Instituto Superior
de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA E DE SISTEMAS

Reestruturação Pedagógica de Gestão de Projeto de Software: Substituição de Metodologia Tradicional por Ágil

Trabalho de Projeto para a obtenção do grau de Mestre em Engenharia Informática

Especialização em Desenvolvimento de Software

Autor

Mickaël Fonseca dos Santos

Orientadores

Professor Doutor João Carlos Costa Faria da Cunha

Professor Doutor Ricardo Ângelo Santos Filipe

Coimbra, outubro de 2024

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

RESUMO

Atualmente, as metodologias ágeis são uma tendência, sendo adotadas cada vez mais pela indústria. Desta forma, este trabalho teve como objetivo a reestruturação da Unidade Curricular de Gestão de Projetos de Software da Licenciatura em Engenharia Informática do Instituto Superior de Engenharia de Coimbra, substituindo a metodologia de ensino tradicional por uma abordagem baseada em metodologias ágeis. Para fundamentar esta reformulação, realizou-se uma análise das boas práticas da engenharia de software e das metodologias de desenvolvimento de software existentes. Foi realizado um trabalho de campo, que incluiu entrevistas a empresas influentes a nível regional, nacional e internacional, assim como a outras instituições de ensino superior, com o propósito de identificar metodologias e práticas utilizadas no ensino de gestão de projetos de software. Tendo em conta que este trabalho teve a duração de dois anos letivos, no primeiro ano letivo, a unidade curricular foi analisada internamente, de modo a avaliar tanto os processos como as práticas já em utilização, identificando os aspetos eficazes e as melhorias e ajustes necessários. No segundo ano letivo, após o planeamento da reformulação da Unidade Curricular, os alunos implementaram a metodologia Scrum, juntamente com práticas e processos derivados de DevOps no desenvolvimento de software. Os resultados superaram as expectativas, demonstrando um aumento significativo nas competências adquiridas pelos alunos do segundo ano letivo na gestão de projetos de software, apesar das previsões iniciais favorecerem os alunos do primeiro ano, que apresentavam maior conhecimento prévio nas áreas, tecnologias e processos da unidade curricular, conforme evidenciado por um formulário aplicado no início de cada ano letivo.

Palavras-chave: Metodologias de desenvolvimento de software, metodologias ágeis, gestão de projetos de software, engenharia de software

ABSTRACT

Agile methodologies are currently a trend and are increasingly being adopted by industry. Therefore, the aim of this work was to restructure the Software Project Management course unit of the Computer Engineering degree programme at the Coimbra Institute of Engineering, replacing teaching the traditional methodology with an approach based on agile methodologies. To support this reformulation, an analysis of good software engineering practices and existing software development methodologies was carried out. Fieldwork was carried out, including interviews with influential companies at regional, national and international level, as well as other higher education institutions, with the aim of identifying methodologies and practices used in teaching software project management. Given that this work took two school years, in the first academic year, the curricular unit was analysed internally to evaluate both the processes and the practices already in use, identifying the effective aspects and identifying the necessary improvements and adjustments. In the second academic year, after planning the reformulation of the curricular unit, the students implemented the Scrum methodology, along with practices and processes derived from DevOps in software development. The results exceeded expectations, demonstrating a significant increase in the skills acquired by second-year students in software project management, despite initial predictions favouring first-year students, who had greater prior knowledge of the areas, technologies and processes of the course unit, as evidenced by a form applied at the start of each academic year.

Keywords: Software development methodologies, agile methodologies, software project management, software engineering

AGRADECIMENTOS

Finalizada uma etapa importante da minha vida, não poderia deixar de agradecer a todos aqueles que me apoiaram nesta longa caminhada e contribuíram para a realização deste Projeto.

Gostaria, em primeiro lugar, de agradecer aos meus orientadores, Professor Doutor João Carlos Costa Faria da Cunha e Professor Doutor Ricardo Ângelo Santos Filipe, pelo apoio constante, orientação incansável e pela partilha de conhecimentos que foram essenciais para o desenvolvimento deste projeto. A confiança de ambos no meu trabalho e a motivação que sempre transmitiram foram fundamentais para superar os desafios deste percurso.

Expresso o meu agradecimento para com todos os meus colegas do curso de Engenharia Informática, que me ajudaram ao longo deste trabalho, pela ajuda e apoio prestado. A todos os meus professores pelo conhecimento que me proporcionaram ao longo desta longa caminhada.

Por fim, mas não menos importante, à minha família, namorada e amigos, pela motivação transmitida ao longo destes anos, mesmo quando não estava motivado e estiveram sempre presentes para ultrapassar os maus momentos. Sem eles não teria sido possível chegar tão longe na minha vida académica.

A todos o meu sincero muito obrigado!

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIGLAS E ACRÓNIMOS

ACM	Association for Computing Machinery
BDD	Behavior Driven Development
CD	Continuous Delivery
CI	Continuous Integration
CRC Cards	Use Class, Responsibilities and Collaboration Cards
ES	Engenharia de Software
ESTG	Escola Superior de Tecnologia e Gestão
EVA	Earned Value Analysis
FCTUC	Faculdade de Ciências e Tecnologias da Universidade de Coimbra
FCUL	Faculdade de Ciências da Universidade de Lisboa
FEUP	Faculdade de Engenharia da Universidade do Porto
GPS	Gestão de Projetos de Software
IEEE – CS	IEEE Computer Society
IES	Instituições de Ensino Superior
ISEC	Instituto Superior de Engenharia de Coimbra
IST	Instituto Superior Técnico
LEI	Licenciatura em Engenharia Informática
MR	Merge Request
MVP	Minimum Viable Product
OA	Objetivo de Aprendizagem
PB	Product Backlog
PMI	Project Management Institute
PSI	Projeto de Sistemas de Informação
QAP	Quality Assurance Plan
RA	Resultado de Aprendizagem
RUP	Rational Unified Process
SB	Sprint Boards
SDLC	Software Development Life Cycle
SRD	Software Requirements Document
TAES	Tópicos Avançados de Engenharia de Software
UC	Unidade Curricular
US	User Story
WBS	Work Breakdown Structure
WiP	Work in Progress
XP	Extreme Programming

1 INTRODUÇÃO

1.1 Âmbito e motivação

O ensino da gestão de projetos é uma pedra basilar para o ramo de engenharia de software. No Instituto Superior de Engenharia de Coimbra (ISEC), essa componente é parte integrante da unidade curricular (UC) de Gestão de Projetos de Software (GPS) da Licenciatura em Engenharia Informática (LEI). A UC de GPS tem como objetivo envolver os estudantes em todas as fases de desenvolvimento de um projeto de software, desde a definição dos requisitos, arquitetura, desenvolvimento e testes unitários, testes de aceitação e terminando na sua entrega. Por uma opção pedagógica, segue-se uma metodologia tradicional baseada no modelo em cascata (Waterfall). O uso da mesma deve-se ao seu caráter prescritivo e aos processos bem definidos, sendo mais adequada a equipas inexperientes, quando comparadas com as metodologias ágeis. No entanto, a rigidez no planeamento e no acompanhamento do projeto [1] e a dificuldade de estas se adaptarem às rápidas mudanças nos mercados e às necessidades dos clientes [2] são considerados um problema. Por essa razão, as metodologias ágeis têm sido uma tendência em ascensão nos últimos vinte anos. De acordo com o State of Agile [3], as organizações aumentaram a adoção do Scrum de 57% para 87%, de 2019 a 2022. Além disso, em 2020, o Scrum passou por pequenas mudanças nos seus processos e práticas, demonstrando ser uma metodologia amplamente aceite pela indústria e em constante evolução [4].

1.2 Objetivos e abordagem

Para ir ao encontro das expectativas da indústria relativamente às competências dos licenciados em Engenharia Informática, decidiu-se fazer uma transição para o ensino de metodologias de desenvolvimento de software ágeis. Assim, pretende-se definir um conjunto de práticas, ferramentas e princípios que ofereçam aos estudantes da UC uma experiência próxima das metodologias ágeis, incluindo, se possível, algumas práticas e processos de DevOps[1].

No entanto, devem manter-se os resultados de aprendizagem apesar da mudança de processos, pois pretende-se que os estudantes continuem a adquirir competências centradas na gestão de projetos de software. Assim, com o objetivo de dotar os estudantes com conhecimento e prática de metodologias ágeis de desenvolvimento de software, foram seguidos os seguintes passos:

---

[1] DevOps – É um conjunto de práticas e princípios que permitem um trabalho em equipa entre as equipas de desenvolvimento e as equipas de operações.

- Analisar e avaliar os resultados de aprendizagem e processos utilizados anteriormente na UC de GPS;
- Examinar metodologias e práticas de desenvolvimento de software utilizadas na indústria, incluindo algumas empresas influentes, tanto a nível regional, como nacional;
- Investigar como é lecionada a gestão de projetos de software por outras instituições de ensino superior (IES);
- Analisar e propor novos objetivos de aprendizagem para a UC, baseados em práticas ágeis, ficando sujeitos à aprovação do responsável da UC;
- Desenhar novos processos para o desenvolvimento do trabalho prático de GPS;
- Avaliar os resultados de aprendizagem pelos estudantes de GPS, após adoção de metodologias ágeis.

Resumindo, este trabalho teve a duração de dois anos letivos, onde no primeiro ano foi feita uma análise aos processos e práticas utilizadas pelos estudantes e no segundo ano foi feita uma transição para metodologias ágeis. No final, foram avaliados os resultados desta transição, pela análise das competências adquiridas pelos estudantes na área de gestão de projetos de software.

1.3 Unidade curricular de Gestão de Projetos de Software

A UC de GPS é uma unidade curricular que está presente no final da licenciatura em Engenharia Informática. É de seguida apresentado de forma sucinta o funcionamento da unidade curricular, no entanto, no capítulo 4 será detalhado todo o funcionamento da mesma.

1.3.1 Objetivos genéricos

Ao longo do semestre, os estudantes têm aulas teóricas e práticas. Nas aulas teóricas são transmitidos aos estudantes conceitos e conhecimentos relativos a boas práticas de engenharia, gestão de projetos de software e processos e metodologias de desenvolvimento de software.

Na componente prática, os estudantes desenvolvem um projeto de software em equipa (grupos de cinco), seguindo uma metodologia de desenvolvimento de software. Inicialmente, os estudantes seguiam uma metodologia tradicional. No entanto, pretendeu fazer-se uma transição para uma metodologia ágil.

O objetivo desta UC é dotar os estudantes da capacidade de planear, executar e controlar um projeto de software simples, utilizando práticas de engenharia adequadas e entregá-lo dentro dos custos, prazos, qualidade e âmbito expectáveis. Este projeto tem a duração de todo o semestre. Os estudantes devem fazer um

esforço semanal de quatro horas por cada membro do grupo. Os estudantes registam os seus esforços semanais individuais e coletivos para acompanhar a evolução do projeto.

1.3.2 Resultados de aprendizagem

Como já foi mencionado anteriormente, os resultados de aprendizagem (RA) são os mesmos nos dois anos letivos, mesmo com a mudança na metodologia. É bastante comum confundir resultados de aprendizagem (*learning outcomes*) com objetivos de aprendizagem (*learning objectives*). Na Figura 1 é possível vermos as diferenças entre os dois termos.

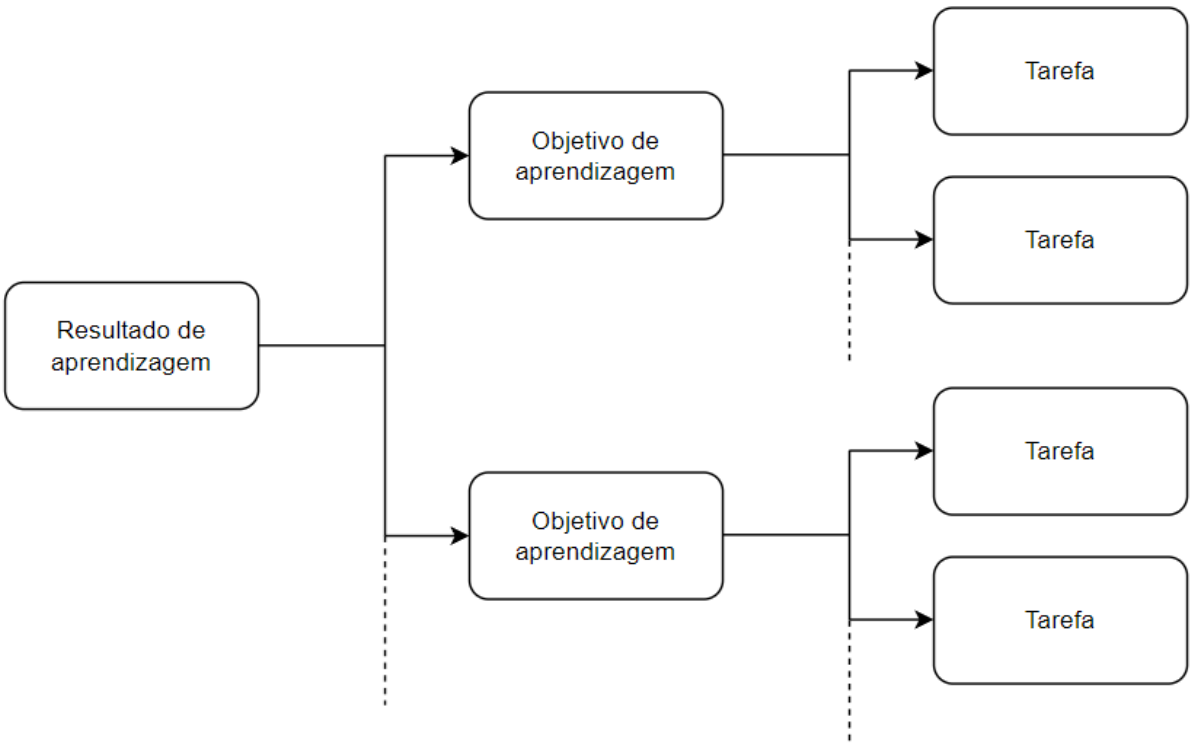


Figura 1 - Resultados de aprendizagem *vs.* objetivos de aprendizagem [5]

Os objetivos de aprendizagem representam as várias etapas necessárias para atingir determinados resultados de aprendizagem, podendo existir vários objetivos para cada resultado. Os resultados de aprendizagem são os mesmos para ambos os anos letivos; no entanto, os objetivos de aprendizagem devem ser diferentes. No primeiro ano, os objetivos de aprendizagem são mais voltados para metodologias tradicionais, enquanto no segundo ano, serão mais focados em metodologias ágeis.

No final desta UC, os estudantes devem ser capazes de demonstrar serem capazes de:

RA-1 – Descrever o desenvolvimento de software como uma disciplina de engenharia, utilizando a terminologia apropriada;

RA-2 – Realizar um projeto de software em equipa, seguindo processos definidos e boas práticas de engenharia de software;

RA-3 – Planear um projeto de desenvolvimento de software utilizando estimativas, calendarização de tarefas e alocação de recursos;

RA-4 – Aplicar as melhores práticas de gestão da qualidade, gestão de riscos, gestão de equipas e gestão de *stakeholders*;

RA-5 – Gerir um projeto de software, monitorizando e controlando o seu progresso, garantindo a entrega do software dentro dos prazos e custos planeados e gerindo as expectativas dos *stakeholders*;

RA-6 – Demonstrar conhecimento dos processos e metodologias de desenvolvimento de software, bem como dos princípios e fundamentos da melhoria de processos de software.

Estes resultados de aprendizagem definidos para a UC de GPS basearam-se fortemente nas diretrizes da IEEE Computer Society (IEEE – CS) [6], Association for Computing Machinery (ACM) [7] e Project Management Institute (PMI) [8], cujas boas práticas estão descritas em mais detalhes na secção 2.1.

Todos os resultados de aprendizagem podem ser demonstrados no exame teórico, enquanto os resultados de aprendizagem 1 a 5 podem ser demonstrados ao longo do semestre, com o desenvolvimento do trabalho prático.

1.4 Resultados e contribuições

Deste trabalho resultaram:

- Análise qualitativa do estado de arte no uso de metodologias de desenvolvimento de software em Portugal, tanto em ambiente académico como industrial;
- Onze novos processos relativos ao novo ciclo de vida do projeto a desenvolver pelos estudantes, devidamente desenhados (processos apresentados no capítulo 5);
- Quatro tutoriais para auxiliar os estudantes nas configurações pretendidas (Anexo N, Anexo O, Anexo R e Anexo T);
- Quatro *templates* a serem utilizados por todos os estudantes no âmbito de gestão do projeto (Anexo P, Anexo Q, Anexo S e Anexo U);
- Um artigo científico publicado numa conferência internacional – "*The 16th International Conference on Education Technology and Computers (ICETC 2024)*" – intitulado de "*From Traditional to Agile Methodologies in Software Project Management Education: A Case Study*" (Anexo V).

1.5 Estrutura do documento

A restante estrutura do documento é a seguinte:

No capítulo 2 são analisadas as boas práticas na engenharia de software e são analisadas algumas metodologias de desenvolvimento de software.

No capítulo 3 é apresentado um estudo realizado tanto na indústria como em ambiente académico relativamente à gestão de projetos de software. Para além disso, são mostrados alguns trabalhos relacionados.

No capítulo 4 é detalhado todo o funcionamento da UC de GPS até ao ano letivo 2022/2023. Adicionalmente é feita uma comparação entre GPS e UC semelhantes de outras IES, participantes no estudo do capítulo 3.

No capítulo 5 é apresentado o novo plano para a UC de GPS.

No capítulo 6 é detalhada a implementação desse novo plano, bem como é feita uma avaliação ao mesmo.

No capítulo 7 são apresentadas as conclusões e trabalhos futuros.

2 METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Existem essencialmente dois tipos de metodologias de desenvolvimento de software: metodologias tradicionais e metodologias ágeis. De forma resumida, tipicamente as metodologias tradicionais seguem um processo linear e sequencial, em que cada fase deve ser concluída antes de se iniciar a seguinte, dão ênfase à documentação exaustiva, e aplicam-se a projetos com âmbito fixo ou com calendários bem delineados, e com requisitos bem definidos. Já as metodologias ágeis, são iterativas e incrementais, centrando-se na flexibilidade, na colaboração e no feedback do cliente. Os projetos que seguem estas metodologias são divididos em iterações pequenas, permitindo que as equipas se adaptem às alterações dos requisitos e forneçam melhorias contínuas.

De acordo com vários autores [9-11] a escolha do ciclo de vida de desenvolvimento de software[2] (SDLC – *software development life cycle*) depende de vários fatores, incluindo o tamanho da equipa, o âmbito do projeto, a complexidade do software, o tempo disponível e o orçamento, entre outros. A seleção do SDLC é crucial para o sucesso de um projeto de software, sendo uma decisão que terá consequências a longo prazo. Antes de abordarmos qualquer metodologia mais detalhadamente, é importante apresentar alguns processos e boas práticas inerentes ao SDLC escolhido.

2.1 Conceitos fundamentais na engenharia de software e na gestão de projetos

De acordo com I. Sommerville [12], todas as metodologias de desenvolvimento de software precisam de conter processos para estas quatro atividades:

- **Especificação de software** – V. S. Alagar e K. Periyasamy [13] defendem que a especificação de software se refere ao ato de detalhar e descrever os "aspetos estruturais e comportamentais" do sistema a ser desenvolvido. A especificação de software é importante para que todos os membros do projeto (desde os elementos da equipa de gestão até aos elementos da equipa de desenvolvimento) trabalhem em conformidade com os requisitos e interfaces do sistema. Esta especificação difere entre as várias metodologias de desenvolvimento de software;
- **Desenvolvimento de software** – fase dedicada ao desenvolvido do software para ser entregue ao cliente;

[2] Ciclo de vida de desenvolvimento de software – é um processo estruturado utilizado por equipas de desenvolvimento de software para planear, desenvolver, testar e manter um determinado software.

- **Validação de software** – segundo M. Stoica, M. Mircea e B. Ghilic-Micu [14], a validação do software é muito importante, pois o desenvolvimento envolve processos complexos que muitas vezes estão predispostos a erros. Qualquer sistema de software deve ser testado e validado antes de ser entregue ao cliente, para garantir que foi desenvolvido e implementado de acordo com as especificações do projeto;
- **Evolução do software** – de acordo com T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse e M. Jazayeri [15], a única forma de evitar que o software fique desatualizado é "colocar a mudança no centro do processo de desenvolvimento de software". Caso não haja abertura para mudança e evolução, os "sistemas de software tornam-se desnecessariamente complexos e não confiáveis". Com isto, os autores sublinham que o software deve ser tratado como algo dinâmico e em constante evolução. O processo de desenvolvimento precisa ser estruturado para que a adaptação às mudanças seja uma prática natural e não uma exceção ou um problema a ser enfrentado.

Posto isto, e de acordo com o PMBOK do PMI, 6ª edição [16], existem dez áreas de conhecimento e processos importantes na gestão de projetos que são inerentes a qualquer metodologia de desenvolvimento de software, sendo elas:

- **Gestão de integração** – diz respeito à identificação, definição, combinação, unificação e coordenação de diversos processos e atividades de gestão de projetos;
- **Gestão do âmbito** – refere-se à área que contem processos e atividades para garantir que o projeto inclui apenas o trabalho necessário para garantir o seu sucesso;
- **Gestão de calendário** – é necessária para garantir a conclusão atempada do projeto;
- **Gestão de custos** – diz respeito à gestão financeira do projeto e é importante para que o projeto possa ser concluído dentro do orçamento previsto;
- **Gestão de qualidade** – inclui processos responsáveis por garantir que os objetivos dos *stakeholders* são atingidos;
- **Gestão de recursos** – os processos da área de gestão de recursos do projeto servem para garantir que os recursos estão no momento/local certo quando necessários;
- **Gestão de comunicação** – processos responsáveis por garantir que existe uma comunicação eficaz entre todas as partes interessadas;
- **Gestão de riscos** – o objetivo desta área de conhecimento é minimizar probabilidades e impactos de riscos do projeto, aumentando a probabilidade de o projeto ser bem-sucedido;

- **Gestão de aquisições** – diz respeito à aquisição de produtos ou serviços do projeto;
- **Gestão de *stakeholders*** – inclui processos necessários para gerir todas as partes interessadas do projeto;

Na Tabela 1 é possível vermos todos os processos identificados pelo PMBOK do PMI [16].

Tabela 1 - Áreas de conhecimento e respetivos processos apresentados no PMBOK do PMI [16]

Área de conhecimento	Processos
Gestão de integração	Desenvolver o contrato do projeto
	Desenvolver o plano de gestão do projeto
	Gerir o trabalho do projeto
	Gerir o conhecimento do projeto
	Monitorizar e controlar o trabalho do projeto
	Efetuar um controlo integrado das alterações
	Encerrar o projeto
Gestão de âmbito	Planear a gestão do âmbito
	Recolha de requisitos
	Definir o âmbito
	Criar "Work Breakdown Structure" (WBS)
	Validar o âmbito
	Controlar o âmbito
Gestão de calendário	Planear a gestão do calendário
	Definir atividades
	Sequência das atividades
	Estimar a duração das atividades
	Elaborar o calendário
	Controlar o calendário
Gestão de custos	Planear a gestão dos custos
	Estimar os custos
	Determinar o orçamento
	Controlar os custos
Gestão de qualidade	Planear a gestão de qualidade
	Gerir a qualidade
	Controlar a qualidade
Gestão de recursos	Planear a gestão de recursos
	Estimar recursos
	Aquisição de recursos

	Desenvolver a equipa
	Gerir a equipa
	Controlar os recursos
Gestão de comunicação	Planear a gestão das comunicações
	Gerir as comunicações
	Monitorar comunicações
Gestão de riscos	Plano de gestão de riscos
	Identificar riscos
	Efetuar uma análise qualitativa dos riscos
	Efetuar uma análise quantitativa dos riscos
	Planear as respostas aos riscos
	Implementar as repostas aos riscos
	Monitorar riscos
Gestão de aquisições	Planear a gestão das aquisições
	Efetuar aquisições
	Controlar as aquisições
Gestão de *stakeholders*	Identificar *stakeholders*
	Planear a participação dos *stakeholders*
	Gerir a participação dos *stakeholders*
	Monitorizar a participação dos *stakeholders*

2.2 Metodologias tradicionais

De acordo com E. Masciadra; F. Bilimleri Dergisi, M. Javanmard e M. Alian; e S. Shaikh e S. Abro [17-19], existem diferenças importantes entre as metodologias tradicionais e as metodologias ágeis. Resumindo, as metodologias tradicionais são:

- Mais rigorosas nos processos. As diferentes fases das metodologias tradicionais são bem delineadas e devem ser todas devidamente planeadas;
- Bastante focadas na documentação e no planeamento para prevenir eventuais riscos e para que seja possível prever o produto final. Esta documentação serve como um "contrato" entre a equipa de software e o cliente, de forma a garantir que é entregue o software ao cliente com os requisitos definidos pelo mesmo, dentro do prazo e orçamento estipulados;
- Limitadas relativo ao *feedback*. Assim que é iniciada uma fase, é altamente custoso voltar à fase anterior, o que faz com que não haja alterações de requisitos em fases adiantadas;
- Altamente controladas. As metodologias tradicionais são conhecidas pelo seu alto nível de controlo sobre o processo de desenvolvimento, com forte ênfase na monitorização e controlo do progresso e qualidade.

