. J. Rockall et al.. “AI Adoption and Inequality”. *International Monetary Fund* , 2025.

[Rus+21] S. Russell et al.. Artificial Intelligence: A Modern Approach. Berkeley, 2021.

[Sil21] R. J. Silva. “A inteligência artificial no contexto da ciência da informação: Uma análise de domínio”. Accessed on 2026-02-28. Tese de mestrado. Universidade do Porto, 2021. https://repositorio-aberto.up.pt/bitstream/10216/135714/2/488235.pdf.

[Sin23] K. Sindhya. Unlocking the power of ChatGPT for rapid requirements extraction. https://www.solita.fi/blogs/unlocking-the-power-of-chatgpt-for-rapid-requirements-extraction/.

[Sny26] Snyk. What’s Snyk?. https://docs.snyk.io/discover-snyk/whats-snyk.

[Som16] I. Sommerville. Software Engineering. Pearson, 2016.

[Sou26] Sourcegraph. Cody. https://sourcegraph.com/docs/cody.

[Stu26] Visual Studio. Using agents in Visual Studio Code. https://code.visualstudio.com/docs/copilot/agents/overview.

[Swi26] Swimm. Swimm overview — Benefits of code-coupled documentation with Swimm. https://docs.swimm.io/new-to-swimm/benefits-of-code-coupled-documentation/.

[Tab26] Tabnine. Overview — What is Tabnine?. https://docs.tabnine.com/main.

[Tes25] Testim. Testim overview. https://help.testim.io/docs/testim-overview.

[Tiw25] A. Tiwari. Who Is Responsible When AI Makes a Mistake?. https://www.riskinfo.ai/post/who-is-responsible-when-ai-makes-a-mistake.

[Uni16] European Union. Regulation (EU) 2016/679 - General Data Protection Regulation (GDPR). https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[Vai22] P. Vaithilingam. “Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models”. *ACM* , 2022.

[Wir23] Business Wire. OutSystems Announces New Series of Low-Code Development Schools to Meet the Growing Demand for Developers Worldwide. https://www.businesswire.com/news/home/20230821177048/en/OutSystems-Announces-New-Series-of-Low-Code-Development-Schools-to-Meet-the-Growing-Demand-for-Developers-Worldwide.

Anexo A

Dados Complementares

A.1 Código gerado pelo GitHub Copilot para a aplicação do respetivo caso prático

A.1.1 Código-Fonte do backend (ficheiro *Program.cs*) da aplicação

```
1
2   using CasoPaticoGHCP.Data;
3   using CasoPaticoGHCP.Models;
4   using Microsoft.EntityFrameworkCore;
5
6   // Configure a basic Web API for managing requests WITH CRUD
       endpoints
7   var builder = WebApplication.CreateBuilder(args);
8
9   // Add services to the container
10  builder.Services.AddDbContext<RequestDbContext>(options =>
11  options.UseSqlite("Data Source=requests.db"));
12
13  builder.Services.AddCors(options =>
14  {
15    options.AddPolicy("AllowAll", policy =>
16    {
17      policy.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
18    });
19  });
20
```

```
21  builder.Services.AddControllers();
22
23  var app = builder.Build();
24
25  // Apply migrations and create database if needed
26  using (var scope = app.Services.CreateScope())
27  {
28      var db = scope.ServiceProvider.GetRequiredService<
            RequestDbContext>();
29      db.Database.EnsureCreated();
30  }
31
32  app.UseRouting();
33  app.UseCors("AllowAll");
34
35  app.MapControllers();
36
37  app.Run();
```

A.1.2 Código-Fonte do backend (ficheiro *Request.cs*) da aplicação

```
1
2   namespace CasoPaticoGHCP.Models;
3
4   public class Request
5   {
6       public int Id { get; set; }
7       public string Titulo { get; set; } = string.Empty;
8       public string Descricao { get; set; } = string.Empty;
9       public string Estado { get; set; } = "Pendente"; //
            Pendente, Em Progresso, Concluído
10      public DateTime DataCriacao { get; set; } = DateTime.
            UtcNow;
11      public DateTime? DataAtualizacao { get; set; }
12      public string CriadoPor { get; set; } = string.Empty;
13  }
```