Em suma, o ciclo de vida de desenvolvimento de software tradicional é altamente centrado no processo. Estes tipos de metodologias de desenvolvimento de software são ideais para:

- Sistemas embebidos devido às restrições de integração do software com o hardware. Para este tipo de software é importante que as tecnologias sejam conhecidas, devido à necessidade de comunicação entre hardware e software. A interação entre hardware e software exige um alto nível de precisão e previsibilidade, tornando necessário um processo bem controlado e estruturado;
- Sistemas críticos onde a segurança funcional e certificações são pontos cruciais para o sucesso deste tipo de projetos. Para este tipo de software, é necessário que todos os requisitos estejam muito claros e muito bem definidos desde o início do projeto. Como mudanças durante o desenvolvimento podem comprometer a segurança e a conformidade regulatória, é essencial que todas as especificações estejam claras e completas antes do início do projeto.

Na Figura 2 é possível vermos em que situações devem ser utilizadas as metodologias tradicionais. Como mencionado, estas são ideais para projetos onde as tecnologias e os requisitos são conhecidas e claros.

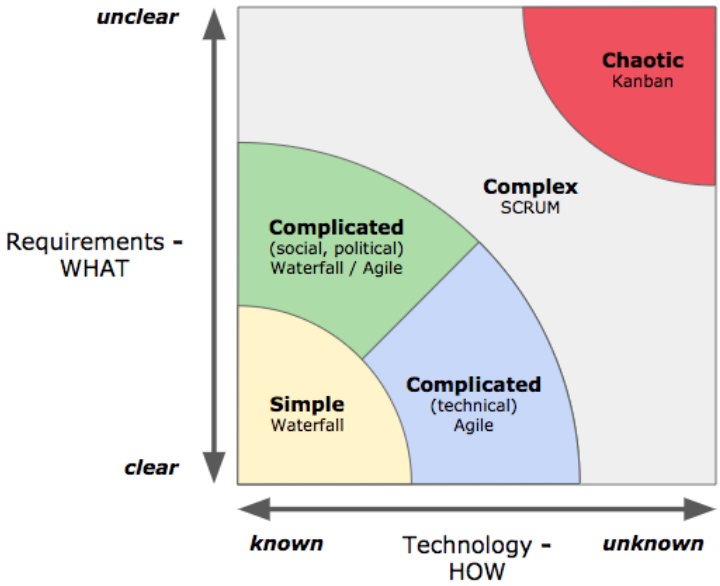


Figura 2 - Gráfico de que mostra quando devem ser utilizadas metodologias tradicionais, relacionando a clareza dos requisitos com o conhecimento relativo às tecnologias [54]

Existem várias metodologias tradicionais, tais como o Waterfall, Iterative Waterfall, Spiral, Rational Unified Process (RUP), entre outros.

2.2.1 Waterfall

De acordo com I. Sommerville [12], o modelo Waterfall é um exemplo de um processo definido por várias fases, em que a fase seguinte só deve ser iniciada quando a fase anterior é terminada, não sendo possível fazer alterações nessas fases posteriormente. Dessa forma, todo o planeamento do projeto é realizado antes do início do desenvolvimento. Segundo S. Shaikh e S. Abro [19], o Waterfall é uma metodologia intuitiva e fácil de entender devido à clareza das suas fases definidas. Além disso, possui outras vantagens, como testar o software apenas no final do desenvolvimento e fechar as fases apenas num momento.

Na Figura 3 é possível vermos um exemplo das várias fases do modelo Waterfall. O autor R. Mall [20] apresenta um exemplo de um modelo com seis fases, no entanto, as fases deste modelo podem variar dependendo da fonte.

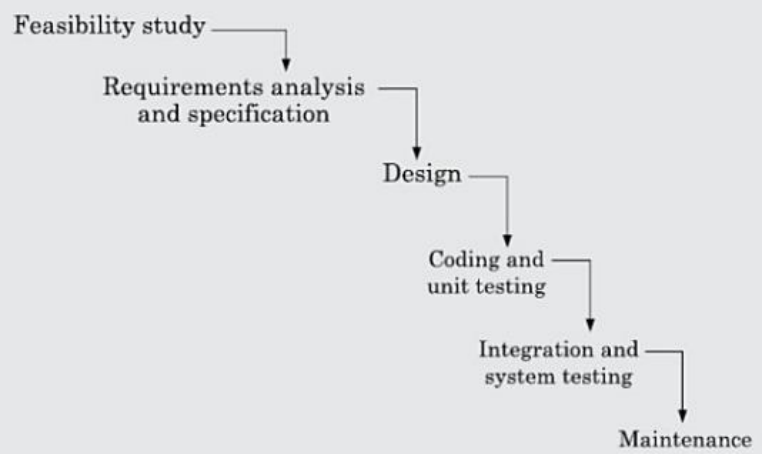


Figura 3 - Fases do modelo Waterfall [20]

Relativamente às fases apresentadas pelo autor, estas são as seguintes:

- **Estudo de viabilidade** – inicialmente é realizada uma pequena análise sobre a viabilidade do produto;
- **Análise e especificação de requisitos** – nesta fase são detalhados todos os requisitos do sistema. Esta especificação resulta num documento de requisitos de software, que funciona como um "contrato" entre o cliente e o fornecedor de software;
- **Design do software e do sistema** – nesta fase é detalhado o design do sistema e do software, incluindo arquiteturas, requisitos de software, hardware, entre outros;
- **Implementação e testes unitários** – nesta fase é desenvolvido todo o software e são realizados testes unitários às unidades de código desenvolvidas;

- **Integração e testes ao sistema** – nesta fase são integradas todas as unidades de código desenvolvidas na fase anterior e é testado o sistema para garantir que os requisitos foram cumpridos;
- **Manutenção** – R. Mall [20] defende que esta é a fase mais longa, visto que a manutenção engloba correções no software para problemas que não foram detetados nas fases iniciais do projeto. Na Figura 4, é possível visualizar que cerca de 60% do esforço pode ser gasto na manutenção de um software.

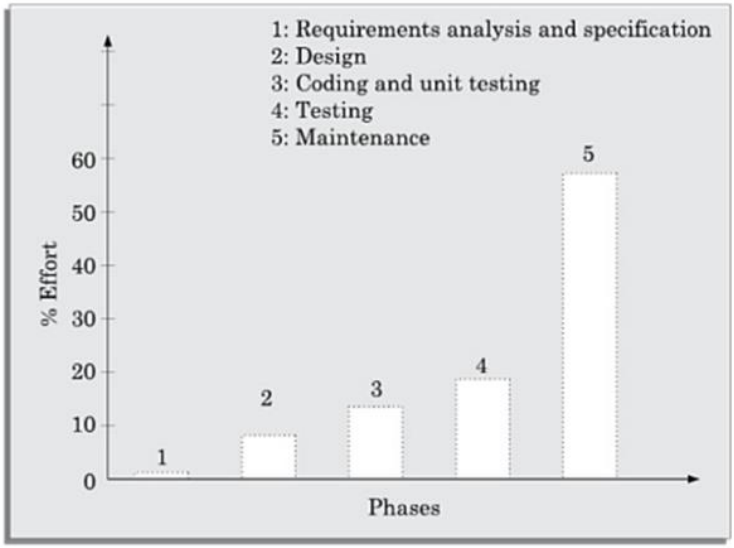


Figura 4 - Esforço de cada uma das fases [20]

No final de cada uma das fases, deve ser elaborado, pelo menos, um documento. Este deve ser aprovado para "oficializar" o término da respetiva fase.

De acordo com F. Bilimleri Dergisi, M. Javanmard e M. Alian [18] e com R. Mall [20], existem algumas desvantagens deste modelo, nomeadamente:

- **Mudanças de requisitos** – no Waterfall, não é suposto haver alterações de software durante o projeto. Essas mudanças de requisitos são muito dispendiosas, pois requerem retroceder a fases anteriores ou até mesmo à fase inicial para realizá-las, visto que os requisitos são detalhados logo numa fase inicial do projeto;
- **Correções de erros ineficientes** – como o desenvolvimento e teste do código são feitos numa fase muito avançada, é extremamente difícil corrigir erros de grande complexidade no final do desenvolvimento;
- **Segue um processo linear e sequencial** – como já foi referido anteriormente, devido ao seu processo sequencial, existe um grande esforço caso haja alteração de requisitos;

- **Sem sobreposição de fases** – é extremamente difícil evitar a sobreposição de fases. Para conseguir isso, alguns membros do projeto podem ficar inativos até que a fase atual seja concluída;
- **Falta de *feedback*** – durante todo o processo de desenvolvimento, não é entregue ao cliente qualquer tipo de versão funcional do produto. Isto pode levar ao não cumprimento de expectativas por parte do cliente;
- **Âmbito do projeto** – Dificuldade na definição de todos os requisitos no início do projeto, bem como na integração da gestão de riscos.

2.2.2 Iterative Waterfall

O Waterfall é um modelo bastante intuitivo e fácil de compreender. No entanto, só é aplicável em projetos onde se saiba exatamente o que se pretende desde o início, devido à dificuldade em fazer alterações de requisitos. Assim, como defendem R. Mall [20] e C. Kaur e V. Kumar [21], surgiu o Waterfall Iterativo. O modelo Waterfall Iterativo é um modelo que, como o próprio nome indica, se baseia no modelo Waterfall, mas introduz iterações para colmatar algumas das suas desvantagens.

Na Figura 5, é possível ver as fases do modelo. Observamos que as fases são as mesmas, no entanto, ocorrem em várias iterações, em vez de serem sequenciais, originando várias *releases* e permitindo obter *feedback*. À semelhança do modelo clássico, também aqui é necessário que a maioria dos requisitos seja clara desde o início do projeto. É de salientar que o exemplo apresentado não pode ser considerado uma regra e as fases do modelo podem ser diferentes consoante a fonte.

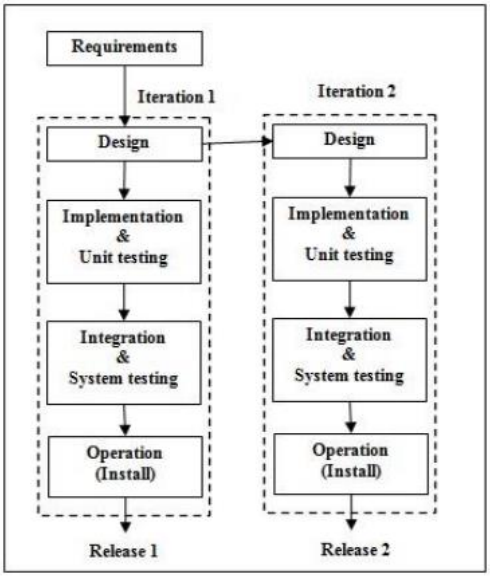


Figura 5 - Modelo Iterativo [21]

No entanto, apesar de este modelo colmatar algumas das desvantagens do modelo clássico, como afirmam R. Mall [20] e C. Kaur e V. Kumar [21], também possui algumas desvantagens, tais como:

- **Alterações de requisitos** – apesar de ser possível obter *feedback* do cliente, continua a ser difícil efetuar pedidos de alterações de requisitos, visto que maior parte dos requisitos continuam a ser definidos no início do projeto, antes de todas as iterações;
- **Fases rígidas nas iterações** – como é possível verificar na Figura 5, as iterações possuem as mesmas fases que o modelo clássico, o que significa que a rigidez presente nesse modelo persiste neste caso;
- **Sobreposição de fases** – continua a não ser possível haver sobreposição de fases;
- **Correção de erros** – tal como no modelo clássico, também no modelo iterativo os testes são realizados no final do desenvolvimento de cada iteração. Isso pode ser uma vantagem, no entanto, também pode resultar numa entrega tardia do software, aumentando os custos.

2.2.3 Spiral

Segundo R. Pressman [22], o modelo espiral é um processo evolutivo[3] de desenvolvimento de software que integra a abordagem iterativa da prototipagem com os elementos controlados e sistemáticos do modelo Waterfall. Na Figura 6 é possível vermos as diferentes fases do modelo Spiral.

N. Mohammed, A. Munassar e A. Govardhan [23] apresentaram um modelo Spiral com quatro fases, que não são consideradas uma regra fixa e podem variar conforme a fonte. As quatro fases apresentadas são:

- Planeamento;
- Análise do risco;
- Engenharia;
- Avaliação.

Estas quatro fases são realizadas de forma evolutiva, sendo que vão sendo repetidas sequencialmente. Uma das principais diferenças deste modelo comparado aos modelos anteriormente apresentados é o foco que é dado à análise de riscos e, como já foi referido anteriormente, à prototipagem.

[3] Modelo evolutivo - no contexto de desenvolvimento de software refere-se a uma abordagem que permite o desenvolvimento gradual e iterativo de um sistema, sendo que o projeto evolui iteração a iteração. Ao invés de haver um plano rígido elaborado logo no início, este vai sendo ajustado, conforme o resultado de cada iteração. A avaliação do risco no final de cada iteração pode servir para tomar decisões sobre a próxima (ou até cancelar o projeto).

Inicialmente, é feita uma análise de requisitos com uma avaliação do risco (terceiro quadrante, segundo quadrante e metade do primeiro quadrante da Figura 6). Nesta avaliação, é realizado um processo de análise de riscos e possíveis soluções dos mesmos. Terminada a análise de riscos, é desenvolvido um protótipo (correspondente à segunda metade do primeiro quadrante da Figura 6).

Já na fase de engenharia, é desenvolvido e testado o software (quarto quadrante da Figura 6). Na fase de avaliação, como o nome indica, é feita uma avaliação pelo cliente do software produzido anteriormente. Terminadas estas quatro fases, inicia-se outro ciclo de quatro fases e assim sucessivamente.

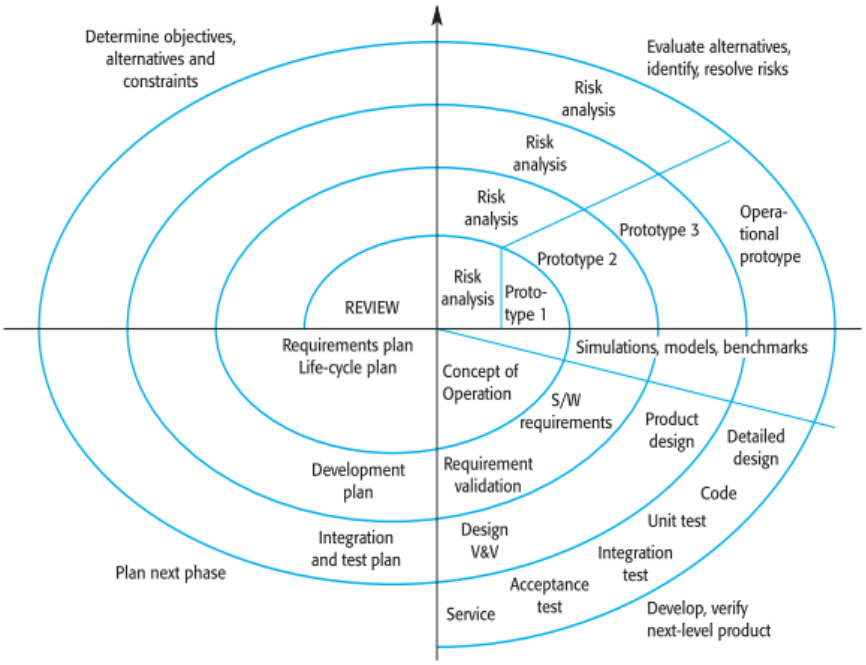


Figura 6 - Modelo Spiral [23]

De acordo N. Mohammed, A. Munassar e A. Govardhan; A. Mujumdar, G. Masiwal e P. M. Chawan; A. Alshamrani, A. Bahattab e I. Fulton [23-25] as vantagens do modelo Spiral são:

- Bastante destaque à análise do risco;
- O software não é desenvolvido numa fase tão avançada do projeto;
- O *feedback* do cliente já não é tão limitado como nos modelos anteriormente apresentados;
- A prototipagem ajuda os *stakeholders* a visualizarem rapidamente o estado/progresso do projeto;

- Foi desenhado com base nas boas práticas do Waterfall e da prototipagem[4].

No entanto, este modelo também conta com algumas desvantagens, nomeadamente:

- A análise de riscos requer conhecimento específicos;
- Pode tornar-se num modelo caro de utilizar;
- O sucesso do projeto está muito dependente da análise de riscos;
- Não é bom para projeto curtos, visto que a alta análise de riscos pode tornar o projeto caro e demorado.

Resumindo, o modelo em espiral é um modelo mais seguro de utilizar, devido à forte análise dos riscos, no entanto, essa análise pode implicar aumentos de custos e de tempos. Também pode ser o modelo escolhido quando os clientes não têm uma visão clara do que pretendem.

2.2.4 Rational Unified Process (RUP)

De acordo com G. Kumar e P. K. Bhatia [26], o RUP "é uma abordagem iterativa para sistemas orientados a objetos." Como é possível ver na Figura 7, O RUP é organizado com base em dois conceitos: fases e fluxos de trabalho. À semelhança das outras metodologias de desenvolvimento de software, o RUP também apareceu para tentar ir ao encontro com as necessidades das organizações e para tentar ter os seus processos mais eficientes.

Na Figura 7, podemos observar as quatro fases presentes no RUP. É de salientar que todas as fases, à exceção da *Inception*, têm iterações. As diferentes fases são:

- ***Inception*** – nesta fase são definidos o âmbito e os requisitos do projeto;
- **Elaboração** – nesta fase é realizado todo o planeamento do projeto, nomeadamente a parte mais técnica do mesmo. Como defende [27], no final desta fase é feita uma análise de riscos e é realizada uma avaliação para decidir se o projeto deve prosseguir ou não. Assim, pode-se afirmar que esta é a fase mais crítica do projeto;
- **Construção** – esta fase corresponde à implementação e testes do software. No final desta fase, é suposto existir uma versão beta e um manual de utilizador;
- **Transição** – por fim, nesta fase ocorre a entrega do software, bem como toda a manutenção do mesmo.

[4] Prototipagem – é uma técnica utilizada no software, que consiste em demonstrar ideias / possíveis requisitos através de desenhos.

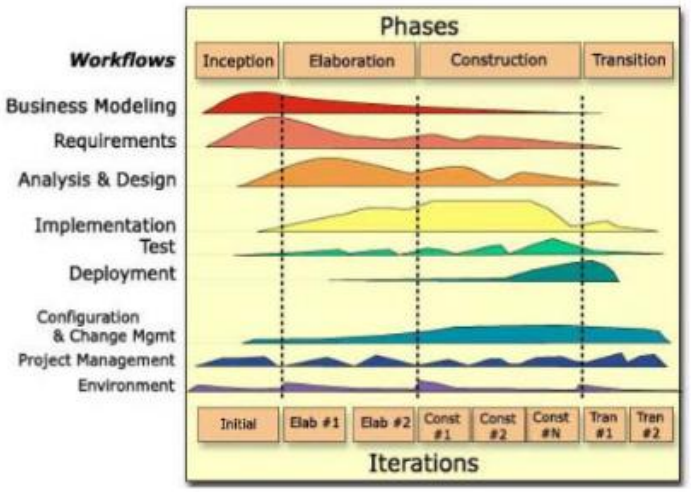


Figura 7 - Fases do modelo RUP [26]

De acordo com A. Anwar [27] e J. Manalil [28], o RUP apresenta algumas vantagens, tais como:

- Ao contrário de outras metodologias tradicionais, esta abordagem não é tão focada em documentação, permitindo dedicar mais tempo à codificação do software;
- Visto que existem várias iterações, é possível mitigar riscos mais cedo quando comparado com processos sequenciais;
- Também devido às diversas iterações, é possível efetuar algumas alterações de requisitos ou tecnológicas;
- Devido aos testes mais regulares, a qualidade do software pode ser superior quando comparado com o desenvolvimento sob metodologias tradicionais.

No entanto, como afirma S. Shaikh e S. Abro [19], o RUP também tem algumas desvantagens, tais como:

- No caso de alterações tecnológicas durante o projeto, a reutilização de componentes pode tornar-se um problema. O RUP enfatiza a reutilização de componentes existentes para acelerar o desenvolvimento, reduzir custos e aumentar a consistência do sistema. No entanto, quando ocorre uma alteração tecnológica significativa durante o projeto, os componentes planeados para reutilização podem tornar-se incompatíveis ou exigirem modificações significativas;
- É necessário que exista um gestor de projetos experiente, visto que este processo é demasiado difícil e complicado de aplicar corretamente;
- Não é adequado para pequenos projetos.

2.2.5 V-Model

A metodologia de desenvolvimento de software V-Model é uma abordagem de desenvolvimento de software derivada da metodologia Waterfall, onde cada fase do ciclo de vida é associada a uma fase correspondente de teste. Na Figura 8 é possível vermos o ciclo de vida do modelo V-Model. O modelo tem o formato de um "V", simbolizando o relacionamento direto entre etapas de definição e etapas de verificação e validação. I. Sommerville [12] descreve o modelo V-Model como um processo rigoroso e bem estruturado que enfatiza a qualidade através de atividades de teste sistemáticas

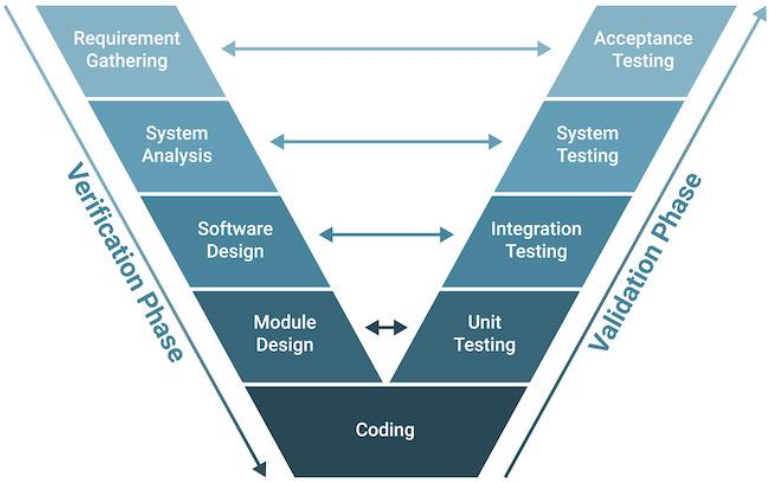


Figura 8 - Fases do modelo V-Model [29]

O modelo V-Model tem duas fases: verificação e validação. Cada fase da verificação corresponde a uma fase de validação. No que diz respeito às fases, estas são:

- Recolha de requisitos – identificação detalhada das necessidades dos *stakeholders* e especificação dos requisitos do sistema. A fase de validação correspondente é a fase de Testes de aceitação, onde os requisitos são validados contra os resultados;
- Análise do sistema – tradução dos requisitos em funcionalidades do sistema. A fase de validação corresponde é a fase de Testes de sistema, que verificam se o sistema funciona de acordo com as especificações funcionais;
- Design de sistema – estruturação da arquitetura geral do sistema. A fase de validação correspondente é a fase de Testes de Integração, que garantem a interação correta entre os módulos;

- Design de componentes – definição de detalhes técnicos e especificação de módulos e componentes. A fase de validação correspondente é a fase de Testes unitários, que verificam cada componente isoladamente.
- Implementação – desenvolvimento do software com base nos designs especificados.

2.3 Metodologias ágeis

Segundo I. Sommerville [12], nos anos 90 começaram a surgir algumas práticas ágeis, sendo o Extreme Programming (XP) a metodologia ágil mais significativa na altura.

No entanto, antes de abordar qualquer metodologia ágil, é fundamental mencionar o Manifesto Ágil [30]. Até 2001, ano em que o manifesto foi criado, os processos de desenvolvimento de software eram considerados processos pesados. Havia, portanto, a necessidade de criar uma alternativa que evitasse a documentação excessiva e esses processos complexos. Assim, surgiu o Manifesto Ágil, que é um conjunto de valores e princípios para o desenvolvimento de software.

De acordo com o Manifesto Ágil [30], as metodologias ágeis pretendem valorizar mais:

- Os indivíduos e interações do que processos e ferramentas;
- O software funcional do que documentação abrangente;
- A colaboração com o cliente do que negociação contratual;
- A resposta à mudança do que seguir um plano.

O manifesto ágil foi orientado por doze princípios do desenvolvimento ágil, sendo eles:

1. Satisfazer o cliente através de uma entrega rápida e contínua de software;
2. Aceitar alterações de requisitos mesmo numa fase avançada do ciclo de desenvolvimento;
3. Fornecer frequentemente software funcional;
4. Ter o cliente e a equipa a trabalhar juntos ao longo de todo o projeto;
5. Desenvolver projetos com base em indivíduos motivados;
6. Utilizar conversas pessoais e diretas para transmissão de informação;
7. Ter a entrega de software funcional como principal medida de progresso;
8. Promover o desenvolvimento sustentável através de processos ágeis, permitindo que promotores, a equipa e os utilizadores mantenham um ritmo constante indefinidamente;

9. Manter sempre em atenção a excelência técnica e um bom desenho da solução para aumentar a agilidade;
10. Focar na simplicidade, maximizando o trabalho que não é feito;
11. Fazer surgir as melhores arquiteturas, requisitos e desenhos a partir de equipas auto-organizadas;
12. Refletir regularmente sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.

Assim e de acordo com I. Sommerville [12], F. Bilimleri Dergisi, M. Javanmard e M. Alian [18] e S. Kumar Dora e P. Dubey [31], algumas das principais características das metodologias ágeis passam por:

- Centrar as atividades das metodologias ágeis no design e implementação do software, ao invés de se focar nos processos;
- Permitir alterações de requisitos devido às suas constantes entregas de software;
- Gerar valor numa fase muito inicial do projeto;
- Manter um baixo nível de documentação;
- Ser ideal para pequenas equipas;
- Garantir bastante comunicação entre a equipa e o cliente.

No que diz respeito às práticas do desenvolvimento ágil, estas começaram a ser introduzidas nos anos 90. Desta forma, com as práticas já existentes na altura e com o conjunto de novas práticas, Sommerville [12] afirma que as mais importantes são:

- ***User Story*** (US) – uma US deve representar um cenário possível de utilização que pode ser experienciado por um determinado utilizador do produto. Estas US podem ser "partidas" em várias tarefas, sendo elas planeadas e desenvolvidas pelas equipas de desenvolvimento de software. Por norma, o cliente é que prioriza os requisitos da aplicação, sendo esta priorização refletida na prioridade das US. A objetivo das US é especificar os requisitos do sistema de uma forma mais simples e menos detalhada comparativamente à especificação de requisitos nas metodologias tradicionais;
- ***Refactoring*** – é uma técnica utilizada no desenvolvimento de software e consiste na restruturação sistemática de código já existente, de forma a torná-lo mais simples, claro ou robusto. Visto que a mudança é um princípio fundamental da engenharia de software atualmente, deve-se prever possíveis alterações ao software e ao design de forma a implementar essas mudanças facilmente, quando necessárias;

- ***Test-first development*** – é uma prática que se baseia na criação dos testes antes do desenvolvimento da funcionalidade. Desta forma é possível prever e planear o comportamento de uma determinada funcionalidade;
- ***Pair programming*** – esta prática resume-se a que dois programadores trabalhem em pares para desenvolverem o software, a partir do mesmo computador. Assim, existe uma partilha de conhecimento, existe menor probabilidade de erros, o software é desenvolvido de uma forma mais rápida, entre outras vantagens.

2.3.1 Scrum

Segundo o State of Agile [3], o Scrum é a metodologia ágil mais utilizada atualmente. No entanto, e de acordo com F. Tsui, O. Karam e B. Bernal [32], o Scrum é muitas vezes utilizado juntamente com outras práticas de outras metodologias ágeis de desenvolvimento de software.

K. Schwaber e J. Sutherland [4], K. Bhavsar, Dr. V. Shah e Dr. S. Gopalan [33] e M. Alqudah e R. Razali [33] apresentam várias práticas, artefactos e *roles* do Scrum.

As práticas apresentadas por estes são:

- ***Sprint*** – As *sprints* são uma espécie de ciclo de vida de um incremento, iteração ou *release* no Scrum. Por norma, as durações das *sprints* variam entre duas a quatro semanas. Geralmente, depois de uma *sprint* estar planeada, não é alterada;
- ***Sprint planning*** – como o nome indica, refere-se à atividade do planeamento da *sprint*, sendo que é decidido com base no *product backlog* – artefacto do Scrum que será apresentado a seguir – quais são as *features*/requisitos que vão ser desenvolvidos no decorrer da *sprint*.
- ***Daily scrum meeting*** – esta é uma reunião diária, curta e objetiva, idealmente feita de pé. O principal objetivo desta reunião é perceber o progresso da *sprint*. Geralmente os elementos da equipa identificam o que foi feito desde a última reunião, o que farão até à próxima e falam de eventuais situações bloqueantes;
- ***Sprint review*** – no final de cada *sprint*, os elementos da equipa juntamente com o cliente fazem uma revisão do resultado da mesma, sendo que são mostradas todas as *features*/requisitos que foram desenvolvidos no decorrer da *sprint* ao cliente;
- ***Sprint retrospective*** – o objetivo da *sprint retrospective* é identificar possíveis alterações para melhorar a qualidade e a eficiência da equipa nas próximas *sprints*.

- ***Product backlog refinement*** – esta atividade refere-se ao ato da revisão e alteração dos itens que estão presentes no *product backlog*.

Relativamente aos artefactos, estes são:

- ***Product backlog*** – No *product backlog* existe uma lista de *features* e requisitos (nomeadamente US) que serão implementados. Este *product backlog* deve ser constantemente refinado pelo *product owner* – posteriormente serão apresentadas as diversas *roles* no Scrum;
- ***Sprint backlog*** – à semelhança do *product backlog*, o *sprint backlog* também é uma lista de *features* e requisitos, mas referentes apenas à *sprint*. No fundo, a maior diferença entre os itens da *sprint backlog* e os itens do *product backlog* está no nível de detalhe dos mesmos;
- ***Product increment*** – software desenvolvido, testado e pronto a ser entregue ao cliente desde a última iteração.

Para terminar, as *roles* apresentadas por estes são:

- ***Product owner*** – responsável por comunicar com o cliente e transmitir as informações à equipa de desenvolvimento e vice-versa e toma decisões perante a equipa quanto ao produto a desenvolver;
- ***Scrum master*** –faz a gestão da equipa, protege a equipa de interferências externas e facilita a resolução de problemas;
- ***Team member*** – membro da equipa.

Na Figura 9 é possível ver o fluxo do Scrum.

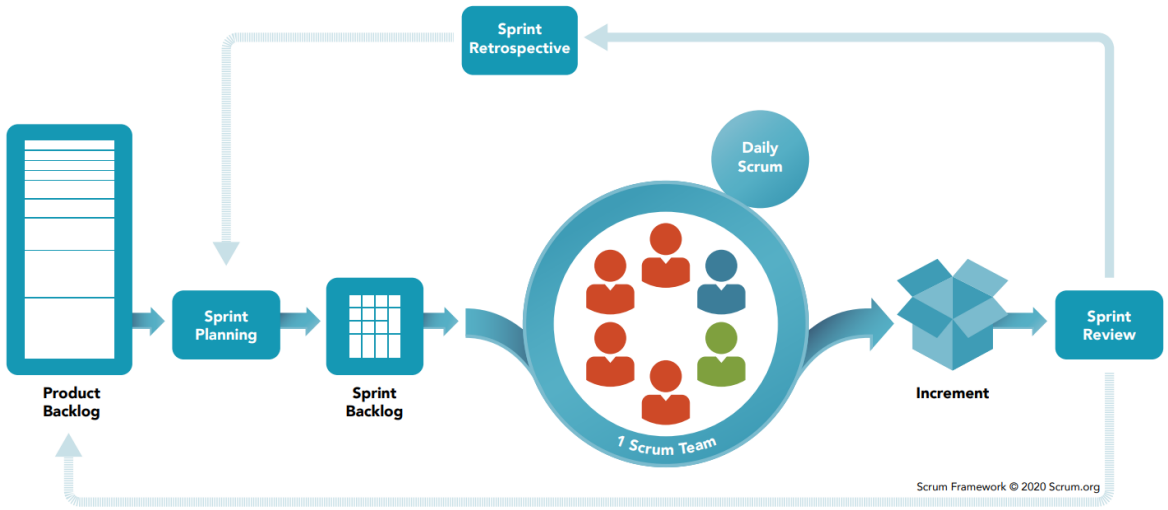


Figura 9 - Fluxo do Scrum [4]

Inicialmente temos o *product backlog*, artefacto que fica presente ao longo de todo o ciclo de vida do projeto. Posteriormente, temos uma iteração, que corresponde à *sprint*. Esta inicia-se com o *sprint planning*, gerando assim o *sprint backlog*. Ao longo da *sprint*, é realizada a *daily scrum meeting*. No final da *sprint*, existe um *product increment*[5], sendo o input da *sprint review* para que seja feita uma demonstração ao cliente. De seguida, é feita a *sprint retrospective*, terminando assim a *sprint* atual e dando início à próxima.

Como já foi referido na secção 1.1, em 2020 houve pequenas mudanças no Scrum, sendo que foram introduzidos os *commitments*:

- **O *commitment* do *product backlog* é o *product goal*** – o *product goal* descreve um estado futuro do produto;
- **O *commitment* da *sprint backlog* é o *sprint goal*** – o *sprint goal* descreve o estado final da *sprint*;
- **O *commitment* do *increment* é o *definition of done*** – o *definition of done* é o termo que se dá quando as funcionalidades cumprem os critérios de aceitação, originando um incremento ao produto.

2.3.2 Kanban

Segundo K. Bhavsar, Dr. V. Shah e Dr. S. Gopalan [34], o Kanban é uma metodologia ágil transparente que se resume à visualização do estado atual do projeto de software através de um quadro (*kanban board*). D. Anderson e A. Carmichael [35] afirmam que o objetivo deste quadro é tornar visível todo o trabalho que está a ser desenvolvido no decorrer do projeto, garantindo que o "serviço trabalha com a quantidade certa" – existe uma regra que será explicada posteriormente. No fundo, e de acordo com A. Janes [36], esta metodologia foi adotada para reduzir perdas de tempo e para evitar processos desnecessários.

Na Figura 10 é possível ver o fluxo do Kanban. K. Bhavsar, Dr. V. Shah e Dr. S. Gopalan [34] dão um exemplo que contem três listas:

- ***To-Do list*** – nesta lista estão todas as US que foram devidamente priorizadas e que futuramente serão desenvolvidas. Antes de se iniciar o desenvolvimento das mesmas, estes itens passam para a lista de *In-progress*;
- ***In-Process list*** – nesta lista estão todas as US que estão a ser desenvolvidas no momento. Quando terminadas, os itens passam para a lista *Done*;
- ***Done list*** – nesta lista estão todas as US que já foram desenvolvidas.

Para além das três listras, existem mais dois quadros, denominados de *product backlog* e *product release*.

[5] Product increment - é o termo que se dá ao valor adicionado ao produto

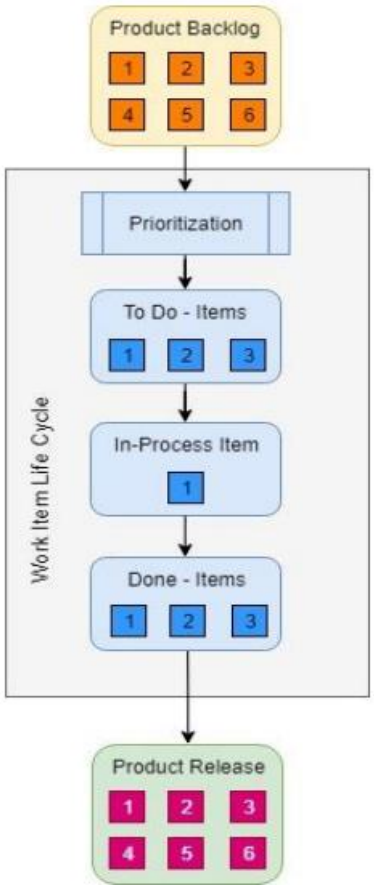


Figura 10 - Fluxo do Kanban [34]

No entanto, o Kanban não são apenas estes dois quadros e as três listas na *kanban board*. Existem regras relativamente aos quadros, como por exemplo o limite do trabalho em progresso (WiP – Work In Progress). Resumidamente, o WiP é uma regra que define qual o limite de trabalho que pode estar em execução. Caso a equipa atinja o limite do WiP, esta não pode iniciar novos trabalhos até que termine alguma coisa. O WiP surgiu porque, segundo D. Anderson e A. Carmichael [35], "ter demasiado trabalho parcialmente concluído é um desperdício e dispendioso e, fundamentalmente, prolonga os prazos de entrega, impedindo a organização de responder aos seus clientes e à evolução das circunstâncias e oportunidades."

Para além do WiP, existem várias métricas que podem ser aplicadas quando é seguido o Kanban, como por exemplo métricas de medição de trabalho concluído ou entregue num determinado período. Estas métricas são denominadas de *delivery rate* e podem ser observadas a partir do número de tarefas realizadas ou US finalizadas que foram movidas para a *done-list*.

No que diz respeito a *roles*, D. Anderson e A. Carmichael [35] defendem que não existem *roles* obrigatórias, no entanto, existem duas importantes:

- ***Service request manager*** ou ***product owner*** – faz um papel semelhante ao do *product owner* no Scrum. É responsável por compreender as necessidades do cliente, selecionar as funcionalidades/requisitos e ordená-los;
- ***Service delivery manager*** – é o responsável pelo fluxo de trabalho e por garantir que o software é entregue ao cliente conforme o esperado.

2.3.3 Extreme Programming

De acordo com o XP Guide [37], "o XP melhora um projeto de software de cinco formas: comunicação, simplicidade, *feedback*, respeito e coragem". Antes de se analisar o ciclo de vida do XP, é importante falar nas cinco áreas fundamentais, nomeadamente o planeamento, a gestão, o *design*, a codificação e os testes.

Relativamente ao planeamento, as equipas devem:

1. Escrever as US;
2. Criar o *release scheduled* através do *release planning*;
3. Planear entregas pequenas;
4. Planear o projeto em diversas iterações;
5. Iniciar cada iteração com o seu planeamento.

No que diz respeito à gestão, as regras são:

1. Dar à equipa um espaço de trabalho aberto;
2. Definir um ritmo de trabalho constante;
3. Iniciar os dias com uma *daily meeting*;
4. Medir a velocidade do projeto;
5. Rodar as pessoas nas suas funcionalidades para evitar dependências de pessoas em determinadas funcionalidades;
6. Ajustar o processo quando necessário.

Já relativo ao design, as equipas devem:

1. Manter um design simplista;
2. Escolher uma metáfora do sistema – é uma prática que é usada no XP para substituir arquiteturas standard do projeto usadas nas metodologias tradicionais;
3. Utilizar CRC Cards (*Use Class, Responsibilities and Collaboration Cards*) para sessões de design de software – os CRC Cards servem para que a equipa faça *brainstorming* de design;

4. Realizar *spikes* para reduzir riscos – um *spike* é desenvolvimento que não está integrado com produto e serve para que a equipa possa explorar possíveis soluções;
5. Efetuar *refactoring* sempre que possível;

Relativamente à codificação, as equipas devem:

1. Ter o cliente sempre disponível;
2. Escrever o código seguindo standards definidos;
3. Utilizar a prática *test-first*;
4. Programar todo o código que vai para produção seguindo a prática *pair-programming*;
5. Permitir que apenas um par integre o código de cada vez;
6. Integrar o código regularmente;
7. Utilizar um servidor dedicado a integrações;
8. Garantir *Collective ownership* – significa que todas as fases do produto (planeamento, gestão, design, codificação e testes) são da responsabilidade da equipa.

Para terminar, no que diz respeito aos testes, a equipa deve:

1. Garantir que todo o código tenha testes unitários;
2. Assegurar que todo o código passe nos testes unitários antes de ser entregue;
3. Criar testes assim que algum *bug* for encontrado para garantir que a origem do bug é testada novamente;
4. Executar testes de aceitação com regularidade.

Na Figura 11 é possível ver o ciclo de vida do XP.

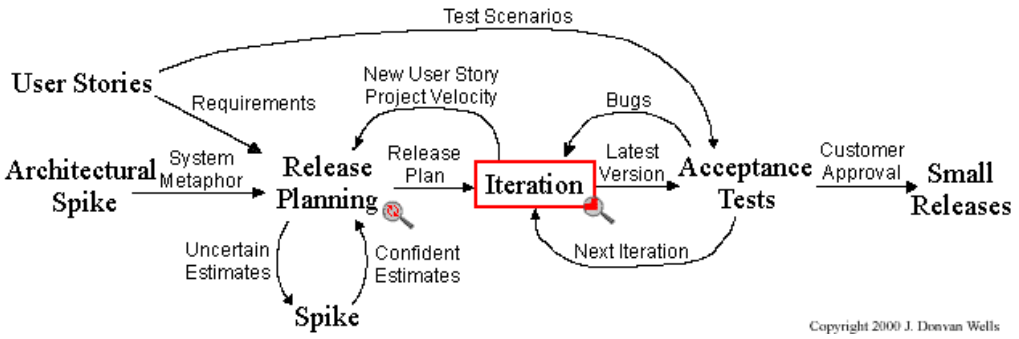


Figura 11 - Ciclo de vida do XP [37]

Inicialmente temos as US que são descritas pelos clientes. Tendo as US escritas, são extraídos os cenários de teste e os requisitos do sistema, para que posteriormente seja feito o *release planning*.

Relativamente ao *release planning*, como o nome indica, é feito o planeamento da *release*. Caso exista dúvidas nas estimativas, é feito um *spike* para que seja feita uma estimativa com base em conhecimento.

Em seguida, é iniciada a iteração. Na Figura 12 é possível visualizarmos o ciclo de vida da mesma.

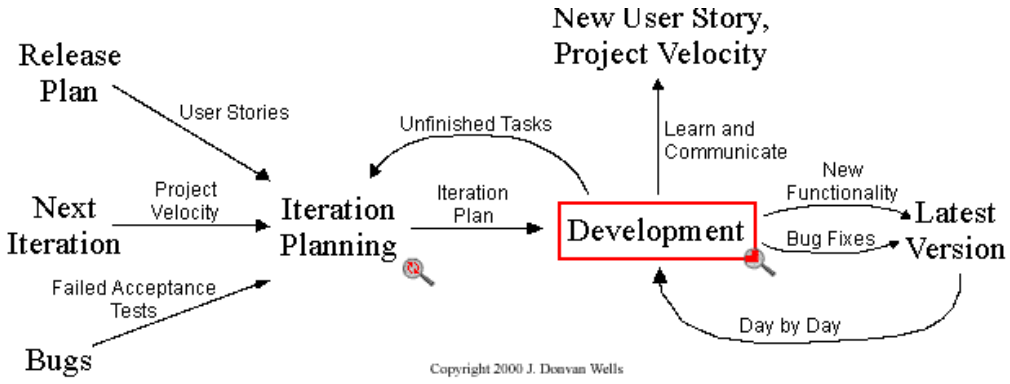


Figura 12 - Ciclo de vida de uma iteração [37]

Todas as iterações começam com um planeamento da iteração. Esse planeamento é feito com base nas US da iteração, bem como nos bugs que surgiram na iteração anterior. Em seguida, procede-se ao desenvolvimento das funcionalidades, sendo importante salientar que, caso surja uma nova funcionalidade, ela pode ser incluída.

Após o desenvolvimento estar concluído, são realizados os testes de aceitação, que podem ou não originar a deteção de alguns bugs. Caso sejam encontrados bugs durante estes testes, são feitas as correções necessárias na próxima iteração. Por fim, é realizada uma entrega ao cliente.

De acordo com R. Mall [20] e R. Fojtik [38], a metodologia XP é mais aplicável a:

- Projetos que envolvam novas tecnologias ou projetos de investigação;
- Projetos de pequenas dimensões.

À semelhança das outras metodologias ágeis, e também conforme afirma R. Fojtik [38], uma das vantagens do XP deve-se ao facto de permitir alterações de requisitos previamente definidos e a inclusão de novas funcionalidades rapidamente.

2.3.4 DevOps

De acordo com L. Leite, C. Rocha, F. Kon, D. Milojicic e P. Meirelles [39], o interesse pelo DevOps aumentou bastante na última década. R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer [40] realizaram um *survey* relativamente ao termo DevOps e verificou-se que existem várias definições, tais como:

- É o resultado da junção dos termos Desenvolvimento e Operações (*development and operation*) – esta é a definição mais comum;
- É um conjunto de princípios e práticas que permitem um trabalho em equipa entre as equipas de desenvolvimento e as equipas de operações;
- É um paradigma ou um conjunto de princípios e práticas que preenchem uma lacuna da comunicação entre as equipas de desenvolvimento e as equipas de operações;
- É definido como um método de desenvolvimento de software moderno para responder às interdependências entre as equipas de desenvolvimento e operações, unificando métodos modernos e ferramentas resultando numa convergência real entre programadores e operadores;
- É um paradigma ou um conjunto de princípios focados na entrega contínua do software, que pretende responder rapidamente às mudanças e uso automatizado de pipelines de entrega, reduzindo os tempos dos ciclos de vida;
- É definido como um paradigma ou conjunto de princípios que permite automatizar a entrega do software entre ambientes de desenvolvimento aos ambientes de produção;
- É definido como uma prática que enfatiza as tarefas permitindo a integração contínua entre o desenvolvimento de software e suas necessidades de implantação operacional;
- É definido como um método que combina as preocupações de garantia de qualidade com operações e práticas de desenvolvimento para melhorar o desempenho;
- É uma cultura que permite às equipas agilizaram e automatizarem alguns processos, podendo ser considerada uma extensão do ágil.

Resumindo, pode concluir-se que DevOps é um conjunto de práticas e processos, que ajudam na comunicação entre as equipas de desenvolvimento e operações, permitindo uma entrega contínua automatizada do software entre os diversos ambientes.

De acordo com o State of DevOps [41], existe um conjunto de práticas relacionadas com o termo DevOps, sendo elas técnicas, processuais, de medição e culturais.

2.3.4.1 Práticas técnicas

As práticas técnicas são:

- ***Version control*** – o controlo de versões é importante tanto para visualização de histórico de ficheiros, como para permitir que vários programadores estejam a trabalhar paralelamente no projeto, como para existir uma integração contínua;
- ***Continuous integration*** **(CI)** – a integração contínua permite aos programadores que façam *merges*[6] constantemente de forma automatizada e com qualidade, visto que permite que sejam executados alguns testes antes da integração, por exemplo;
- ***Deployment automation*** – os *deploys*[7] automáticos permitem aos programadores que sejam feitos *deploys* em vários ambientes reduzindo esforço e tempo;
- ***Trunk-based development*** – existem dois tipos de padrões no controlo de versões: *feature branches* e *trunk-based development*. O padrão *feature branche* é usado para criar uma *branch*[8] para cada funcionalidade. O padrão *trunk-based development* é uma prática onde os programadores dividem o trabalho em pequenas partes e fazem *merges* regularmente para a *branch* principal, sendo que essa *branch* principal tem de estar sempre pronta para entrar em ambientes de produção[9];
- ***Continuous testing*** – é possível garantir a alta qualidade do software desenvolvendo qualquer tipo de testes automatizados, sendo possível acionar a execução dos testes a partir de *pipelines*[10] de entrega contínua;
- ***Continuous delivery*** **(CD)**– a entrega contínua, muitas vezes confundida com o *continuous deployment*[11] no termo CI/CD, é uma prática de lançamento de versões de forma mais rápida, segura e sustentável, visto que é a entrega de novas versões nos diversos ambientes (incluindo produção) a qualquer momento, correndo poucos riscos;
- ***Architecture*** – uma boa arquitetura é bastante importante na medida em que permite que as equipas possam fazer várias mudanças, implementações,

---

[6] *Merge* – ação que aplica as modificações de um determinado *branch* para outro *branch*.
[7] *Deploy* – disponibilização de versões de uma aplicação que estava em desenvolvimento.
[8] *Branch* – no desenvolvimento de software, uma *branch* refere-se ao ato da criação de um novo ramo no código de software, permitindo que os programadores façam novos desenvolvimentos em paralelo, sem alterar o código original.
[9] Ambiente de produção – ambiente operacional do software, para ser utilizado pelos utilizadores finais.
[10] *Pipeline* – Conjunto de processos a serem executados de forma sequencial.
[11] *Continuous deployment* – prática de atualização de software contínua, disponibilizando o software para os clientes finais assim que a *pipeline* termine com sucesso.

entregas, testes, etc, sem que haja dependência de equipas, serviços, entre outros aspetos.