A.1.3 Código-Fonte do backend (ficheiro *RequestDBContext.cs*) da aplicação

```
1
2   using CasoPaticoGHCP.Models;
3   using Microsoft.EntityFrameworkCore;
4
5   namespace CasoPaticoGHCP.Data;
6
7   public class RequestDbContext : DbContext
8   {
9     public DbSet<Request> Requests { get; set; }
10
11    public RequestDbContext(DbContextOptions<RequestDbContext>
        options) : base(options)
12    {
13    }
14
15    protected override void OnModelCreating(ModelBuilder
        modelBuilder)
16    {
17      base.OnModelCreating(modelBuilder);
18
19      modelBuilder.Entity<Request>(entity =>
20      {
21        entity.HasKey(e => e.Id);
22        entity.Property(e => e.Titulo).IsRequired().HasMaxLength
            (255);
23        entity.Property(e => e.Descricao).HasMaxLength(1000);
24        entity.Property(e => e.Estado).HasMaxLength(50);
25        entity.Property(e => e.CriadoPor).IsRequired().
            HasMaxLength(255);
26      });
27    }
28  }
```

A.1.4 Código-Fonte do backend (ficheiro *RequestsController.cs*) da aplicação

```
1
2       using CasoPaticoGHCP.Data;
3       using CasoPaticoGHCP.Models;
4       using Microsoft.AspNetCore.Mvc;
5       using Microsoft.EntityFrameworkCore;
6
7       namespace CasoPaticoGHCP.Controllers;
8
9       [ApiController]
10      [Route("api/[controller]")]
11      public class RequestsController : ControllerBase
12      {
13        private readonly RequestDbContext _db;
14
15        public RequestsController(RequestDbContext db) => _db = db
              ;
16
17        [HttpGet]
18        public async Task<ActionResult<IEnumerable<Request>>>
              GetAll()
19        {
20          var requests = await _db.Requests.ToListAsync();
21          return Ok(requests);
22        }
23
24        [HttpGet("{id:int}")]
25        public async Task<ActionResult<Request>> Get(int id)
26        {
27          var request = await _db.Requests.FindAsync(id);
28          if (request is null)
29          return NotFound(new { message = $"Pedido com ID {id} não
                encontrado" });
30          return Ok(request);
31        }
32
33        [HttpPost]
34        public async Task<ActionResult<Request>> Create(
              CreateRequestDto dto)
35        {
36          if (string.IsNullOrWhiteSpace(dto.Titulo))
37          return BadRequest(new { message = "Titulo é obrigatório"
```

```
            });
38
39          if (string.IsNullOrWhiteSpace(dto.CriadoPor))
40          return BadRequest(new { message = "CriadoPor é obrigatório" });
41
42          var request = new Request
43          {
44            Titulo = dto.Titulo,
45            Descricao = dto.Descricao ?? string.Empty,
46            Estado = "Pendente",
47            DataCriacao = DateTime.UtcNow,
48            CriadoPor = dto.CriadoPor
49          };
50
51          _db.Requests.Add(request);
52          await _db.SaveChangesAsync();
53
54          return CreatedAtAction(nameof(Get), new { id = request.Id }, request);
55        }
56
57        [HttpPut("{id:int}")]
58        public async Task<ActionResult<Request>> Update(int id, UpdateRequestDto dto)
59        {
60          var request = await _db.Requests.FindAsync(id);
61          if (request is null)
62          return NotFound(new { message = $"Pedido com ID {id} não encontrado" });
63
64          if (!string.IsNullOrWhiteSpace(dto.Titulo))
65          request.Titulo = dto.Titulo;
66
67          if (dto.Descricao is not null)
68          request.Descricao = dto.Descricao;
69
70          if (!string.IsNullOrWhiteSpace(dto.Estado))
71          request.Estado = dto.Estado;
72
73          request.DataAtualizacao = DateTime.UtcNow;
```

```
74
75          await _db.SaveChangesAsync();
76
77          return Ok(request);
78        }
79
80        [HttpDelete("{id:int}")]
81        public async Task<IActionResult> Delete(int id)
82        {
83          var request = await _db.Requests.FindAsync(id);
84          if (request is null)
85          return NotFound(new { message = $"Pedido com ID {id} não
                encontrado" });
86
87          _db.Requests.Remove(request);
88          await _db.SaveChangesAsync();
89
90          return NoContent();
91        }
92
93        [HttpPatch("{id}/status")]
94        public async Task<IActionResult> UpdateStatus(int id, [
             FromBody] string newStatus)
95        {
96          if (string.IsNullOrWhiteSpace(newStatus))
97          return BadRequest(new { message = "Novo estado é obrigat
              ório" });
98
99          var validStatuses = new[] { "Pendente", "Em Progresso",
              "Concluído" };
100         if (!validStatuses.Contains(newStatus))
101         return BadRequest(new { message = "Estado inválido.
              Estados válidos: Pendente, Em Progresso, Concluído" })
              ;
102
103         var request = await _db.Requests.FindAsync(id);
104         if (request is null)
105         return NotFound(new { message = $"Pedido com ID {id} não
                encontrado" });
106
107         request.Estado = newStatus;
```

```
108         request.DataAtualizacao = DateTime.UtcNow;
109
110         await _db.SaveChangesAsync();
111
112         return NoContent();
113       }
114     }
```