- ***Cloud infrastucture*** – de acordo com um estudo realizado em 2019 e efetuado pelo State of Devops [41], as empresas que tinham as suas infraestruturas na *cloud* tinham 2,6 vezes mais hipóteses de estimar o custo da operação do software com mais precisão. Para além disso, têm uma melhor disponibilidade de serviços e um melhor desempenho de entrega de software;
- ***Test data management*** – a qualidade dos dados dos testes é crucial, visto que são usados em qualquer tipo de teste, incluindo testes manuais e automatizados. Bons dados de teste permitem testar casos muito específicos, casos extremos, reproduzir falhas e simular erros;
- ***Empowering teams to choose tools*** – o DevOps é uma cultura que é muito dependente de ferramentas. Na Figura 13 é possível ver uma "tabela periódica" com um conjunto de ferramentas disponíveis nas diversas áreas. A utilização de ferramentas aumenta o desempenho das equipas, obtendo melhores resultados;
- ***Shifting left on security*** – de acordo com o State of DevOps 2016 [42], as equipas de alto desempenho passam 50% menos tempo a corrigir problemas relacionados com segurança quando comparadas a equipas de baixo desempenho. O termo *shifting left on security* surgiu do facto das questões relacionadas serem agora abordadas já no início do ciclo de vida do desenvolvimento de software, ao invés de serem abordadas só no final do ciclo.
- ***Database change management*** – refere-se à gestão e manutenção de uma base de dados, envolvendo a monitorização e o controlo de alterações de estruturas e dados de bases de dados existentes;
- ***Code maintainability*** – é fácil um projeto de software escalar no número de linhas de código, tornando-o difícil de ler e com código muitas vezes replicado. A manutenção do código é importante dado que permite às equipas reutilizarem código sempre que necessário e adicionarem novas dependências e fazerem *upgrades* constantes às versões das mesmas.

2.3.4.2 Práticas processuais

No que diz respeito às práticas processuais, estas são:

- ***Team experimentation*** – os programadores podem apenas desenvolver funcionalidades para os requisitos que foram pedidos pelo cliente. No entanto, para que sejam aproveitadas as técnicas modernas do desenvolvimento de software, é necessário que sejam feitos testes com dados reais para alcançar os resultados pretendidos. Assim, os programadores

podem realizar protótipos e testar ideias de forma a validar possíveis cenários e prevenir eventuais problemas que apareçam posteriormente;

- ***Streamlining change approval*** – de acordo com o State of DevOps 2019 [43], as aprovações de alterações automatizadas complementadas com aprovações manuais são melhores do que simplesmente aprovações manuais ou automatizadas;
- ***Customer feedback*** – as equipas têm um melhor desempenho quando recolhem métricas de satisfação dos clientes com frequência, recolhem e seguem o *feedback* dos clientes relativamente à qualidade do produto e quando usam esse *feedback* para projetarem funcionalidades;
- ***Visibility of work in the value stream*** – o termo visibilidade do trabalho refere-se ao entendimento do fluxo de trabalho de cada empresa/equipa até que o produto seja entregue ao cliente;
- ***Working in small batches*** – trabalhar em pequenas partes é importante em qualquer metodologia em que o *feedback* seja fundamental. Caso contrário, a mudança de qualquer requisito ficaria muito mais custosa;

2.3.4.3 Práticas de medição

Relativamente às práticas de medição, estas são:

- ***Monitoring and observability*** – a monitorização e a observabilidade podem ser fatores que contribuem positivamente para a entrega contínua, dado que podem ser extraídos diversos relatórios e métricas;
- ***Monitoring systems to inform business decisions*** – a monitorização é um aspeto importante nas equipas de software visto que, como já foi dito anteriormente, é possível extrair-se dados dessas monitorizações. Assim, esse *feedback* pode ser útil para localizar e corrigir certos problemas de forma rápida;
- ***Proactive failure notification*** – as notificações proativas de falhas são uma mais-valia, visto que permitem às equipas detetarem quando é que os sistemas estão próximos de falhar, podendo evitar essas mesmas falhas;
- ***Work in process limits*** – por vezes, com o aumento de trabalho, algumas pessoas ficam com tarefas em demasia. Este facto deve-se ao elevado tempo para conclusão de cada tarefa e o mau desempenho em cada uma delas. Assim, devem priorizar o trabalho, limitar o trabalho por pessoa e concentrar-se em terminar as tarefas prioritárias;
- ***Visual management capabilities*** – é uma prática usada pelas equipas para que sejam mostradas informações pertinentes, de forma clara, a todos os elementos;

Figura 13 - Tabela periódica de ferramentas do DevOps [55]

2.3.4.4 Práticas culturais

Para finalizar, as práticas culturais:

- ***Job satisfaction*** – a satisfação no trabalho é um fator crucial para o bom desempenho;
- ***Generative organizational culture*** – uma cultura organizacional de alta confiança pode influenciar o desempenho da entrega de software e o desempenho organizacional no setor de tecnologia;
- ***Trasnformational leadership*** – o resultado da entrega de software pode estar diretamente relacionado com a capacidade de liderança;
- ***Learning culture*** – uma cultura que valoriza a aprendizagem faz com que as pessoas implementem com maior frequência.

2.4 Discussão

Neste capítulo, foram analisadas as principais diferenças entre as metodologias tradicionais e ágeis na gestão de projetos de software. As metodologias tradicionais caracterizam-se pela rigorosidade nos processos e na extensa documentação. Apesar de terem surgido algumas metodologias tradicionais que tentam colmatar algumas falhas do clássico Waterfall, todas elas continuaram muito dependentes de processos bem definidos e muito baseadas em documentação. Além disso, existe outra grande desvantagem nas metodologias tradicionais que nunca conseguiu ser superada com o surgimento de novas metodologias: a resposta à mudança/alteração de requisitos.

Por outro lado, as metodologias ágeis promovem a flexibilidade, adaptabilidade e a entrega contínua de valor ao cliente. Ao contrário das metodologias tradicionais, as metodologias ágeis permitem que as equipas respondam rapidamente a alterações de requisitos, através da realização de ciclos de desenvolvimento relativamente curtos e iterativos. Esta dinâmica facilita a integração de *feedback* contínuo e a melhoria contínua do produto.

Além disso, nos últimos anos há uma nova tendência, o DevOps. Esta cultura parece ter vindo a crescer cada vez mais, ajudando as equipas de desenvolvimento de software em vários fatores, como por exemplo a produtividade/velocidade de desenvolvimento, prevenção de problemas, robustez e acessibilidade, entre outros.

Posto isto, também é importante perceber o estado atual da gestão de projetos de software na indústria e como é que é lecionada a gestão de projetos de software noutras IES, de forma a ajudar na tomada de decisão.

3 A GESTÃO DE PROJETOS DE SOFTWARE NA INDÚSTRIA E NA ACADEMIA

É essencial perceber como é que a gestão de projetos de software está a ser abordada tanto na indústria como noutras IES, para que seja possível fazermos um alinhamento com as necessidades do mercado, acompanharmos as tendências tecnológicas na área da gestão de projetos de software e para conseguirmos recolher os pontos positivos de todas as outras IES. Para tal, foi realizado um trabalho de campo, que consistiu em pequenas entrevistas com estudantes e professores que frequentaram ou frequentam UC semelhantes a GPS e com empresas próximas do ISEC.

Estas entrevistas tinham como objetivos:

- Perceber as metodologias de desenvolvimento de software que estão atualmente a ser utilizadas pelas empresas para a gestão de projetos de software ou a ser lecionadas nas escolas;
- Compreender os processos de trabalho utilizados, bem como as adaptações efetuadas relativamente à metodologia de trabalho;
- Identificar as ferramentas utilizadas pelas equipas;

3.1 Indústria

É importante compreender o estado atual da indústria, nomeadamente como algumas empresas influentes a nível regional, nacional e internacional fazem a gestão de projetos de software. Agradecemos desde já às empresas Enso Origins, AIRC, EasyClick, DoDoc, Critical Software, Stratio Automotive, Capgemini Engineering, Altice Labs e Wit Software por terem participado neste estudo.

Para além dos objetivos mencionados, as entrevistas com as empresas tinham também o propósito de:

- Entender as maiores dificuldades enfrentadas pelas empresas ao acolher recém-licenciados do ISEC.

Os resultados são apresentados de forma anónima para garantir a confidencialidade dos dados.

3.1.1 Metodologias de desenvolvimento

No que diz respeito às metodologias de desenvolvimento, na Figura 14 estão apresentadas as diversas metodologias utilizadas pelas empresas.

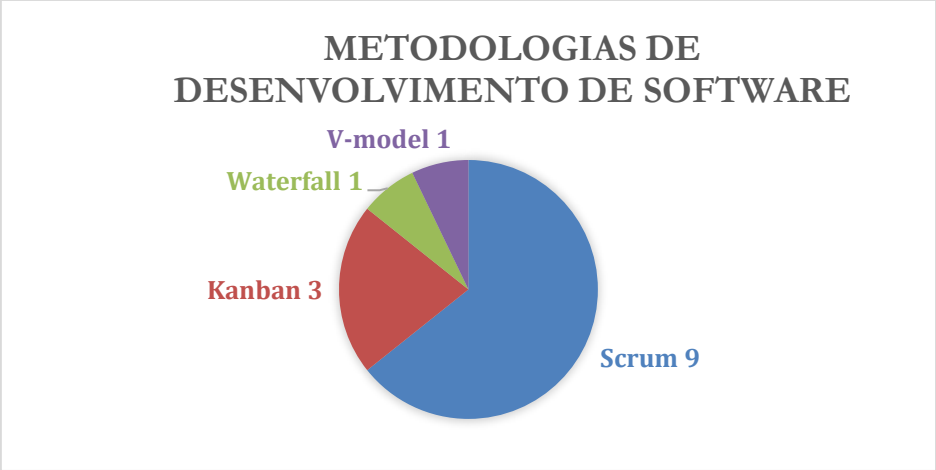


Figura 14 - Metodologias de desenvolvimento de software utilizadas pelas empresas participantes no estudo

Das nove empresas participantes no estudo, todas elas possuem equipas que seguem o Scrum e três delas também têm equipas que adotam o Kanban. Adicionalmente, duas empresas optam ainda pela utilização do Waterfall e do V-model em projetos específicos.

3.1.2 Processos de trabalho

No que diz respeito aos processos de trabalho, estes não podem ser apresentados sob a forma de gráfico, uma vez que as empresas se concentraram em diferentes pontos dos seus processos na entrevista realizada. Portanto, os dados serão apresentados de forma mais descritiva.

No que toca à metodologia, praticamente todas as empresas começam por seguir as metodologias conforme prescritas, ou seja, adotam os processos específicos da metodologia em questão e, posteriormente, efetuam ajustes conforme necessário. É importante destacar que, em algumas ocasiões, uma empresa realiza *sprints* com duração de uma semana, enquanto outra opta por *sprints* de um mês.

Quanto às estimativas, praticamente todas as empresas as baseiam na complexidade ou no esforço [12], em vez de no tempo[13].

No que se refere aos critérios de aceitação, estes variam de empresa para empresa. São utilizados vários tipos de teste, incluindo testes de cobertura, *pipelines* para

[12] Estimativas baseadas em complexidade – as tarefas são estimadas relativamente à dificuldade ou complexidade técnica da tarefa

[13] Estimativas baseadas em tempo - as tarefas são estimadas diretamente em termos de quantidade de tempo (em horas, dias ou semanas), até que esta esteja completa

execução de testes de qualidade de código com recurso à ferramenta SonarQube, testes de aceitação, testes de integração, testes de sistema e testes unitários.

Em relação às regras do Git, a maioria das empresas segue um *Git workflow*, nomeadamente *feature based*. Para além disso, existem várias regras, como revisão de código (por vezes realizada por mais do que uma pessoa) e a obrigatoriedade de passarem pelos diversos ambientes antes de entrar no ambiente de produção. Também é importante salientar que, pelo menos, três empresas usam *pipelines* CI/CD, automatizando assim o processo de entrega de software.

3.1.3 Ferramentas

Relativamente às ferramentas, há uma variedade de opções disponíveis para diversas áreas de trabalho. Na Figura 15, é possível verificar as principais ferramentas utilizadas pelas empresas.

Figura 15 - Ferramentas utilizadas pelas empresas participantes no estudo

Das nove empresas, oito utilizam o Jira como ferramenta de gestão de projetos, enquanto a nona empresa utiliza uma ferramenta proprietária.

Outra ferramenta relevante é o Git, que é utilizado por todas as empresas como ferramenta de versionamento de código. Esta contagem inclui empresas que possuem servidores Git, repositórios GitLab e repositórios GitHub.

No que diz respeito à gestão de documentação, destaca-se o Confluence, utilizado por sete das empresas.

Além destas três ferramentas, não há outras que sejam amplamente utilizadas pelas empresas inquiridas.

3.1.4 Análise empresarial aos estudantes do Instituto Superior de Engenharia

Nas entrevistas realizadas, questionou-se às empresas que habitualmente acolhem estagiários/recém-licenciados do ISEC e de outras IES sobre a preparação dos estudantes relativamente à gestão de projetos de software e foi pedida uma comparação entre estes. Também se perguntou se consideram relevante ajustar os processos de trabalho ou metodologias utilizadas na unidade curricular em função das competências adquiridas pelos estudantes. Das nove empresas participantes, cinco forneceram respostas, que se resumem da seguinte forma:

- Empresa A: Em comparação com estudantes de outras IES, os estudantes do ISEC têm menos conhecimento sobre metodologias e conceitos, sendo considerado "crítico" que saiam de uma licenciatura em Engenharia Informática sem familiaridade com metodologias ágeis, dada a sua ampla utilização atualmente;
- Empresa B: As empresas não esperam que os estudantes saiam da licenciatura em Engenharia Informática com experiência em gestão de projetos de software, mas valorizam os que possuem bases sólidas em metodologias e conceitos de gestão de projetos de software;
- Empresa C: Existe uma perceção de desconhecimento por parte dos estudantes do ISEC sobre como aplicar metodologias ágeis em contextos reais. No entanto, mostram-se adaptáveis e abertos a aprender, tanto em termos técnicos quanto na área de gestão de projetos de software;
- Empresa D: Consideram que abordar uma metodologia tradicional nesta UC pode ser uma vantagem, pois é mais fácil adaptar uma metodologia tradicional para uma ágil do que o contrário;
- Empresa E: As empresas não esperam que os estudantes tenham um vasto conhecimento em gestão de projetos de software ao ingressar na empresa, pois isso é adquirido com a experiência, algo que os estagiários/recém-licenciados ainda não possuem. No entanto, consideram úteis todos os conceitos abordados na UC, independentemente da metodologia utilizada. Desta forma, seria benéfico se os estudantes tivessem um pouco mais de conhecimento sobre testes de software e práticas de metodologias ágeis.

3.2 Ambientes académicos

Para este estudo, foram abordadas algumas IES que contêm unidades curriculares semelhantes a GPS. Essas IES foram: Universidade do Porto, Universidade de Coimbra, Universidade de Lisboa (Instituto Superior Técnico e Faculdade de Ciências da Universidade de Lisboa) e o Instituto Politécnico de Leiria.

Como já foi mencionado, é importante compreender como estas UC são ministradas nas diferentes IES, para conseguirmos recolher os pontos positivos. No geral, nas aulas teóricas são abordados temas relacionados com a gestão de projetos de software, incluindo metodologias de desenvolvimento de software; planeamento, monitorização e controlo de projetos; estimativas e entrega de software; gestão de equipas; plano de desenvolvimento de software; requisitos de software; gestão de qualidade, como testes, planos de qualidade, revisões e inspeções; e controlo de versões de software. Posteriormente, estes conceitos são postos em prática nas aulas práticas da UC, com o desenvolvimento de um projeto em grupo ao longo de todo o semestre.

Nas aulas práticas, os estudantes realizam um projeto de software em grupo, onde cada grupo deve utilizar uma metodologia de desenvolvimento de software para planear, gerir e implementar o projeto proposto.

À semelhança da indústria, também foram realizadas entrevistas individuais com as IES, para que fosse possível obtermos respostas aos objetivos previamente mostrados. Desta forma, foram abordados os seguintes tópicos nas entrevistas:

1. Organização das equipas;
2. Metodologia utilizada;
3. Ciclo de vida do projeto;
4. Gestão do esforço;
5. Critérios de aceitação dos requisitos;
6. Gestão de riscos;
7. Regras relacionadas com o Git;
8. Ferramentas utilizadas.

De forma a facilitar a identificação das escolas, daqui em diante, cada uma vai ser identificada da seguinte forma: <sigla da UC>–<sigla da unidade orgânica>. Por exemplo, para identificar a UC de GPS no ISEC, a sigla será GPS–ISEC.

3.2.1 Universidade do Porto

A UC equivalente a GPS na Faculdade de Engenharia da Universidade do Porto (FEUP), na Licenciatura em Engenharia Informática e de Computação é a UC de Engenharia de Software (ES). Relativamente a ES–FEUP:

- **Organização de equipas:** Cada grupo tinha, em média, seis elementos;
- **Metodologia**: Scrum – cumpriam todas as cerimónias do Scrum;
- **Ciclo de vida do projeto:** *Sprint* 0 que consistia na formação dos grupos e decisão sobre o tema (este ficava ao critério de cada grupo). De seguida, quatro *sprints* com duração de três semanas cada;
- **Gestão do esforço**: Eram feitas estimativas para quatro horas semanais por cada elemento do grupo. Exemplo: Um grupo de seis elementos, tinha de ter um *sprint planning* de setenta e duas horas (6 elementos x 4 horas semanais x 3 semanas = 72 horas);
- **Critérios de aceitação**: Testes de aceitação manuais e automatizados. Relativamente ao Git, os *merge requests* (MR) tinham de ser aceites por, pelo menos, dois elementos diferentes;
- **Gestão do risco**: N/A;
- **Regras de Git**: Usavam um *Git workflow feature based*. Para além disso, e como já mencionado, os estudantes eram obrigados a fazer MR antes de enviarem o seu código para a *branch* principal;
- **Ferramentas/tecnologias utilizadas**:
    - GitHub – ferramenta de versionamento de código, gestão de documentação e gestão do projeto;
    - Figma – *design* e *mockups*;
    - Flutter – linguagem obrigatória no desenvolvimento da aplicação móvel;
- **Outros aspetos relevantes:** Usavam o READ.ME do repositório para documentação. Este devia de seguir um formato fornecido pelo professor.

3.2.2 Universidade de Coimbra

A UC equivalente a GPS na Faculdade de Ciências e Tecnologias da Universidade de Coimbra (FCTUC), na Licenciatura em Engenharia Informática é a UC de Engenharia de Software (ES). No que diz respeito a ES–FCTUC:

- **Organização das equipas:** Cada turma constituía um grupo de trabalho, sendo que dentro de cada grupo existiam miniequipas de cinco elementos;

- **Metodologia**: Scrum (Scrum of Scrums) – cumpriam todas as cerimónias do Scrum. Para além disso, existia uma reunião entre todos os *scrum masters* de cada uma das miniequipas, para todos terem conhecimento do progresso do trabalho;
- **Ciclo de vida do projeto:** Existem quatro *sprints* quinzenais. Para além disto, existem duas semanas no início do semestre para criação de *issues* na ferramenta de gestão do projeto;
- **Gestão do esforço**: Eram feitas estimativas para quatro horas semanais por cada elemento do minigrupo. Exemplo: Um minigrupo de cinco elementos, tinha de ter um *sprint planning* de quarenta horas (5 elementos x 4 horas semanais x 2 semanas = 40 horas);
- **Critérios de aceitação**: Testes unitários ou testes de aceitação. Existia um bónus para os testes automatizados;
- **Gestão do risco:** As funcionalidades com mais risco eram desenvolvidas logo no início do projeto;
- **Regras de Git**: Cada turma tinha um repositório no GitLab. Era utilizada a funcionalidade GitLab CI/CD para que pudessem fazer o *deploy* automático para um servidor. Este aspeto era obrigatório a partir de meio do semestre, obrigado os estudantes a fazerem as demonstrações diretamente do servidor e não em máquinas locais;
- **Ferramentas/tecnologias utilizadas**:
    - GitLab – ferramenta de versionamento de código, gestão de documentação e gestão do projeto;
    - Python/Django – linguagem e *framework* obrigatórias no desenvolvimento da aplicação web;
- **Outros aspetos relevantes:**
    - A atribuição do trabalho é feita por funcionalidades e não por módulos, ou seja, cada miniequipa tem de desenvolver todos os módulos da sua funcionalidade (*backend*, *frontend*, ligação à base de dados, etc.);
    - O professor é que faz o papel de cliente desde início, sendo que o tema é escolhido por ele e é igual para todas as turmas;
    - Segundo o professor, a estratégia de *branching* deve ser melhorada, de forma a existir mais processos definidos e menos flexibilidade por parte dos estudantes. Outro aspeto a melhorar é a remoção da obrigatoriedade da linguagem definida para o *backend*. Considera ser um aspeto negativo pois nem todos os estudantes se sentem confortáveis com Python.

3.2.3 Universidade de Lisboa – Instituto Superior Técnico

A UC equivalente a GPS no Instituto Superior Técnico (IST), na Licenciatura em Engenharia Informática e de Computadores é a UC de Engenharia de Software (ES). No que diz respeito a ES–IST:

- **Organização das equipas:** Cada grupo tinha, em média, seis elementos. Cada grupo era subdivido em três pares, sendo que cada par era responsável por desenvolver as suas funcionalidades;
- **Metodologia:** Scrum – faziam todas as cerimónias;
- **Ciclo de vida do projeto:** Existiam três *sprints* quinzenais;
- **Gestão do esforço**: Cada *sprint* tinha de ter vinte *story points*. No final da *sprint*, todos os grupos tinham de estar num ponto equivalente do projeto, uma vez que os requisitos eram os mesmos para todos os grupos;
- **Critérios de aceitação**: Testes de carga e testes *end-to-end*. Relativamente ao Git, os MR tinham de ser aceites por, pelo menos, um elemento de cada par;
- **Gestão do risco:** N/A;
- **Regras de Git**: Era obrigatório existirem a *branch master* e *develop*. Para além destas, tinha de existir uma *branch* para cada par. Terminado o desenvolvimento de cada par, era feita uma integração na *branch develop* a partir de MR obrigatórios;
- **Ferramentas/tecnologias utilizadas**:
  - GitHub – ferramenta de versionamento de código, gestão de documentos e gestão de projeto;
- **Outros aspetos relevantes:**
  - O professor é que faz o papel de cliente desde início, sendo que o tema é escolhido por ele e é igual para todas as turmas. Para além disso, o professor também entregava um modelo do domínio base.

3.2.4 Universidade de Lisboa – Faculdade de Ciências da Universidade de Lisboa

A UC equivalente a GPS na Faculdade de Ciências da Universidade de Lisboa (FCUL), na Licenciatura em Engenharia Informática é a UC de Projeto de Sistemas de Informação (PSI). Relativamente a PSI–FCUL:

- **Organização das equipas:** Cada grupo tinha, em média, seis elementos;
- **Metodologia:** Scrum;
- **Ciclo de vida do projeto:** Existiam três *sprints* quinzenais;

- **Gestão do esforço**: N/A;
- **Critérios de aceitação**: Os estudantes têm flexibilidade para escolherem quais os tipos de testes que querem fazer para cumprirem os critérios de aceitação;
- **Gestão do risco:** N/A;
- **Regras de Git**: N/A;
- **Ferramentas/tecnologias utilizadas**:
    - Git – ferramenta de versionamento de código. Recomendado, mas não obrigatório;
    - JIRA – ferramenta de gestão de projeto;
    - Discord – ferramenta de comunicação;
- **Outros aspetos relevantes:**
    - O professor é que faz o papel de cliente desde início, sendo que o tema é escolhido por ele e é igual para todas as turmas. Para além disso, entrega já uma lista de US priorizadas e com critérios de aceitação já definidos;
    - Não é obrigatório que os estudantes façam todas as US entregues pelo professor. A avaliação é feita com base no planeamento *vs* entrega;
    - Visto que o projeto começa a ser desenvolvido apenas a meio do semestre, na primeira parte do semestre os estudantes adquirem conhecimentos sobre desenvolvimento web e algumas ferramentas de utilização obrigatória no trabalho.