A.1.5 Código-Fonte do frontend (ficheiro *App.jsx*) da aplicação

```
1     import { useEffect, useState } from 'react'
2     import axios from 'axios'
3     import './App.css'
4
5     const API_BASE = import.meta.env.VITE_API_BASE_URL || 'http
          ://localhost:5000/api/requests'
6
7     function PedidoForm({ initialData, onSubmit, onCancel }) {
8       const [pedido, setPedido] = useState(
9       initialData || { titulo: '', descricao: '', criadoPor: ''
            }
10      )
11
12      useEffect(() => {
13        if (initialData) setPedido(initialData)
14      }, [initialData])
15
16      const handleChange = (e) => {
17        const { name, value } = e.target
18        setPedido((prev) => ({ ...prev, [name]: value }))
19      }
20
21      const handleSubmit = (e) => {
22        e.preventDefault()
23        if (!pedido.titulo.trim() || !pedido.criadoPor.trim())
              return
24        onSubmit(pedido)
25        setPedido({ titulo: '', descricao: '', criadoPor: '' })
26      }
27
```

```
28        return (
29        <form className="pedido-form" onSubmit={handleSubmit}>
30        <h2>{initialData ? 'Atualizar Pedido' : 'Criar Pedido'}</
             h2>
31        <label>
32        Título
33        <input name="titulo" value={pedido.titulo} onChange={
             handleChange} required />
34        </label>
35        <label>
36        Descrição
37        <textarea name="descricao" value={pedido.descricao}
             onChange={handleChange} />
38        </label>
39        <label>
40        Criado por
41        <input name="criadoPor" value={pedido.criadoPor} onChange
             ={handleChange} required />
42        </label>
43        <div className="actions">
44        <button type="submit">{initialData ? 'Salvar' : 'Criar'}</
             button>
45        {initialData && (
46          <button type="button" className="cancel" onClick={
               onCancel}>
47          Cancelar
48          </button>
49          )}
50        </div>
51        </form>
52        )
53      }
54
55      function PedidoRow({ pedido, onEdit, onDelete }) {
56        return (
57        <tr>
58        <td>{pedido.id}</td>
59        <td>{pedido.titulo}</td>
60        <td>{pedido.descricao}</td>
61        <td>{pedido.criadoPor}</td>
62        <td>{pedido.estado || 'Pendente'}</td>
```

```
63          <td>
64          <button onClick={() => onEdit(pedido)}>Editar</button>
65          <button onClick={() => onDelete(pedido)} className="danger
                ">
66          Excluir
67          </button>
68          </td>
69          </tr>
70          )
71        }
72
73        function App() {
74          const [pedidos, setPedidos] = useState([])
75          const [loading, setLoading] = useState(true)
76          const [error, setError] = useState('')
77          const [editing, setEditing] = useState(null)
78
79          const loadPedidos = async () => {
80            setLoading(true)
81            setError('')
82            try {
83              const { data } = await axios.get(API_BASE)
84              setPedidos(Array.isArray(data) ? data : [])
85            } catch (err) {
86              setError('Falha ao carregar pedidos. Verifique API.')
87            } finally {
88              setLoading(false)
89            }
90          }
91
92          useEffect(() => {
93            void loadPedidos()
94          }, [])
95
96          const handleCreate = async (payload) => {
97            setError('')
98            try {
99              await axios.post(API_BASE, { ...payload, estado: '
                    Pendente' })
100             setEditing(null)
101             await loadPedidos()
```

```
102         } catch (err) {
103           setError('Falha ao criar pedido.')
104         }
105       }
106
107       const handleUpdate = async (payload) => {
108         if (!payload.id) {
109           setError('ID do pedido ausente para atualização.')
110           return
111         }
112         setError('')
113         try {
114           await axios.put(`${API_BASE}/${payload.id}`, payload)
115           setEditing(null)
116           await loadPedidos()
117         } catch (err) {
118           setError('Falha ao atualizar pedido.')
119         }
120       }
121
122       const handleDelete = async (pedido) => {
123         if (!pedido.id) return
124         setError('')
125         try {
126           await axios.delete(`${API_BASE}/${pedido.id}`)
127           await loadPedidos()
128         } catch (err) {
129           setError('Falha ao excluir pedido.')
130         }
131       }
132
133       if (loading) return <div className="container">Carregando
              pedidos...</div>
134
135       return (
136       <div className="container">
137       <h1>Gestão de Pedidos</h1>
138       <p>API Base: <code>{API_BASE}</code></p>
139
140       {error && <div className="error">{error}</div>}
141
```

```
142         <PedidoForm
143         initialData={editing}
144         onSubmit={editing ? handleUpdate : handleCreate}
145         onCancel={() => setEditing(null)}
146         />
147
148         <section className="table-wrap">
149         <h2>Listagem de Pedidos</h2>
150         {pedidos.length === 0 ? (
151           <p>Nenhum pedido encontrado.</p>
152           ) : (
153           <table>
154           <thead>
155           <tr>
156           <th>ID</th>
157           <th>Título</th>
158           <th>Descrição</th>
159           <th>Criado por</th>
160           <th>Estado</th>
161           <th>Ações</th>
162           </tr>
163           </thead>
164           <tbody>
165           {pedidos.map((pedido) => (
166             <PedidoRow
167             key={pedido.id ?? pedido.requestId ?? pedido.pedidoId}
168             pedido={pedido}
169             onEdit={setEditing}
170             onDelete={handleDelete}
171             />
172             ))}
173           </tbody>
174           </table>
175           )}
176         </section>
177         </div>
178         )
179       }
180
181       export default App
```