3.2.5 Instituto Politécnico de Leiria

A UC equivalente a GPS na Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria, na Licenciatura em Engenharia Informática é a UC de Tópicos Avançados de Engenharia de Software (TAES). Relativamente a TAES–ESTG:

- **Organização das equipas:** Cada grupo tinha, em média, seis elementos;
- **Metodologia:** Metodologia ágil adaptada para o ensino, que é um misto entre Scrum, XP e Kanban – utilizam as demonstrações e *sprint retrospective* a cada aula;
- **Ciclo de vida do projeto:** Seis *sprints* semanais;
- **Gestão do esforço**: N/A;

- **Critérios de aceitação**: Testes funcionais manuais e automatizados, sendo que utilizam a técnica Behavior Driven Development (BDD)[14]. Relativamente ao Git, os estudantes são obrigados a fazer MR para a *branch master*;
- **Gestão do risco:** N/A;
- **Regras de Git**:
    - Existência de, pelo menos, duas *branches*;
    - Usam *smart commits;*
    - Obrigatoriedade de utilização de MR;
    - Existência de dois repositórios, um para testes e outro para código;
- **Ferramentas/tecnologias utilizadas**:
    - Git – ferramenta de versionamento de código;
    - JIRA – ferramenta de gestão de projeto;
    - Katalon – ferramenta de testes automatizados;
- **Outros aspetos relevantes:**
    - Visto que o projeto começa a ser desenvolvido apenas a meio do semestre, na primeira parte do semestre os estudantes realizam alguns exercícios que serão úteis para a realização do projeto, nomeadamente sobre arquiteturas de software e resolução de problemas da arquitetura em metodologias ágeis, testes manuais e automatizados, algumas das ferramentas que vão utilizar e abordam os processos das metodologias ágeis;
    - Os estudantes já têm conhecimentos de testes de software em UC anteriores;

Uma vez que a UC de GPS será detalhadamente abordada no capítulo 4, a análise comparativa das IES será realizada na secção 4.3, dado que apenas nesse ponto teremos toda a informação necessária para a comparação, tanto do ISEC como de outras instituições.

[14] BDD – abordagem de validação de software que surgiu como sendo uma extensão da prática *test-first*

3.3 Trabalhos relacionados

Esta transição do ensino tradicional para o desenvolvimento ágil já foi feita noutras IES, havendo vários exemplos na literatura. Neste subcapítulo, apresentamos alguns trabalhos com objetivos semelhantes ao nosso.

M. Alshayeb, S. Mahmood e K. Aljasser [44], fizeram um estudo onde estudantes de engenharia de software criaram um produto, seguindo metodologias tradicionais e ágeis. O objetivo do artigo era relatar a sua experiência na passagem da metodologia tradicionais (Waterfall) para metodologias ágeis na realização deste projeto de engenharia de software com a duração de dois semestres. Para avaliar se houve melhorias na aprendizagem dos estudantes, vinte e oito alunos responderam a um inquérito com nove perguntas. Os resultados "mostraram que a abordagem ágil ajudou os estudantes a ter um design globalmente melhor e a evitar os erros que cometeram no design inicial concluído na primeira fase do projeto de finalização". As conclusões, baseadas nos questionários dos estudantes, mostraram que existiam mais vantagens seguindo uma metodologia ágil. No entanto, o facto de este estudo ter sido realizado no âmbito de um projeto final de curso, pode influenciar os resultados, pois os alunos podiam já ter mais conhecimentos sobre gestão de projetos de software.

R. Wlodarski, A. Poniszewska-Maranda e J. R. Falleri [45] realizaram uma experiência durante um semestre em duas escolas de engenharia. O objetivo do trabalho era avaliar o impacto das metodologias de desenvolvimento de software no sucesso dos estudantes. Foram avaliadas equipas que seguiam as metodologias tradicionais e ágeis e equipas que não seguiam qualquer metodologia. A avaliação baseou-se em métricas de qualidade do produto, produtividade da equipa e qualidade do trabalho em equipa. Os resultados mostram que, no que diz respeita à qualidade do produto, a equipa que não utilizou qualquer metodologia teve mais qualidade interna e a equipa que seguiu metodologias tradicionais teve mais qualidade externa. Em termos de produtividade da equipa, nenhuma equipa se destacou. Em termos de qualidade da equipa, a equipa que seguiu metodologias ágeis obteve melhores resultados. As diferenças entre este trabalho e o nosso residem no facto de o estudo ter sido realizado no mesmo ano, mas em IES diferentes. É possível que os alunos tenham sido preparados de forma diferente para a experiência devido a variações nos seus currículos e professores. Além disso, existe outra distinção notável relativamente ao processo de avaliação, visto que se centraram apenas em métricas de qualidade e produtividade.

V. Kate, S. Bhalerao e V. Sharma [46] fizeram um estudo que tinha como objetivo comparar a fase de testes em cada uma das metodologias. Quatro pontos foram considerados: "Dividir e conquistar", "Mudanças são sempre bem-vindas", "Estimativa de tempo e custo" e "Satisfação do cliente". Os resultados mostram que as metodologias ágeis têm sempre melhores resultados quando comparados com as

metodologias tradicionais. As diferenças neste trabalho são que, neste estudo, os autores apenas consideraram a fase de testes e não foi desenvolvido apenas em ambiente académico.

A. Sinha e P. Das [47] exploram a aplicação de metodologias ágeis no contexto académico e da gestão de projetos de software. Concluíram que as metodologias ágeis trazem muitos benefícios, mas que a liberdade nos processos pode ser uma desvantagem para os estudantes. Por esta razão, afirmam que "os professores e os estudantes devem esforçar-se por adotar uma abordagem equilibrada, considerando tanto as metodologias ágeis como as tradicionais". Esta era a principal razão pela qual ainda não tinha sido feita a transição para o ágil – ou seja, sem um processo rigoroso como o Waterfall, a transição podia falhar e tornar-se num projeto caótico.

Por último, D. F. Rico e H. H. Sayani [48] realizaram um estudo com apenas três equipas. As equipas mencionaram pontos positivos, como o facto de terem um produto operacional desde uma fase inicial e uma melhor comunicação com os clientes. No entanto, também mencionaram desvantagens. Apesar de terem melhores relações com os clientes, referem que o facto de terem de os contactar frequentemente é um ponto negativo em termos de programação mais lenta.

4 ANÁLISE DA UNIDADE CURRICULAR DE GESTÃO DE PROJETOS DE SOFTWARE

Tendo toda a informação das IES e de algumas empresas que estabelecem contacto com o ISEC, é importante fazer uma análise comparativa com o atual plano desta UC. Esta análise é importante porque nos vai permitir identificar quais os processos que devem ser alterados, de forma a estarem mais alinhados com as expectativas do mercado de trabalho, mantendo os resultados de aprendizagem definidos.

4.1 Gestão de Projetos de Software no Instituto Superior de Engenharia de Coimbra

A UC de GPS é uma UC lecionada no quinto semestre da LEI, mesmo antes dos estudantes irem para estágio ou projeto no sexto semestre. Ao longo do semestre, os estudantes têm aulas teóricas e práticas.

Nas aulas teóricas, os estudantes adquirem conceitos e conhecimentos sobre boas práticas de engenharia, gestão de projetos de software e processos e metodologias de desenvolvimento de software.

Na componente prática, os estudantes desenvolvem um projeto para construírem um pequeno produto de software. O objetivo deste projeto é capacitar os estudantes para planear, executar e controlar um projeto de software simples em equipa (idealmente grupos de cinco elementos), utilizando práticas de engenharia adequadas e entregá-lo dentro do custo, prazo, qualidade e expectativas previstos. Como o principal foco desta UC é a gestão de projetos de software, não se pretende que o produto a ser desenvolvido seja tecnicamente complexo. Assim, a escolha do tema é feita pela equipa, sujeito à aprovação do professor e deve ser simples o suficiente para que uma única pessoa consiga programá-lo em alguns dias. Este trabalho deve ser um trabalho original. As aulas práticas visam o acompanhamento semanal de cada grupo, sob a forma de reuniões de progresso com o professor.

Até ao ano letivo 2022/2023, os estudantes utilizaram o Waterfall como metodologia de desenvolvimento de software, para gerir todo o projeto.

Conforme mostrado em 2.2.1, o Waterfall é uma metodologia rigorosa, com todos os seus processos bem detalhados e, por isso, existia um processo bem definido que os estudantes tinham de seguir.

4.1.1 Ciclo de vida do projeto

Na Figura 16 é possível visualizar o ciclo de vida do projeto. Existiam seis fases, agrupadas em três etapas.

As três etapas eram:

1. **Preparação do projeto** – esta etapa tinha como principal objetivo a construção das equipas, a escolha de um tema para a realização do projeto e o planeamento do mesmo;
2. **Execução do projeto** – esta etapa tinha como principal objetivo a especificação de requisitos, o desenvolvimento do produto e garantir que eram atingidos os níveis de qualidade previamente planeados;
3. **Conclusão do projeto** – esta etapa tinha como principal objetivo que os estudantes explicassem, de forma breve e crítica, como planearam, geriram e implementaram todo o projeto.

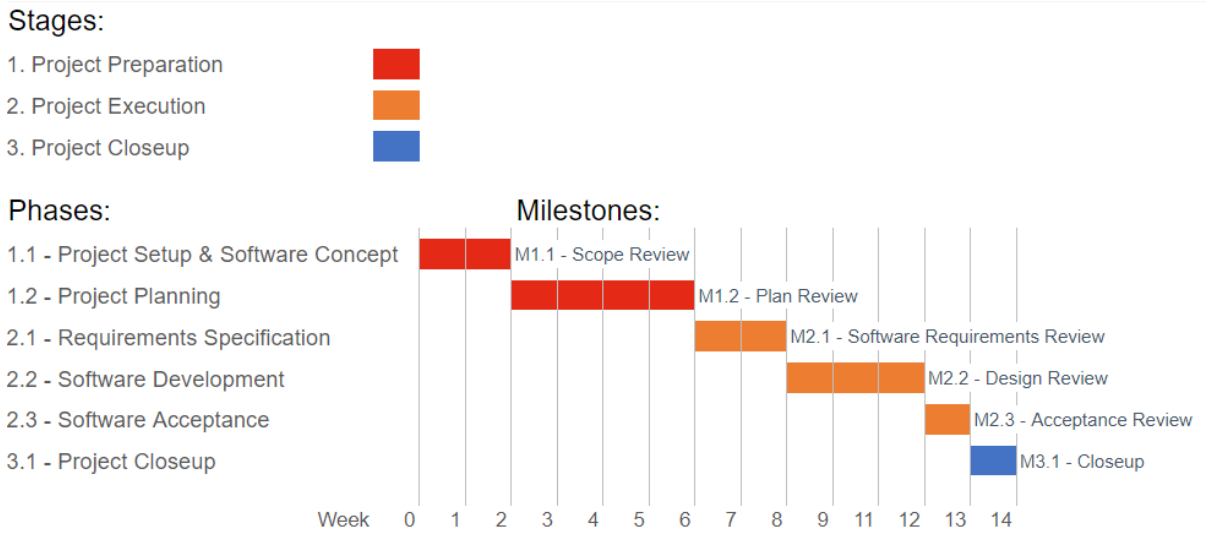


Figura 16 - Ciclo de vida do projeto (Anexo A)

Para cada uma das etapas existiam algumas fases, sendo que deviam criar alguns documentos no decorrer das mesmas. No final de cada uma das fases, existia uma *Milestone*, que resultava na entrega de, pelo menos, um documento.

Relativamente à primeira etapa, preparação do projeto, existiam duas fases:

1.1. **Configuração do projeto e conceito do software** – esta fase tinha como objetivos (1) a criação da equipa, definição do tema e criação do ambiente de desenvolvimento e (2) identificação do problema, da visão e do âmbito da solução;

1.2. **Planeamento do projeto** – os objetivos desta fase eram (1) fazer um planeamento de desenvolvimento do projeto, identificando o ciclo de vida, os documentos que seriam entregues ao cliente, como seria feita a gestão relativamente a estimativas, planeamento e controlo do trabalho e processos

técnicos e (2) fazer um plano de qualidade, identificando quais eram os objetivos que pretendiam alcançar, que revisões iam fazer, quais os tipos de testes que iam implementar, como fariam a gestão de riscos de alterações e que *standards* de código iam utilizar.

No que diz respeito à segunda etapa, execução do projeto, existiam três fases:

2.1. Especificação dos requisitos – os objetivos desta fase passavam por (1) especificar os requisitos, identificando requisitos funcionais, requisitos não funcionais, *mockups* e casos de uso, (2) identificar os riscos e eventuais ações de mitigação e (3) a criar um plano de testes de aceitação, identificando que requisitos seriam ou não testados, casos de teste, abordagens, entre outros aspetos;

2.2. Desenvolvimento do software – esta fase tinha como objetivos (1) documentação da arquitetura e *design* do software e (2) implementação e testes unitários do mesmo;

2.3. Aceitação do software – os objetivos desta fase eram (1) a criação de um relatório de testes de aceitação, identificando os resultados e incidentes encontrados e (2) a elaboração de um relatório de avaliação de qualidade, passando por todos pontos que foram identificados no objetivo 2 do ponto 1.2;

Para terminar, em relação à terceira e última etapa, conclusão do projeto, existia apenas uma fase:

3.1. Conclusão do projeto – esta fase tinha como objetivo a apresentação de como foi o planeamento, gestão e implementação de todo o projeto.

Quanto aos documentos que deviam ser elaborados em cada uma das etapas/fases, estes podem ser vistos na Figura 17.

Documents				Milestones					
Stage	Phase	ID	Name	M1.1	M1.2	M2.1	M2.2	M2.3	M3.1
1	1.1	D1.1.1	Vision & Scope (VS)	X					
		D1.1.2	Milestone M1.1 Report (M11)	X					
	1.2	D1.2.1	Software Development Plan (SDP)		X				
		D1.2.2	Quality Assurance Plan (QAP)		X				
		D1.2.3	Milestone M1.2 Report (M12)		X				
2	2.1	D2.1.1	Software Requirements Specification (SRS)			X			
		D2.1.2	Risk Plan (RP)			X			
		D2.1.3	Acceptance Test Plan (ATP)			X			
		D2.1.4	Milestone M2.1 Report (M21)			X			
	2.2	D2.2.1	Software Architecture & Design (SAD)				X		
		D2.2.2	Source code				X	X	
		D2.2.3	Milestone M2.3 Report (M22)				X		
	2.3	D2.3.1	Acceptance Test Report (ATR)		X			X	
		D2.3.2	Quality Assessment Report (QAR)		X	X	X	X	
		D2.3.3	Milestone M2.3 Report (M23)					X	

Figura 17 - Documentos a elaborar ao longo do projeto

Todos os documentos, exceto o código-fonte, eram escritos em Word, com base em *templates* fornecidos pelo professor, e estão em Anexo (Anexo B ao Anexo K)

Existia um versionamento de documentação. Todos os documentos deviam ser publicados no *Sharepoint*, após serem aprovados.

Na Figura 18 é possível ver o ciclo de vida de um documento. No momento de criação do documento, este ficava com a versão 0.1. Ficava no estado *draft* até estar pronto para revisão e a sua versão era incrementada em 0.1 sempre que houvesse alterações.

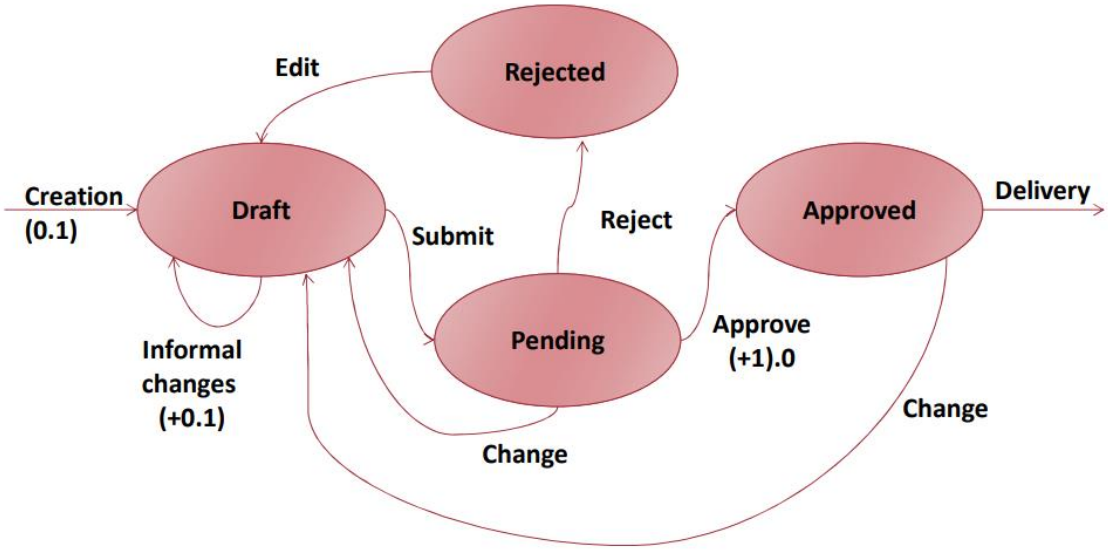


Figura 18 - Ciclo de vida de um documento

Assim que o documento era submetido, este ficava no estado pendente. Existiam três resultados possíveis:

- **Rejeitado** (correspondente ao *reject*) – ao ficar rejeitado, a equipa tinha de refazer o documento e tinha de ser submetido novamente ao cliente;
- **Aprovado condicionalmente** (correspondente ao *change*) – ao ficar aprovado condicionalmente, a equipa devia fazer as correções de baixa importância identificadas pelo cliente e ficava aprovado sem necessidade de nova submissão. A partir desse momento, o documento incrementava a versão para 1.0*;*
- **Aprovado** (correspondente ao *approve*) – ao ficar aprovado, a equipa podia publicar o documento de imediato. Os documentos eram aprovados na *review meeting* da *milestone*.

4.1.2 Ferramentas e tecnologias de trabalho

Durante a realização do projeto de software, os estudantes deviam utilizar um conjunto de ferramentas, nomeadamente:

- Microsoft Teams para comunicação;
- SharePoint para gestão de documentação;
- Git para versionamento de código;
- Word e Excel para elaboração de documentação;
- JUnit para realização de testes unitários.

Adicionalmente, devido à estruturação do plano curricular da licenciatura, era altamente recomendada a utilização de Java como linguagem de programação.

4.1.3 Esforço

Dado que a UC de GPS tem seis ECTS, é expectável que os estudantes tenham um esforço semanal de cerca de nove horas (das quais cinco são utilizadas nas aulas). Desta forma, cada estudante deve dedicar um esforço semanal de quatro horas ao projeto, fora de aula. Nessas quatro horas, não estão incluídas as reuniões semanais com o professor, uma vez que se realizam em contexto de aula. Este esforço pode ser maior ou menor durante as fases um e três, no entanto, os estudantes devem dedicar exatamente quatro horas semanais durante a fase dois (execução do projeto). Isto pode acontecer visto que numa fase inicial os estudantes podem não ter trabalho suficiente para utilizarem as quatro horas.

Os estudantes deviam registar o esforço semanal individual e coletivo num Excel, que seguia um *template* fornecido pelo professor (Anexo L). Nesse *template*, os estudantes deviam escrever uma pequena descrição do trabalho realizado durante a semana.

Existia bastante rigor no esforço semanal que os estudantes deviam ter, para que os estudantes tivessem uma noção/perceção de como é que funciona o desenvolvimento de software na indústria.

4.1.4 Dashboard

Todas as equipas deviam manter uma *dashboard*[15] atualizada. A *dashboard* devia conter:

- Informações do projeto – nome do projeto, declaração de objetivo e membros da equipa;
- Plano base, com o ciclo de vida do projeto;
- Registos de esforço individual e coletivo;
- Estado atual do projeto;
- Links para documentos e repositórios de código.

Esta *dashboard* devia estar interligada com o grupo da equipa no Teams. Na Figura 19 é possível vermos um exemplo de uma *dashboard.*

4.1.5 *Milestones*

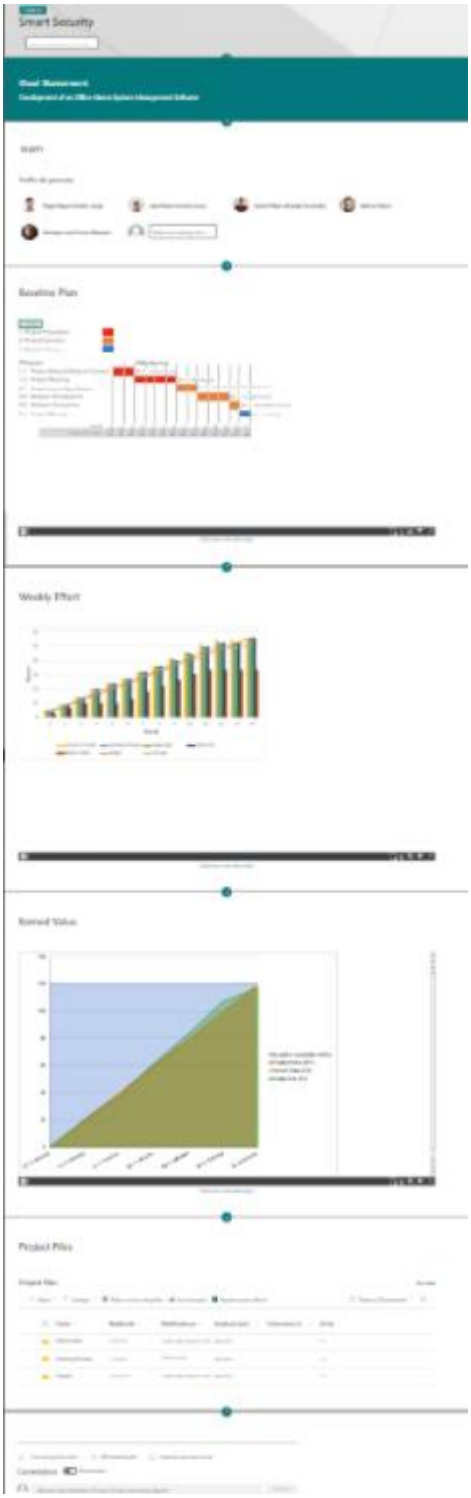
Uma *Milestone* é um marco do projeto, que marca o término de uma fase ou etapa do mesmo. Por norma, existem quando uma parte importante do trabalho foi concluída. As *Milestones* ajudam na organização e gestão do projeto, sendo uma forma eficaz de medir o progresso.

Assim, os estudantes deviam entregar um relatório das *Milestones,* onde tinham de descrever sucintamente qual foi o progresso do projeto no decorrer da fase correspondente. Na Figura 20 é possível ver o fluxo de uma revisão de um relatório de uma *Milestone*, desde a sua preparação à publicação.

A equipa começava por preparar os artefactos. Posteriormente, reunia com o cliente, de forma a obter *feedback* do mesmo. Resultante desse *feedback*, o relatório da *Milestone* ficava sujeito a aprovação do cliente.

O *template* do relatório da *Milestone* pode ser visto no Anexo K.

[15] *Dashboard* –é um painel com informação sob a forma de gráficos, que permite retirar algumas informações relativamente à equipa de uma forma muito mais rápida, como por exemplo métricas de desenvolvimento.



Nome do projeto

Elementos da equipa

Ciclo de vida do projeto

Esforço semanal dos alunos

Gráfico com o estado atual do projeto

Links para a documentação

Figura 19 - Exemplo de *dashboard* a utilizar pelos estudantes

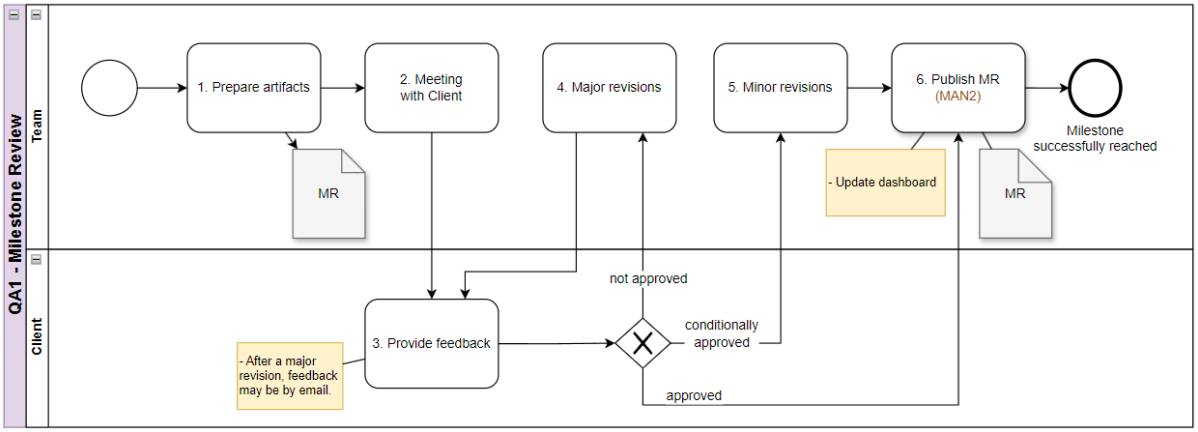


Figura 20 - *Workflow* das *Milestones*

4.1.6 Qualidade

Relativamente à qualidade, existia um controlo rigoroso tanto a nível documental como a nível de código. Os estudantes deviam escrever um documento denominado Plano de Garantia de Qualidade (QAP – *Quality Assurance Plan*) que possui a seguinte estrutura:

- **Objetivos de qualidade** – definir um objetivo externo e um objetivo interno;
- **Revisões** – especificar como serão feitas as diferentes revisões a nível documental, incluindo inspeções, *walkthroughs* e *deskchecks*;
- **Testes** – detalhar o desenvolvimento dos vários tipos de teste, como testes unitários, testes de aceitação e testes de integração;
- **Análise de risco** – explicar a abordagem para a análise de risco. Este processo resulta num documento, chamado Plano de Risco, que especifica os riscos do projeto, as probabilidades de ocorrência e o impacto no projeto caso aconteçam. As classificações são feitas numa escala de 1 a 5. Se a multiplicação da probabilidade pelo impacto for superior a 15, os estudantes devem definir ações de mitigação para esses riscos;
- ***Standards* de código** – especificar os standards de código a serem utilizados;
- **Métricas** – identificar as métricas que serão usadas para verificar se os objetivos de qualidade definidos no início do documento foram cumpridos.