A.2 Estimativas de desenvolvimento dos casos práticos sem recurso a IA como coautora

A.2.1 Estimativa de programador pleno ASP.NET + React para desenvolvimento da aplicação do caso prático Github Copilot sem recurso a este

Tabela A.1: Estimativa de esforço para implementação sem Copilot — Developer Pleno

Atividade	**Esforço (horas)**
Setup solução backend	4
Modelo de domínio	4
Configuração EF Core	5
Migrações e seed	4
API CRUD de pedidos	8
Endpoint de atualização de estado	4
Tratamento de erros	4
CORS e configuração API	2
Setup app React	4
Integração com API	5
Componente de listagem	6
Componente de formulário	5
Gestão de estado	5
Filtro/pesquisa (opcional)	4
Estilização básica	4
Testes manuais end-to-end	4
Refinamentos e correções	4
Documentação técnica	5
Total estimado	**81 horas**

A.2.2 Estimativa de programador sénior ASP.NET + React para desenvolvimento da aplicação do caso prático Github Copilot sem recurso a este

Tabela A.2: Estimativa de esforço para implementação sem Copilot — Developer Sénior

Atividade	**Esforço (horas)**
Setup solução backend	3
Modelo de domínio	3
Configuração EF Core	4
Migrações e seed	3
API CRUD de pedidos	6
Endpoint de atualização de estado	3
Tratamento de erros	3
CORS e configuração API	1
Setup app React	3
Integração com API	4
Componente de listagem	4
Componente de formulário	4
Gestão de estado	4
Filtro/pesquisa (opcional)	3
Estilização básica	3
Testes manuais end-to-end	3
Refinamentos e correções	3
Documentação técnica	4
Total estimado	**62 horas**

A.2.3 Estimativa de programador pleno OutSystems para desenvolvimento da aplicação do caso prático sem recurso ao Mentor App Generator (sem IA)

Tabela A.3: Estimativa de esforço manual para um programador Pleno OutSystems (sem IA)

Atividade	**Esforço (horas)**
Modelação de dados	14
Configuração de roles e autenticação	8
Ecrãs CRUD completos	26
Dashboard + KPIs + gráficos	14
Pesquisa avançada e filtros	8
Upload/download de ficheiros	6
Notificações por email	6
Knowledge Base	12
API pública (REST)	8
Internacionalização (i18n)	6
Testes funcionais + correções	16
Hardening (GDPR, validações, segurança)	10
Total estimado	**134 horas**

A.2.4 Estimativa de programador sénior OutSystems para desenvolvimento da aplicação do caso prático sem recurso ao Mentor App Generator (sem IA)

Tabela A.4: Estimativa de esforço manual para um programador Sénior OutSystems (sem IA)

Atividade	**Esforço (horas)**
Modelação de dados	10
Configuração de roles, permissões e autenticação	6
Implementação dos ecrãs CRUD	18
Dashboard + KPIs + gráficos	10
Pesquisa avançada e filtros	6
Upload/download de ficheiros	4
Notificações por email	4
Knowledge Base (CRUD + UI + ligação a tickets)	8
API pública (REST) para extensibilidade	6
Internacionalização (i18n)	4
Testes funcionais + correções	10
Hardening (GDPR, validações, segurança)	6
Total estimado	**92 horas**