O *template* do documento QAP pode ser visto no Anexo D.

4.1.7 Roles

Cada estudante devia assumir, pelo menos, uma das seguintes funções:

- **Gestor de projeto** – era o líder da equipa e era responsável por controlar o projeto;
- **Gestor de qualidade** – era responsável pela construção e manutenção do QAP e monitorizar a qualidade do projeto;
- **Gestor de risco** – era responsável por identificar e acompanhar os riscos;
- **Gestor técnico** – era responsável por tratar as questões mais técnicas;
- **Gestor de clientes** – era responsável por interagir com o cliente;
- **Gestor de testes** – era responsável por criar o plano de testes;

Apesar de existirem estas funções, todos os estudantes deviam ajudar os colegas com as suas responsabilidades. Por exemplo, todos os estudantes deviam contribuir para a identificação e acompanhamento dos riscos, não devia ser apenas o gestor de risco. Todos os estudantes faziam parte da equipa de desenvolvimento.

4.1.8 Avaliação do projeto

Como já foi mencionado, todas as semanas eram realizadas reuniões na aula prática da UC com o professor para avaliar o progresso do projeto. Os grupos tinham de ir devidamente preparados para estas reuniões.

Os estudantes recebiam um *feedback* semanal e eram avaliados numa escala de um a dez. Estas avaliações eram públicas para todos os elementos que estavam inscritos na UC, criando assim algum nível de competitividade entre os estudantes. Na Figura 21 é possível ver um exemplo de uma avaliação semanal.

Alem disso, os estudantes deviam monitorar a evolução do projeto utilizando um gráfico de *Earned Value Analysis* (EVA) [49]. O EVA é um método que permite gerir de forma gradual a quantidade de trabalho planeada e efetivamente realizada. Na Figura 22 é apresentado um exemplo deste gráfico EVA.

Team	Comments	Grade
	Lab1, 27/9/2022	
11	Dashboard is OK, with minor issues. Vision and Scope is quite complete, but the scope is not clear. Team must pay attention with some potential technical problems. The team is showing the will to do a good job.	**8**
12	Dashboard OK Vision and Scope with some misalignment between problem and solution. Must test if proposed solution is technically viable.	**8**
13	Dashboard with some problems. V&S with unclear vision. Must work to clarify both vision and solution.	**7**
14	A Dashboard necessita de várias correções. V&S necessita clarificar o problema, assim como a solução.	**7**
15	Dashboard com alguns problemas, nomeadamente no Log. V&S com ideia pouco clara e amadurecida. É necessário trabalhar mais neste documento.	**6**
	Lab2, 27/9/2022	
21	Dashboard com vários problemas. É necessário mais atenção aos detalhes. Problema demasiado genérico e visão pouco focada.	**7**
22	Diversos problemas com Dashboard, nomeadamente falhas no Log Problema e visão pouco claros.	**7**
23	Many issues with Dashboard. V&S is not presenting clearly the problem and the solution. The team must pay attention to the details	**6**
24	Many problems with Dashboard (e.g. goal statement, log, etc.) V&S needs simplification and clarification	**6**
25	The team did not complete the Dashboard - many issues. Must work better on V&S	**6**

Figura 21 - Exemplo de uma avaliação semanal de duas turmas práticas

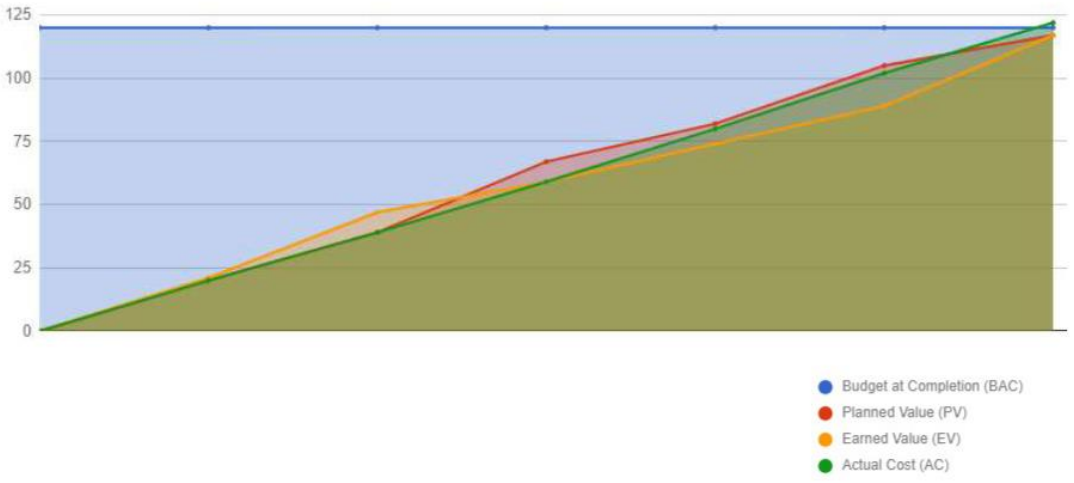


Figura 22 - Exemplo de um gráfico EVA

4.2 Objetivos de aprendizagem

Na secção 1.3.2 são apresentados os seis resultados de aprendizagem definidos para GPS-ISEC. Estes resultados de aprendizagem são alcançados através de objetivos, que os estudantes devem alcançar. Cada objetivo de aprendizagem deve, portanto, estar diretamente ligado a um ou mais desses resultados.

Na Tabela 2, é possível ver a relação entre os resultados de aprendizagem e os objetivos de aprendizagem de GPS do ano letivo 2022/2023.

Tabela 2 - Relação resultados de aprendizagem com os objetivos de aprendizagem de GPS do ano letivo 2022/2023

Resultados de aprendizagem	Objetivos de aprendizagem
RA1 - Descrever o desenvolvimento de software como uma disciplina de engenharia, utilizando a terminologia apropriada	OA1 – Explicar os principais termos e conceitos relacionados com a engenharia de software, tais como análise de requisitos, padrões de design e garantia de qualidade. OA2 – Comunicar eficazmente sobre tópicos de desenvolvimento de software, usando terminologia precisa e exata.
RA2 - Realizar um projeto de software em equipa, seguindo processos definidos e boas práticas de engenharia de software	OA3 – Interpretar e seguir as etapas dos processos de software numa metodologia Waterfall. OA4 – Demonstrar, através do envolvimento numa equipa, os elementos fundamentais da criação e gestão de equipas.
RA3 - Planear um projeto de desenvolvimento de software utilizando estimativas, calendarização de tarefas e alocação de recursos	OA5 – Criar um SDP, um WBS e um plano de qualidade. OA6 – Atribuir recursos e estimar o esforço de desenvolvimento de software. OA7 – Extrair tarefas dos requisitos funcionais. OA8 – Criar um plano EVA.
RA4 - Aplicar as melhores práticas de gestão da qualidade, gestão de riscos, gestão de equipas e gestão de *stakeholders*	OA9 – Definir um plano de qualidade e um relatório de avaliação. OA10 – Rever documentos, utilizando inspeções, *walktroughs* e *deskchecks*. OA11 – Definir e seguir um plano de risco. OA12 – Planear testes de software ao nível do utilizador, da integração e unitários e executar os testes.
RA5 - Gerir um projeto de software, monitorizando e controlando o seu progresso, garantindo a entrega do software dentro dos prazos e custos planeados e gerindo as expectativas dos *stakeholders*	OA13 – Gerir e controlar o progresso da execução do software utilizando o EVA. OA14 – Efetuar a análise das etapas para gerir os progressos e as expectativas das partes interessadas.

RA6 - Demonstrar conhecimento dos processos e metodologias de desenvolvimento de software, bem como dos princípios e fundamentos da melhoria de processos de software	OA15 – Interpretar as diferentes etapas e fases de um projeto seguindo o Waterfall OA16 – Compreender todos os resultados de um projeto OA17 – Melhorar os processos de um projeto em função de vários aspetos, como a equipa, o cliente, o âmbito, etc.

4.3 Comparação com outras instituições de ensino superior

Neste subcapítulo vai ser feita uma comparação das diversas UC que participaram no estudo, tanto a nível de objetivos e competências a serem adquiridas, como a nível de ferramentas e processos de trabalho.

4.3.1 Competências

Na Tabela 3, é apresentada uma comparação dos resultados de aprendizagem, analisando se as IES participantes no estudo possuem resultados de aprendizagem semelhantes aos de GPS, que estão descritos na secção 4.2, ou se de alguma forma procuram incutir esses resultados na realização dos trabalhos práticos das suas UC.

Tabela 3 - Comparação de resultados de aprendizagem

	RA-1	RA-2	RA-3	RA-4	RA-5	RA-6
ES–FEUP	✓	✓	✓	±	✓	✓
ES–FCTUC	✓	✓	✓	✓	✓	✓
ES–IST	✓	✓	✓	±	✓	✓
PSI–FCUL	✓	✓	±	±	✓	✓
TAES–ESTG	✓	✓	±	✓	✓	✓

± – não aborda todos os pontos descritos no resultado de aprendizagem

Assim, é possível verificar que apenas a Universidade de Coimbra tenta abordar todos os resultados de aprendizagem previamente definidos.

No Anexo M, é possível ver em detalhe os resultados de aprendizagem definidos para as UC de gestão de projetos de software de cada uma das IES.

4.3.2 Processos de trabalho

Na secção 3.2, foi realizada uma análise detalhada das metodologias de desenvolvimento de software utilizadas no decorrer dos trabalhos práticos das UC lecionadas nas diferentes IES. Para avaliar possíveis pontos positivos e negativos de cada uma das IES, realizámos uma análise comparativa. Na Tabela 4, apresentamos de forma sucinta toda a informação discutida na secção 3.2.

Tabela 4 - Comparação sucinta dos processos de trabalho de outras IES

UC	Dimensões da equipa	Metodologia	Gestão de esforço	Critérios de aceitação	Gestão de riscos	Regras de Git
GPS-ISEC	Cinco elementos	Waterfall	Estimativas para 4h semanais	Testes unitários, de aceitação e de integração	Criação de um plano de risco	N/A
ES–FEUP	Seis elementos	Scrum	Estimativas para 4h semanais	Testes de aceitação (manuais e automatizados)	N/A	- *Git workflow (feature based)* - *Merges requests* tinham de ser aceites por dois elementos
ES–FCTUC	Turma inteira com minigrupos de cinco elementos	Scrum (Scrum of Scrums)	Estimativas para 4h semanais	Testes unitários ou testes de aceitação (bónus para testes automatizados)	Requisitos com mais risco são feitas logo no início do projeto	- Utilização da funcionalidade GitLab ci/cd
ES–IST	Seis elementos	Scrum (Scrum of Scrums adaptado)	Estimativas por *story points* (20 SP por *sprint*)	Testes de carga e testes *end-to-end*	N/A	- *Git workflow (feature based)* - *Merges requests* tinham de ser aceite por dois estudantes
PSI–FCUL	Seis elementos	Scrum	N/A	Obrigatoriedade de testes, no entanto existe flexibilidade no tipo de teste	N/A	N/A
TAES–ESTG	Seis elementos	Agile (Scrum, Kanban e XP)	N/A	Testes de aceitação (manuais e automatizados)	N/A	- *Smart commit* - *Git Workflow (feature based)* - Dois repositórios (um para testes, outro para código) - *Merges requests*

4.3.3 Ferramentas

Relativamente às ferramentas, a Tabela 5 apresenta uma comparação das ferramentas utilizadas ao longo do projeto em todas as IES analisadas.

Tabela 5 - Ferramentas para gestão de projeto usadas nas diferentes UC

	Jira	Git	Teams	Figma	Discord	SharePoint
GPS-ISEC		✓	✓			✓
ES–FEUP		✓		✓		
ES–FCTUC		✓				
ES–IST		✓				
PSI–FCUL	✓	✓			✓	
TAES–ESTG	✓	✓ *				

* apenas recomendado

4.4 Análise das aulas práticas de Gestão de Projetos de Software

Durante o ano letivo 2022/2023 participei nas reuniões semanais de uma turma prática, como observador. Esta participação tinha como principal objetivo a recolha de informação, como por exemplo identificar as dificuldades sentidas pelos estudantes, pontos positivos e pontos menos positivos da UC, processos que estavam a ser bem assimilados ou processos que necessitavam de alguns ajustes, entre outros. Devido a esses acompanhamentos semanais e a algumas conversas que tive com os estudantes no final do ano letivo, foi possível concluir que:

- Os processos claramente definidos e bem documentados foram mais fáceis de compreender pelos estudantes;
- Os estudantes enfrentaram mais dificuldades quando os modelos de documentos fornecidos continham menos informações;
- Houve alguns desafios na utilização das várias ferramentas, como a configuração da integração entre o Teams e o Sharepoint, Git, entre outras, o que atrasou o projeto de certa forma;
- A falta de experiência em estimativas resultou na incapacidade da maioria dos grupos em cumprir todas as entregas previstas com o esforço disponível, resultando na não entrega de que algumas funcionalidades ou exigiram esforço adicional. Embora seja uma questão natural devido à inexperiência dos estudantes, o facto de passarem por este processo apenas uma vez ao longo do semestre pode representar uma desvantagem.

4.5 Discussão

Existem várias áreas dentro da gestão de projetos de software onde outras IES parecem estar mais bem preparadas quando comparadas com o ISEC. O facto de outras IES já possuírem regras mais avançadas relativamente ao Git e muitas delas já abordarem testes automatizados desde o início do plano de estudos, faz com que os estudantes adquiram mais competências e estejam mais alinhados com o que é procurado na indústria.

No entanto, o ISEC é uma das IES que aplica práticas mais avançadas no que diz respeito à gestão de esforço e à gestão de riscos, sendo que muitas delas nem têm processos definidos para estas áreas. Já na gestão de equipa, todas as IES parecem estar no mesmo nível.

Apesar disso, é evidente que a maioria das empresas influentes, tanto a nível regional como nacional, sente a necessidade de receber estudantes estagiários provenientes do ISEC mais bem preparados no que diz respeito a metodologias ágeis. Também se verificou que todas as IES participantes no estudo já abordam metodologias ágeis.

Assim, visando alinhar os conteúdos lecionados tanto com as exigências da indústria quanto com as abordagens adotadas por outras IES, decidiu-se avançar para a utilização de uma metodologia ágil em GPS no ano letivo 2023/2024, mais concretamente o Scrum. Pretendeu-se também adotar regas mais avançadas de Git, incluir algumas práticas de DevOps, entre outras.

De uma forma bastante resumida, decidiu-se manter, entre outros:

- Gestão da equipa – grupos de cinco elementos;
- Gestão do esforço – os estudantes devem continuar a ter um esforço semanal de quatro horas;
- Gestão de risco – sendo este um ponto positivo, pretende-se que os alunos continuem a fazer uma boa gestão de risco do projeto;
- Reuniões semanais – as aulas práticas vão continuar a ter o mesmo propósito, de acompanhamento e avaliação do trabalho de grupo;

No entanto, decidiu-se alterar fundamentalmente o seguinte:

- A metodologia – foi escolhido o Scrum, logo, todo o ciclo de vida alterou;
- Documentação – visto que se adotou uma metodologia ágil, o nível de documentação não é o mesmo que uma metodologia tradicional. Desta forma, os estudantes concentraram toda a documentação num documento único;
- Regras claras de Git – era uma área onde não havia grandes processos. Sendo um aspeto menos positivo, foram criadas várias regras de Git;
- Gestão de qualidade – passaram a utilizar-se *pipelines* que auxiliaram os alunos na gestão de qualidade do código;

5 RESTRUTURAÇÃO DA UNIDADE CURRICULAR

Neste capítulo vamos apresentar as alterações aos processos a utilizar pelos estudantes no trabalho prático da UC no ano letivo 2023/2024. Daqui em diante, de modo a facilitar a distinção entre os dois anos letivos, vai ser utilizada a terminologia "ano 1" para referências ao ano letivo 2022/2023, onde os estudantes seguiam uma metodologia tradicional, e ano 2 para referências ao ano letivo 2023/2024, onde os estudantes seguiam uma metodologia ágil.

Como já foi referido anteriormente, optámos por adotar uma metodologia ágil, nomeadamente o Scrum. No entanto, de forma a ir de encontro com os objetivos definidos já apresentados, selecionámos alguns processos e práticas de outras metodologias.

De todas as áreas de conhecimento apresentadas na secção 2.1, decidimos descartar algumas, visto não fazerem sentido para este contexto. Assim sendo, optámos por focar nas seguintes áreas e processos:

- **Gestão de integração**

  Na Tabela 6 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão de integração e uma correspondência entre os processos apresentados no PMBOK do PMI [16] e os processos criados para a UC de GPS.

Tabela 6 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão de integração

	Pre-game / Sprint 0	*Project execution*	*Sprint planning*	*Development*	*Sprint review*	*Sprint retrospective*
Gerir o trabalho	**X**		**X**	**X**	**X**	**X**
Monitorizar e controlar o trabalho do projeto	**X**		**X**	**X**	**X**	**X**
Efetuar um controlo integrado das alterações				**X**	**X**	
Encerrar o projeto		**X**				

- **Gestão do âmbito**:

  Na Tabela 7 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão do âmbito e uma correspondência entre os processos apresentados no PMBOK do PMI [16] e os processos criados para a UC de GPS.

Tabela 7 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão do âmbito

	Pre-game / Sprint 0	*Sprint planning*	*Sprint review*	*Backlog refinement*
Recolha de requisitos	**X**	**X**	**X**	**X**
Definir o âmbito	**X**			
Criar WBS	**X**	**X**	**X**	

- **Gestão do calendário**:

  Na Tabela 8 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão do calendário e uma correspondência entre os processos apresentados no PMBOK e os processos criados para a UC de GPS.

Tabela 8 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão do calendário

	Pre-game / Sprint 0	*Sprint planning*	*Sprint review*
Planear a gestão do calendário	**X**	**X**	**X**
Definir atividades	**X**	**X**	**X**
Sequência das atividades	**X**	**X**	**X**
Estimar a duração das atividades	**X**	**X**	
Elaborar o calendário	**X**	**X**	
Controlar o calendário	**X**	**X**	**X**

- **Gestão da qualidade**:

  Na Tabela 9 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão da qualidade e uma correspondência entre os processos apresentados no PMBOK e os processos criados para a UC de GPS.

Tabela 9 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão da qualidade

	Setup do projeto	*Pre-game / Sprint 0*	*Development*	*Sprint retrospective*	*Risk management*
Planear a gestão de qualidade	**X**	**X**	**X**	**X**	**X**
Gerir a qualidade		**X**	**X**	**X**	**X**
Controlar a qualidade		**X**	**X**	**X**	**X**

- **Gestão de riscos**:

  Na Tabela 10 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão do calendário e uma correspondência entre os processos apresentados no PMBOK e os processos criados para a UC de GPS.

Tabela 10 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão de riscos

	Pre-game / Sprint 0	*Sprint planning*	*Development*	*Risk management*
Planear de gestão de riscos	**X**			**X**
Identificar riscos	**X**	**X**	**X**	**X**
Efetuar uma análise qualitativa dos riscos				**X**
Efetuar uma análise quantitativa dos riscos				**X**
Planear as respostas aos riscos				**X**
Implementar as respostas aos riscos				**X**
Monitorar riscos				**X**

- **Gestão de *stakeholders***:

  Na Tabela 11 é possível visualizarmos o conjunto de processos escolhidos dentro da área de gestão do calendário e uma correspondência entre os processos apresentados no PMBOK e os processos criados para a UC de GPS.

Tabela 11 - Correspondência entre os processos do PMBOK e os processos criados para a UC de GPS na área de gestão de *stakeholders*

	Pre-game / Sprint 0	*Sprint review*
Identificar *stakeholders*	**X**	
Gerir a participação dos *stakeholders*	**X**	**X**

Relativamente a práticas de DevOps, quisemos aplicar algumas práticas apresentadas na secção 2.3.4, que achámos serem bastante úteis e que se conseguiam encaixar nos processos definidos, tendo em conta o trabalho prático em questão. Essas práticas são:

- *Version control*
- *Continuous integration*
- *Deployment automation / Continous Delivery*
- *Trunk-based development*
- *Continuous testing*
- *Architecture*
- *Code maintainability*
- *Working in small batches*

5.1 Ciclo de vida

Na Figura 23 é possível visualizarmos o novo ciclo de vida do projeto, que é constituído pelas mesmas três etapas anteriores: preparação, implementação e conclusão.

As três etapas visam o seguinte:

1. **Preparação do projeto** – à semelhança do ano 1, esta etapa também tem como objetivo a construção das equipas, a escolha de um tema para a realização do projeto e a preparação do mesmo. No entanto, nesta etapa os estudantes já começam a definir alguns requisitos;

2. **Execução do projeto** – esta etapa tem como principal objetivo o desenvolvimento do produto através da realização de três *sprints*. É importante salientar que a *sprint* 1 e a *sprint* 3 têm durações diferentes. Metade dos grupos têm a *sprint* 1 com a duração de três semanas e a *sprint* 3 com a duração de duas semanas, enquanto a outra metade dos grupos faz o inverso, tendo a *sprint* 1 com a duração de duas semanas e a *sprint* 3 com a duração de três semanas. Desta forma, é possível desfasar os grupos de trabalho, evitando que todos os grupos tenham as cerimónias na mesma aula;
3. **Conclusão do projeto** – também como no ano 1, esta etapa tem como principal objetivo que os estudantes expliquem, de forma breve, como planearam, geriram e implementaram todo o projeto.

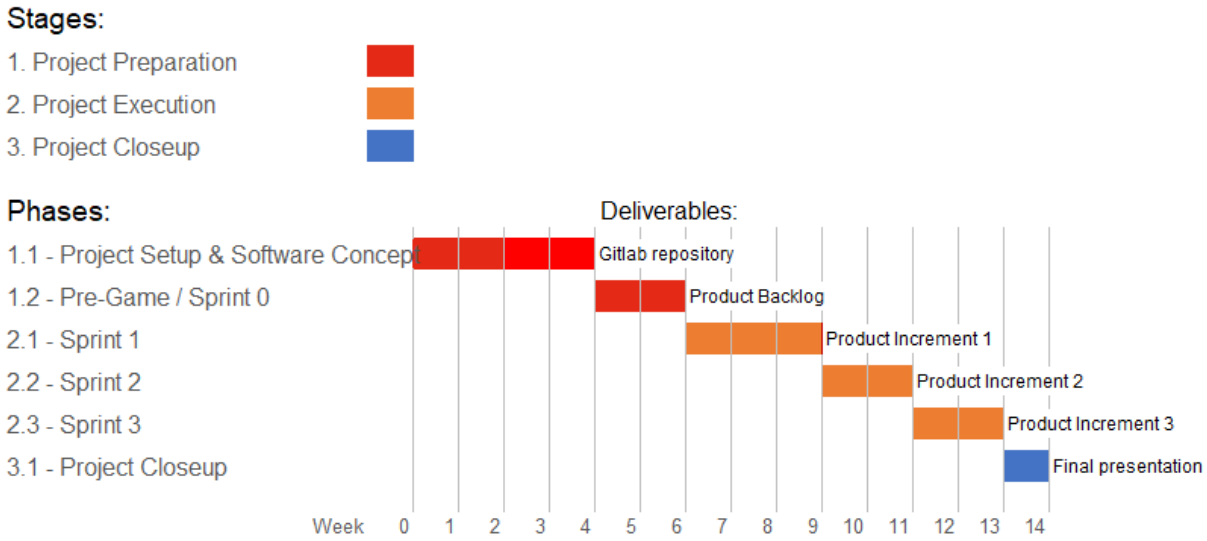


Figura 23 - Ciclo de vida do projeto utilizando Scrum

Relativamente à primeira etapa, preparação do projeto, existem duas fases:

1.1. **Configuração do projeto e conceção do software** – esta fase tem como objetivos (1) a criação da equipa e definição do tema, (2) aprenderem os princípios e práticas fundamentais das metodologias ágeis e do Scrum e (3) prepararem o ambiente de desenvolvimento, como por exemplo a criação do repositório GitLab [50], máquina virtual, entre outras;

1.2. **Pre–Game / *Sprint* 0** – esta fase tem como objetivos (1) escrita da visão e âmbito do produto, (2) criação de um *product backlog* (PB) com US estimadas e priorizadas, (3) desenhar uma possível arquitetura do produto, (4) criação de alguns *mockups* e (5) planear as *releases*.

No que diz respeito à segunda etapa, execução do projeto, existem três fases iguais:

2.1, 2.2, 2.3. *Sprint* – esta fase tem como objetivos a realização das *sprints*;

Para terminar, em relação à terceira e última etapa, conclusão do projeto, existe apenas uma fase:

3.1. Conclusão do projeto – esta fase tem como objetivo a apresentação de como foi o planeamento, gestão e implementação de todo o projeto.

Relativamente às *releases*, existem duas previstas. A *release* 1.0, correspondente à entrega do *Minimum Viable Product* (MVP), deve ser entregue no final da *sprint* 2. A *release* 2.0, com a versão final da aplicação, é entregue no final da *sprint* 3.

Todas as fases serão explicadas mais ao detalhe posteriormente.

5.2 Ferramentas de trabalho

Relativamente às ferramentas e tecnologias de trabalho, algumas são de utilização obrigatória e outras de utilização facultativa. As ferramentas de utilização obrigatória são:

- GitLab – será utilizada para gestão de código, gestão de documentação, gestão de projeto e integração contínua (CI);
- Java – linguagem de programação obrigatória;
- Teams – ferramenta de comunicação;
- SonarCloud – ferramenta de análise estática de código;
- Ubuntu/Ansible – é obrigatório que os estudantes tenham uma máquina virtual com Ansible instalado, para simular uma máquina de cliente e aplicar práticas de entrega contínua (CD);
- JUnit – ferramenta de desenvolvimento de testes unitários.

As ferramentas facultativas, que servem como auxílio, são:

- SceneBuilder – ferramenta que auxilia no desenvolvimento de interfaces gráficas em JavaFX, permitindo criar *mockups* e gerar automaticamente o código FXML necessário;
- ChatGPT e GitHub Copilot – estas ferramentas de inteligência artificial são permitidas, uma vez que o objetivo da unidade curricular é focar-se na gestão e não apenas na programação;

5.3 Setup do projeto

Na Figura 24 é possível visualizar a fase de configuração do projeto. Nesta fase, espera-se que os estudantes:

- Preparem um repositório na ferramenta GitLab;

- Configurem uma máquina virtual;

Os tutoriais destas configurações podem ser vistos no Anexo N e no Anexo O.

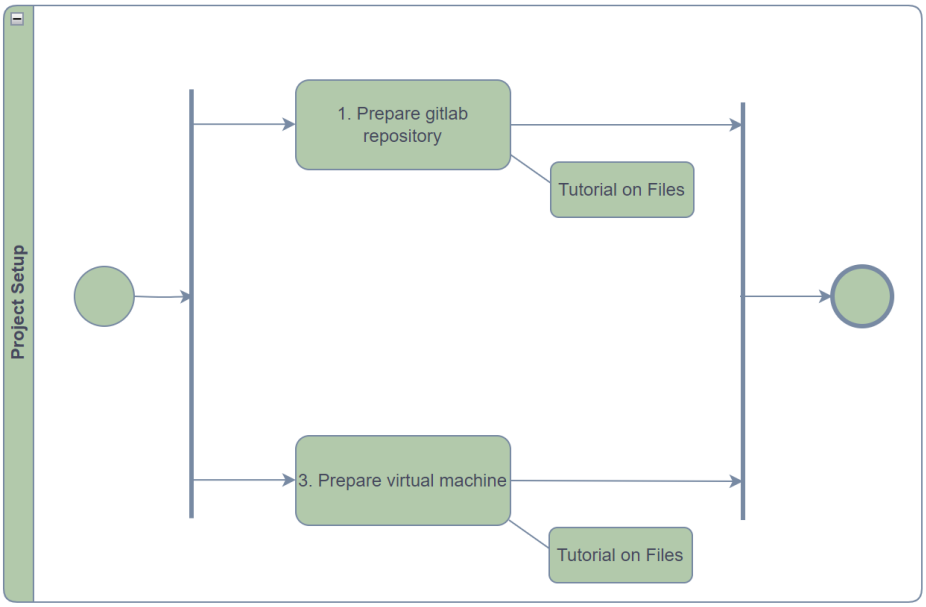


Figura 24 - Processo de configuração do projeto

5.3.1 GitLab

O GitLab servirá como ferramenta de gestão de projetos, versionamento de código, gestão de documentação e integração contínua (CI). Optámos por utilizar apenas o GitLab, visto que esta plataforma oferece todas as funcionalidades necessárias, reduzindo assim o número de ferramentas a utilizar pelos estudantes.

No que diz respeito às regras de Git, definimos algumas diretrizes:

- É obrigatório a utilização de, pelo menos, três *branches*: **main**, **dev** e **qa**;
- As *branches* de **qa** e **main** devem ser protegidas, não sendo permitido o *push*[16] direto para as mesmas;
- É obrigatório o uso de MR para a *branch* **qa**. Estes MR devem ser aprovados por, pelo menos, um estudante diferente do que efetuou o *commit* (ou seja, devem utilizar a prática *code review*);
- A partir da *sprint* 2, é obrigatória a utilização de *pipelines* CI/CD. Estas *pipelines* serão detalhadas posteriormente;

[16] Push – o comando "git push" é utilizado para atualizar o repositório remoto com as alterações realizadas no repositório local.

- Para a *branch* **main**, a utilização de MR também é obrigatória, mas os estudantes não são obrigados a fazer *code review*. O MR só é feito caso o cliente aceite os desenvolvimentos;

Na Figura 25, apresenta-se um exemplo do *Git workflow* pretendido.

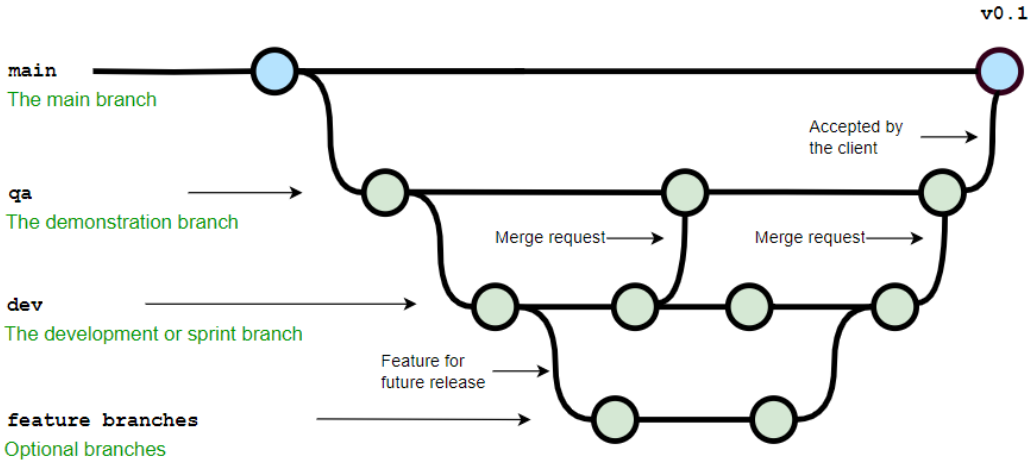


Figura 25 - *Git Workflow* pretendido

No que toca à documentação, os estudantes devem utilizar um *template* de READ.ME fornecido por nós, que deve ser adicionado ao repositório GitLab. Neste documento devem ser apresentadas algumas informações, nomeadamente a estrutura da equipa, visão e âmbito, plano de *releases*, incrementos das *sprints*, entre outras. Este *template* pode ser consultado no Anexo P.

5.3.1.1 GitLab Boards

É obrigatória a criação de quatro *boards*, sendo elas PB, *sprint* 1 *backlog*, *sprint* 2 *backlog* e *sprint* 3 *backlog*.

Na Tabela 12 é possível ver as colunas obrigatórias para a PB *board*, que tipos de cartões são permitidos, bem como estimativas e priorização (apenas válidos para US).

Já na Tabela 13, é possível ver as colunas obrigatórias das *sprint boards* (SB), que tipos de cartões são permitidos e as *labels* que estão relacionadas com as US (apenas válidas para tarefas e critérios de aceitação).

Todos os cartões devem ter uma estimativa de tempo, em horas, seguindo a sequência de Fibonacci. Fazem parte da sequência de Fibonacci os números 0, 1, 2, 3, 5, 8, 13, entre outros. Não devem ser utilizados números superiores a 13.

Tabela 12 - Informação necessária para construir uma PB

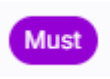
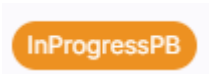
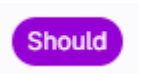
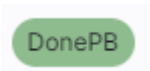
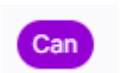
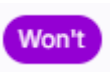
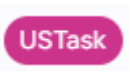
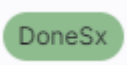
Colunas	Tipos de cartões	Estimativas (apenas para US)	Priorização (apenas para US)
Product backlog 	*User story <number>* 	*Large* 	*Must have* 
In progress 	*Configuration task* 	*Medium* 	*Should have* 
Done 	*Documentation task* 	*Small* 	*Can have* 
	Management task 		*Won't have* 
	Any fix (code, documentation or configuration) 		

Tabela 13 - Informações necessárias para construir uma SP

Colunas	Tipos de cartões	Relacionado com US (apenas para tarefas e AC)
Sprint backlog 	US *task* 	*User story* <número> 
In progress 	*Acceptance criteria* 	
Done 	*Configuration task* 	
	Documentation task 	
	Management task 	
	Any fix (code, documentation or configuration) 	

5.3.2 Pipelines CI/CD

Os estudantes têm de criar uma máquina virtual, Ubuntu, que vai servir para simularem uma máquina do cliente, onde executavam a *pipeline* CD. Desta forma, têm uma proximidade com processos de DevOps e conseguem perceber a necessidade de cada uma das *pipelines*.

No Anexo Q está presente o *template* da pipeline CI que foi fornecido aos estudantes.

A *pipeline* CI tem duas fases:

- *Test* – esta tem duas etapas:
    - *Unit-tests* – etapa responsável pela execução de testes unitários;

  - *Sonarcloud-check* – etapa responsável pela execução de testes de análise estática de código. No Anexo R é possível ver como é que deve ser feita a configuração nesta ferramenta;

- *Build* – esta tem apenas a fase de *build_and_deploy_image*. O objetivo desta etapa é construir e enviar uma imagem docker para o repositório DockerHub[17]. Ao fazer o *push* da imagem docker, são criadas também duas *labels*, uma com a referência da *branch* e *commit* efetuado e outra *latest*.

Relativamente à pipeline CD, o *template* da mesma está presente no Anexo S.

A *pipeline* CD tem várias tarefas, tais como:

- Instalação das bibliotecas e software necessários, como por exemplo o docker e xorg (serviço responsável por emular a aplicação que estava a ser executada no *container* docker);
- Iniciar/reiniciar alguns serviços importantes, nomeadamente os descritos no ponto anterior;
- Garantir permissões necessárias para executar a aplicação num *container* docker;
- Executar o container.

Depois de executar a *pipeline*, a aplicação fica disponível e é iniciada automaticamente.

No Anexo T está presente um tutorial de auxílio de criação da pipeline CI no repositório.

5.4 *Pre-Game / Sprint* 0

Na Figura 26 podemos visualizar a fase de *Pre–game / sprint* 0.

Os estudantes devem começar por definir uma visão e âmbito do produto. Esta não deve ser demasiado fechada, de modo que seja possível ir adicionando funcionalidades com o decorrer do projeto.

A nível de documentação, os estudantes devem documentar no READ.ME (Anexo P) do repositório do GitLab o seguinte:

- Documentar os requisitos, com recurso a US, diagramas de casos de uso e a protótipos;
- Desenhar um possível modelo de domínio;
- Documentar um plano de riscos;

[17] DockerHub – repositório para controlo de versões de imagens Docker

- Documentar o *Definition of Done*;
- Documentar o plano de *releases*.

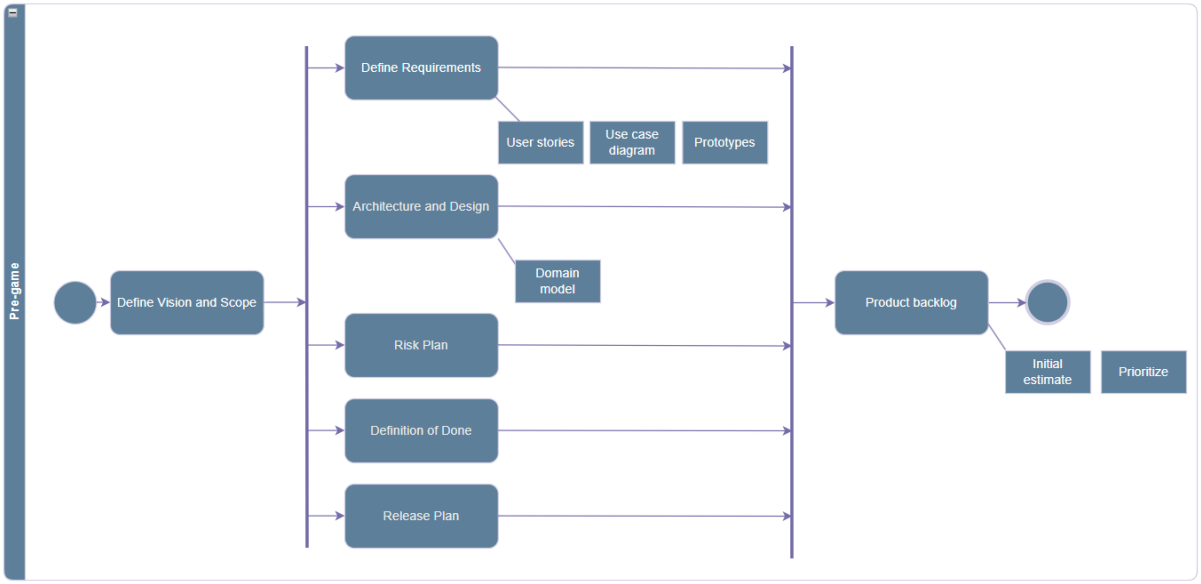


Figura 26 - Processo do *Pre-game* / *sprint* 0

Como resultado disto, deve ser criada a PB já descrita anteriormente, com as devidas colunas, cartões, uma estimativa inicial para cada uma das US e a devida prioridade.

5.5 *Project Execution*

A etapa da execução está dividida em três fases, cada uma correspondente a uma *sprint*. Na Figura 27 é possível visualizar o ciclo de vida de uma *sprint*. Estas são iniciadas com o seu planeamento (*sprint planning*), seguido do seu desenvolvimento (*development*) e é terminada com a sua revisão e retrospetiva (*sprint review* e *sprint retrospective*).

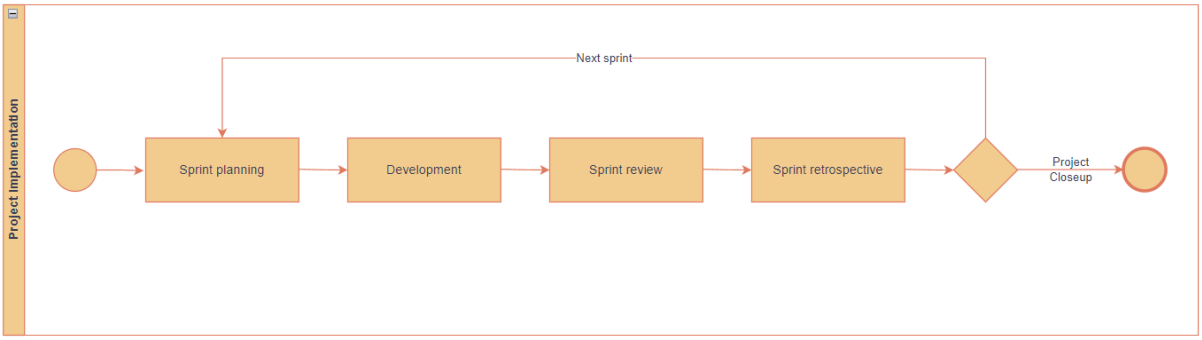


Figura 27 - Processo de uma *sprint*

5.5.1 *Sprint planning*

O processo de planeamento de uma *sprint* está presente na Figura 28.

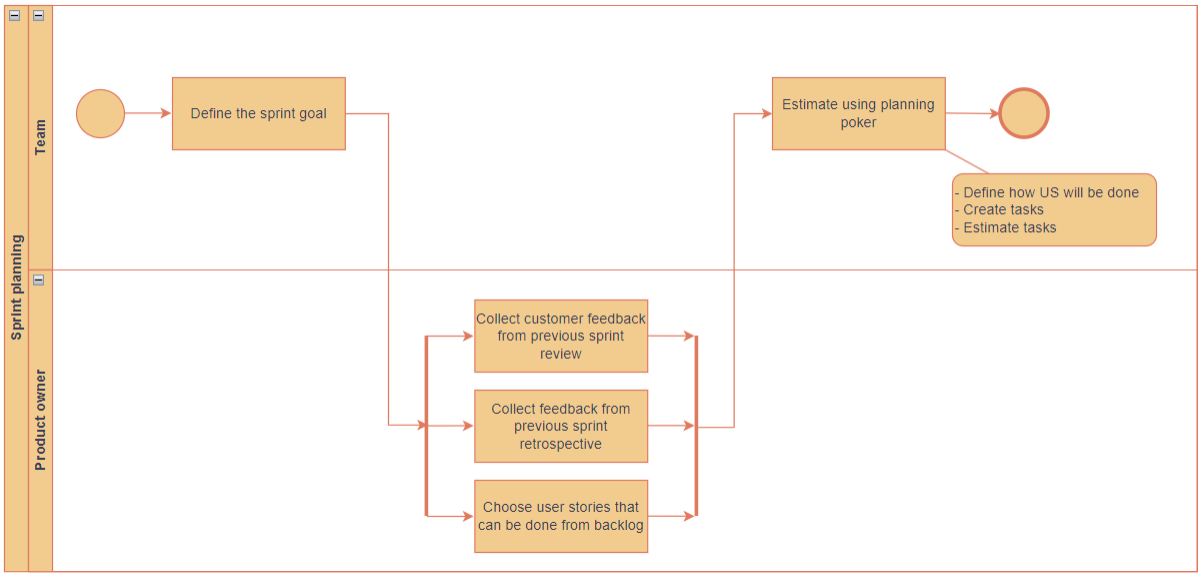


Figura 28 - Processo do planeamento de uma *sprint* (*sprint planning*)

A equipa começa por definir os objetivos da *sprint*. De seguida, o PO deve recolher algum *feedback* dado pelo cliente na *sprint* e *sprint review* anteriores e fazer uma pré-seleção de US que estão na PB *board*. Posteriormente, toda a equipa deve definir/discutir como serão feitas as US, criar as devidas tarefas na *board* da *sprint* a decorrer e estimar cada tarefa, em horas, usando *planning poker*[18].

5.5.2 Desenvolvimento

O processo de desenvolvimento de uma *sprint* pode ser visto na Figura 29.

A equipa começa por desenvolver as tarefas previamente planeadas. Terminada uma tarefa, devem criar e executar os testes unitários necessários manualmente, se aplicável. Caso estes falhem, devem fazer as devidas correções antes de enviarem o seu código para o repositório (voltar ao ponto *Developing tasks*). Caso passem, então devem fazer o *commit*, na *branch* **dev**. Ao fazerem o *commit*, a *pipeline* CI será acionada. Como já foi mencionado anteriormente, esta *pipeline* executa todos os testes unitários e analisa o código de forma automática.

[18] *Planning poker* – técnica utilizada para realização de estimativas em equipa.

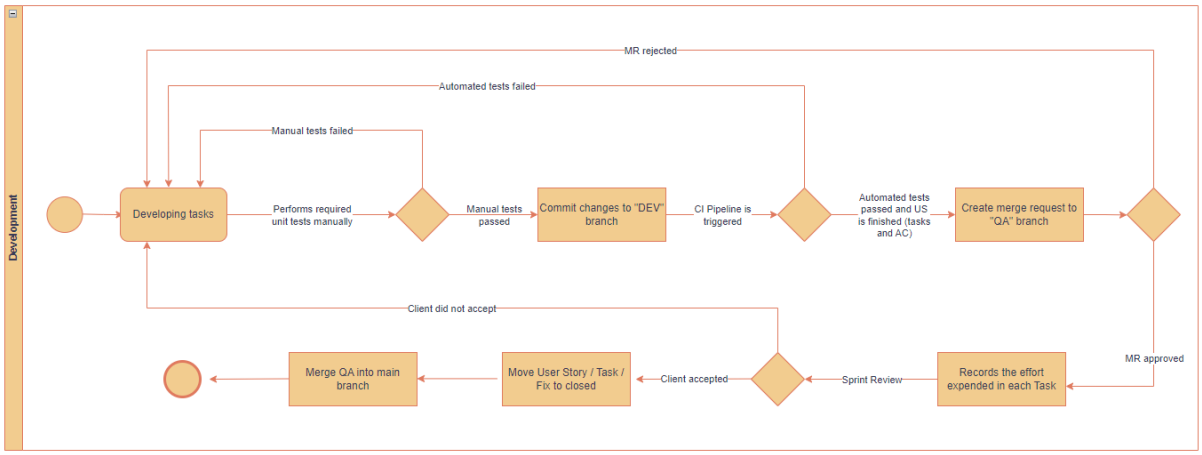


Figura 29 - Processo de desenvolvimento

Se os testes unitários ou a análise estática de código falharem, então os estudantes devem recuar ao ponto inicial do processo, fazer as devidas correções e refazer todo o processo até então. Caso passem, então abrem um MR para a *branch* **qa**. Este MR só deve ser feito caso a US esteja finalizada. Caso contrário, devem continuar com os desenvolvimentos até terminarem todas as tarefas da US em questão. Este MR tem de ser aceite por, pelo menos, um estudante diferente do que abriu o MR. Caso o MR seja reprovado, devem recuar novamente para o ponto inicial do processo, efetuar as devidas correções e refazer todo o processo até aqui. Se for aprovado, então devem registar o esforço da tarefa. Esta ação de registo de esforço pode e deve ser uma ação contínua, de forma a serem mais precisos.

Chegando à *sprint review*, o cliente pode ou não aceitar o software entregue. Caso não seja aceite, independentemente da razão, devem voltar ao ponto inicial do processo, fazer as devidas correções/alterações e refazer todo o processo. Caso seja aceite, então podem mover as tarefas para a coluna "*closed*" e abrir um MR para a *branch* **main**.

5.5.3 *Sprint review*

Na Figura 30 é possível vermos o processo da *sprint review*.

Os estudantes devem executar a *pipeline* CD manualmente e inserir alguns dados na aplicação de forma a fazer a demonstração. Paralelamente a isto, devem também documentar o incremento da *sprint* (*product increment*), fazendo uma comparação entre o que foi planeado e o que foi realmente entregue.

Posto isto, fazem a demonstração ao cliente, fecham as US que o cliente aceitar (movendo-as para a coluna "*closed*" na PB *board*) e reúnem as prioridades do cliente para poderem planear a próxima *sprint*.

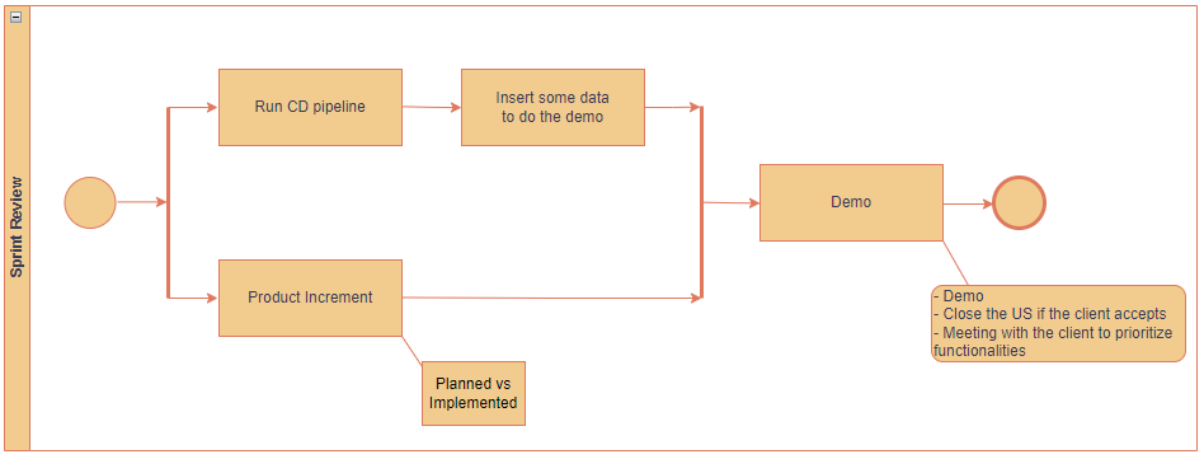


Figura 30 - Processo de uma *sprint review*

5.5.4 *Sprint retrospective*

Na Figura 31 está detalhado o processo da *sprint retrospective*.

Este processo é bastante simples, sendo que consiste em discutirem o que correu bem, menos bem e como podem melhorar para a próxima *sprint*.

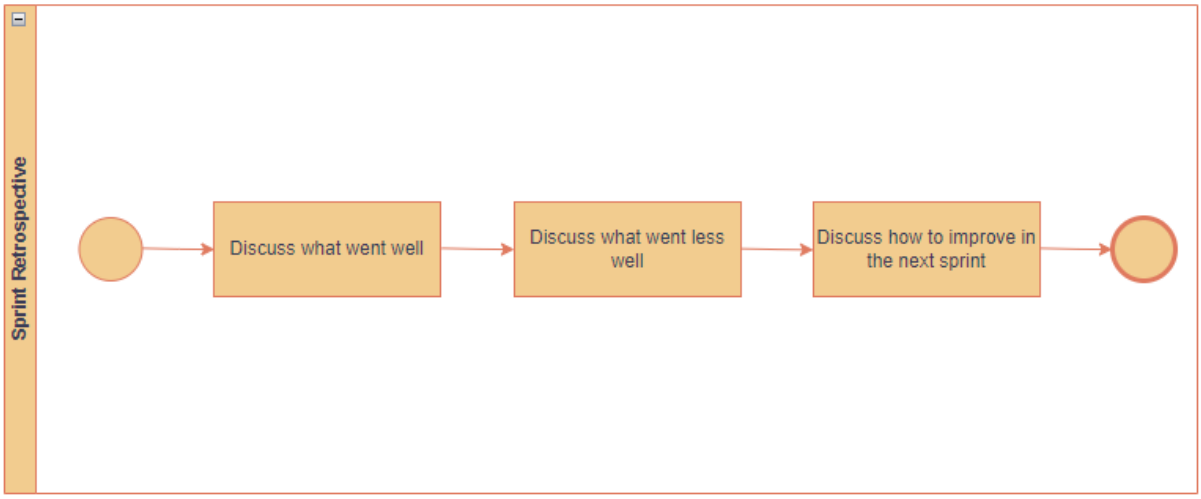


Figura 31 - Processo de uma s*print retrospective*

5.6 *Risk Management*

Como foi possível ver na secção 2.3, as metodologias ágeis não documentam a gestão de riscos, iniciando apenas pelas tarefas mais arriscadas. No entanto, e visto que o ISEC é uma das melhores IES no que diz respeito à gestão de riscos, optámos por incluir um processo de gestão de riscos, de forma a controlarem melhor o projeto. Este processo pode ser visto na Figura 32.

Figura 32 - Processo de gestão de risco (*risk management*)

Para além de iniciarem a desenvolver as US que considerem ser as mais arriscadas, devem identificar outros potenciais riscos ao projeto. Assim, devem documentar um plano de riscos, onde fazem uma análise a cada um desses riscos. À semelhança do processo de gestão de riscos utilizado até ao ano 1, classificam as probabilidades desses riscos acontecerem e qual é o seu impacto no projeto caso aconteçam. Essas classificações são feitas numa escala de 1-5 e caso a multiplicação da probabilidade com o impacto for superior a 15, os estudantes devem definir ações de mitigação para esses riscos.

Posto isto, devem fazer a monitorização destes riscos semanalmente, fazendo ajustes tanto às probabilidades como aos impactos, se necessário.

5.7 *Backlog refinement*

O processo de *backlog refinement* é um processo simples e pode ser visto na Figura 33.

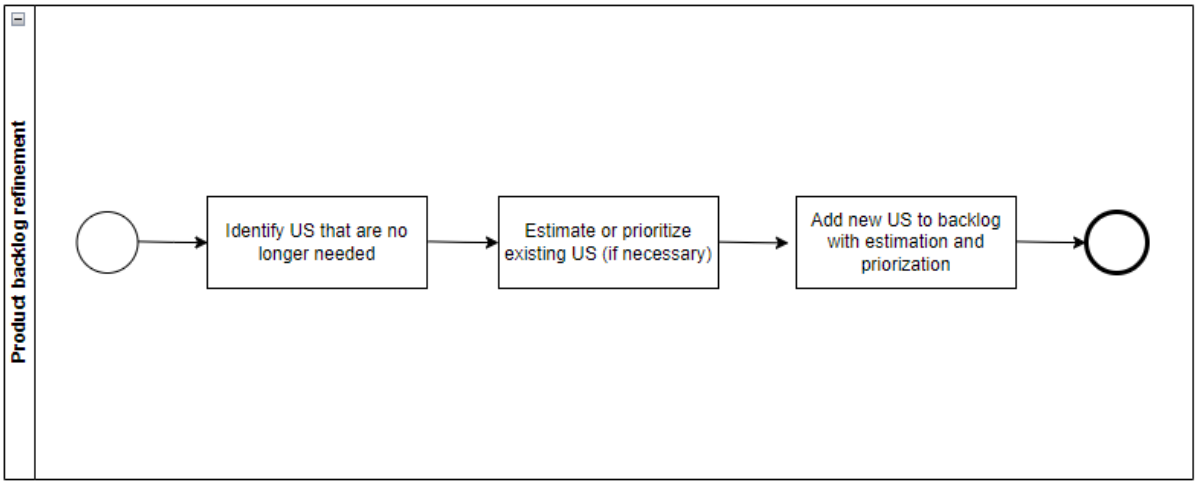


Figura 33 - Processo de *backlog refinement*

A meio da segunda *sprint*, os estudantes devem reunir-se e identificar que US é que necessitam de ajustes, remover as que já não são necessárias ou adicionar novas.

Estas novas US devem ser novas funcionalidades que não estavam planeadas no início do projeto e devem ser inseridas na PB já com uma estimativa e priorização.

5.8 *Project Closeup*

Para terminar, temos o término do projeto. Não existe nenhum processo para esta fase, visto que consiste apenas numa apresentação à restante turma. Os estudantes devem apresentar as seguintes informações:

- **Introdução** – apresentação de objetivos, visão e âmbito e apresentação dos elementos do grupo;
- **Produto** – planeado *vs* entregue, por *sprint*. Esta comparação deve ter em conta objetivos/incrementos e US;
- **Gestão do projeto** – ciclo de vida do projeto, *sprints*, *releases*, *budget* e esforço por *sprint* e por *release*, como foi feito o controlo do projeto, entre outros;
- **Plano de risco** – evolução do plano de risco, bem como se o *threshold of success*[19] foi atingido;
- **Qualidade** – testes unitários, testes de integração e testes de aceitação que foram ou não feitos;
- **DevOps** – Git Worfklow utilizado e *pipelines* CI/CD;
- **Aprendizagens** – devem identificar quais foram as aprendizagens que tiveram com o desenvolvimento do projeto de software;
- **Análise individual e conclusão** – cada elemento deve apresentar as principais contribuições durante o projeto, participações e mostrando a sua influência no projeto.

5.9 Objetivos de aprendizagem

Os objetivos de aprendizagem do ano 1 já foram apresentados na secção 4.2. No entanto, na Tabela 14 serão mostrados novamente de modo a facilitar a comparação entre os dois anos.

[19] *Threshold of success* – refere-se a um conjunto de critérios ou métricas que determinam quando um projeto, funcionalidade ou produto pode ser considerado bem-sucedido.

Tabela 14 - Relação resultados de aprendizagem com os objetivos de aprendizagem

Resultados de aprendizagem	Objetivos de aprendizagem (ano um)	Objetivos de aprendizagem (ano dois)
RA1 - Descrever o desenvolvimento de software como uma disciplina de engenharia, utilizando a terminologia apropriada	OA1 – Explicar os principais termos e conceitos relacionados com a engenharia de software, tais como análise de requisitos, padrões de design e garantia de qualidade. OA2 – Comunicar eficazmente sobre tópicos de desenvolvimento de software usando terminologia precisa e exata.	OA1 – Explicar os principais termos e conceitos relacionados com a engenharia de software, tais como análise de requisitos, padrões de design e garantia de qualidade. OA2 – Comunicar eficazmente sobre tópicos de desenvolvimento de software usando terminologia precisa e exata
RA2 - Realizar um projeto de software em equipa, seguindo processos definidos e boas práticas de engenharia de software	OA3 – Interpretar e seguir as etapas dos processos de software numa metodologia Waterfall. OA4 – Demonstrar, através do envolvimento numa equipa, os elementos fundamentais da criação e gestão de equipas.	OA3 – Identificar o âmbito de um projeto de software OA4 – Definir uma estratégia de *branching* OA5 – Organizar uma equipa
RA3 - Planear um projeto de desenvolvimento de software utilizando estimativas, calendarização de tarefas e alocação de recursos	OA5 – Criar um SDP, um WBS e um plano de qualidade. OA6 – Atribuir recursos e estimar o esforço de desenvolvimento de software. OA7 – Extrair tarefas dos requisitos funcionais. OA8 – Criar um plano EVA.	OA6 – Extrair requisitos de software, utilizando casos de uso e US OA7 – Estimar e priorizar às US OA8 – Definir as *releases* do projeto OA9 – Planear as sprints e as *releases*
RA4 - Aplicar as melhores práticas de gestão da qualidade, gestão de riscos, gestão de equipas e gestão de *stakeholders*	OA9 – Definir um plano de qualidade e um relatório de avaliação. OA10 – Rever documentos, utilizando inspeções, *walktroughs* e *deskchecks*. OA11 – Definir e seguir um plano de risco. OA12 – Planear testes de software ao nível do utilizador, da integração e unitários e executar os testes.	OA10 – Definir um plano de qualidade ao nível do produto e de código OA11 – Definir uma estratégia de revisão de código OA12 – Definir o plano de testes OA13 – Saber como analisar os resultados da execução de testes através da pipeline OA14 – Utilizar uma ferramenta de gestão de versões e de artefactos OA15 – Identificar e definir um plano de riscos

RA5 - Gerir um projeto de software, monitorizando e controlando o seu progresso, garantindo a entrega do software dentro dos prazos e custos planeados e gerindo as expectativas dos *stakeholders*	OA13 – Gerir e controlar o progresso da execução do software utilizando o EVA. OA14 – Efetuar a análise das etapas para gerir os progressos e as expectativas das partes interessadas.	OA16 – Planear as sprints e estimar as tarefas OA17 – Rever as sprints, comparando o planeado vs entregue OA18 – Negociar com o cliente
RA6 - Demonstrar conhecimento dos processos e metodologias de desenvolvimento de software, bem como dos princípios e fundamentos da melhoria de processos de software	OA15 – Interpretar as diferentes etapas e fases de um projeto seguindo o Waterfall OA16 – Compreender todos os resultados de um projeto OA17 – Melhorar os processos de um projeto em função de vários aspetos, como a equipa, o cliente, o âmbito, etc.	OA19 – Interpretar os diferentes processos e cerimónia do Scrum OA20 – Compreender a necessidade do DevOps OA21 – Melhorar os processos de um projeto de acordo com vários aspetos, tais como equipa, cliente, âmbito, entre outros.

6 IMPLEMENTAÇÃO E AVALIAÇÃO DE RESULTADOS

Já com os novos planos a serem utilizados pelos estudantes, foi necessário efetuar pequenas correções nos processos. Assim, neste capítulo são mostradas as alterações feitas no plano e uma avaliação da implementação dos novos processos.

6.1 Caracterização dos estudantes

No início de cada semestre (do ano 1 e do ano 2), os estudantes responderam a um pequeno questionário. O questionário tinha vários objetivos, sendo eles:

- Obter o tipo de personalidade dos estudantes;
- Saber se os estudantes tinham algum tipo de experiência profissional;
- Analisar processos de trabalho que os estudantes já utilizavam de forma voluntária noutros projetos da licenciatura;
- Obter níveis de conhecimento nalgumas áreas e ferramentas e tecnologias utilizadas noutros projetos da licenciatura, ou até mesmo fora do âmbito académico.

Estas informações poderão ser úteis na medida em que, ao sabermos o estado dos estudantes quando começaram a UC, será possível demonstrar ou falsificar que as alterações de processos beneficiaram efetivamente os estudantes. Desta forma, recolhemos dados para que seja possível verificar se as variações de resultados dos estudantes podem ser explicadas por:

- Hipótese 1: variações de perfil - para verificar esta hipótese, vamos recolher dados que caracterizam o perfil dos estudantes em cada ano e vamos verificar se a variação de resultados está correlacionada com a metodologia lecionada;
- Hipótese 2: nível de experiência profissional – para verificar esta hipótese, vamos recolher dados que mostram se os estudantes de um ano têm mais experiência profissional que os estudantes de outro ano;
- Hipótese 3: hábitos de trabalho – para verificar esta hipótese, vamos recolher dados que permitem perceber se os estudantes de um ano têm mais hábitos de trabalho noutros trabalhos de outras UC, por exemplo;
- Hipótese 4: níveis de conhecimento – para verificar esta hipótese, vamos recolher dados que nos mostram se os estudantes de um dos anos têm mais conhecimentos prévios em alguns processos e práticas utilizadas aos longo do projeto de software.

Como se tratava de um formulário opcional, no ano 1 obtivemos trinta e seis respostas de um total de setenta e oito estudantes, enquanto no ano 2 obtivemos vinte e nove respostas num total de setenta.

Na Figura 34 é possível vermos os diferentes tipos de personalidades dos estudantes do ano 1 e 2. Os tipos de personalidade foram obtidos a partir da ferramenta "16 personalities" [51], sendo eles:

- **Analistas** – são pessoas que combinam traços de intuição e pensamento. Elas tendem a ser racionais, lógicas e focadas em resolução de problemas;
- **Diplomatas** – reúnem traços de intuição e sentimento, destacando-se pela empatia e habilidades interpessoais;
- **Exploradores** – essas pessoas combinam observação e prospeção, e são caracterizados pela flexibilidade e adaptabilidade.
- **Vigilantes** – combinam observação e julgamento. São práticos, organizados e valorizam a estabilidade e ordem;

Podemos observar que a personalidade de analista e explorador são semelhantes entre os dois anos letivos, no entanto existe uma maior discrepância na percentagem de estudantes diplomatas e uma menor percentagem de estudantes vigilantes no ano 1, quando comparado ao ano 2.

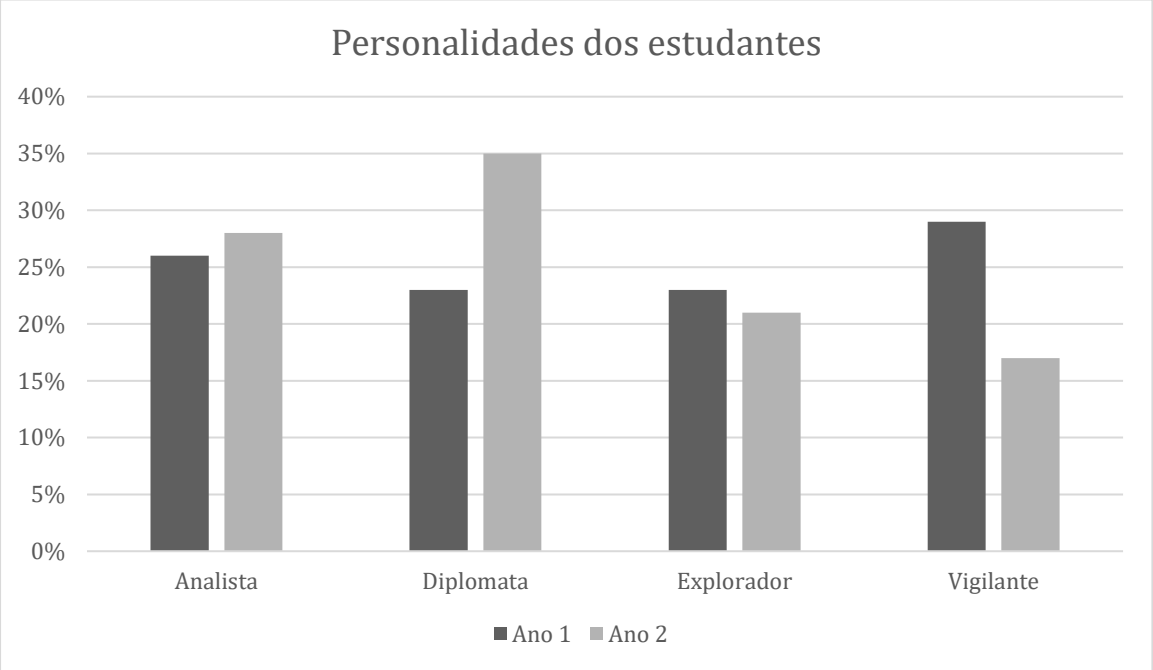


Figura 34 - Personalidades dos estudantes

Relativamente à experiência de trabalho dos estudantes, 22,9% dos estudantes do ano 1 tinham experiência profissional, enquanto no ano 2 havia uma percentagem de 24,1%. Na Figura 35 é possível vermos o número de anos e áreas de experiência profissional dos estudantes.

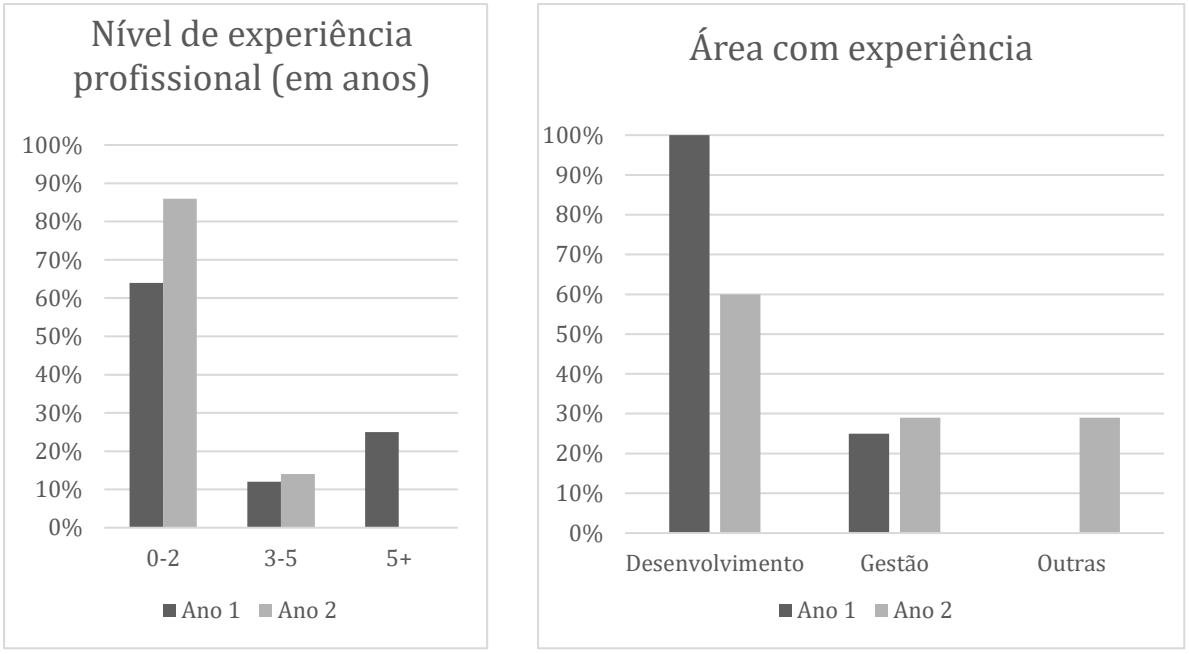


Figura 35 – Anos e áreas de experiência profissional dos estudantes

É de salientar que no ano 1 existe uma percentagem considerável de estudantes com mais de cinco anos de experiência profissional. No que diz respeito às várias áreas das quais os estudantes têm experiência profissional, no ano 1 todos os estudantes tinham experiência na área de desenvolvimento de software, sendo que 25% também tinham funções na área de gestão de projetos de software. Já no ano 2, temos uma menor percentagem de estudantes com experiência na área de desenvolvimento de software, no entanto tinham experiência noutras áreas, nomeadamente *Helpdesk* e integração/gestão de sistemas.

Ainda relativamente à experiência profissional, na Figura 36 é possível vermos as dimensões (número de colaboradores) das maiores empresas e equipas com que já trabalharam.

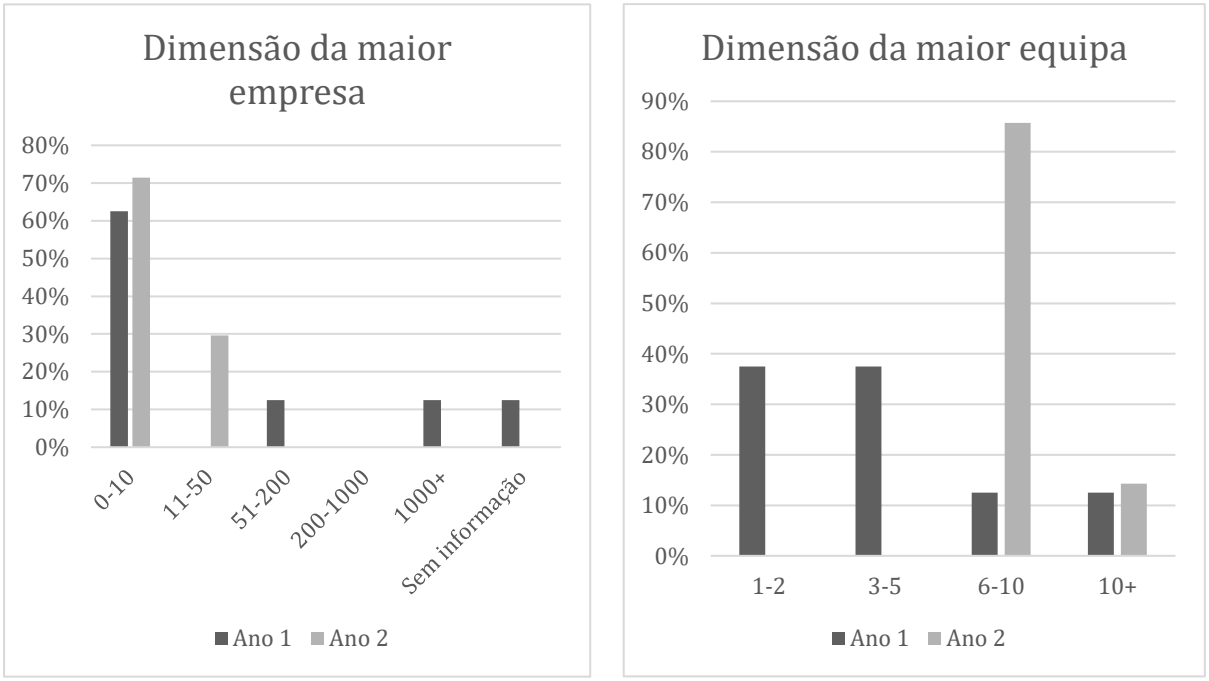


Figura 36 - Dimensões da maior empresa e maior equipa com que os estudantes com algum tipo de experiência profissional já trabalharam

É possível observar que, apesar de os estudantes do ano 1 terem experiências profissionais em empresas com maior número de colaboradores, os estudantes de ano 2 têm mais experiência a trabalhar em equipas maiores.

No que toca a processos de trabalho, na Figura 37 é possível vermos as preferências dos estudantes quando questionados se preferem fazer os trabalhos práticos das outras unidades curriculares sozinhos, a pares ou a grupos (três a cinco elementos). Também na Figura 37 é possível vermos o que é que os estudantes acham fundamental num projeto de software, sendo que a classificação foi alterada. No questionário, os estudantes tinham de atribuir uma classificação de "Discordo completamente" a "Concordo completamente", sendo esta substituída por uma classificação de 1-5 para apresentar os dados.

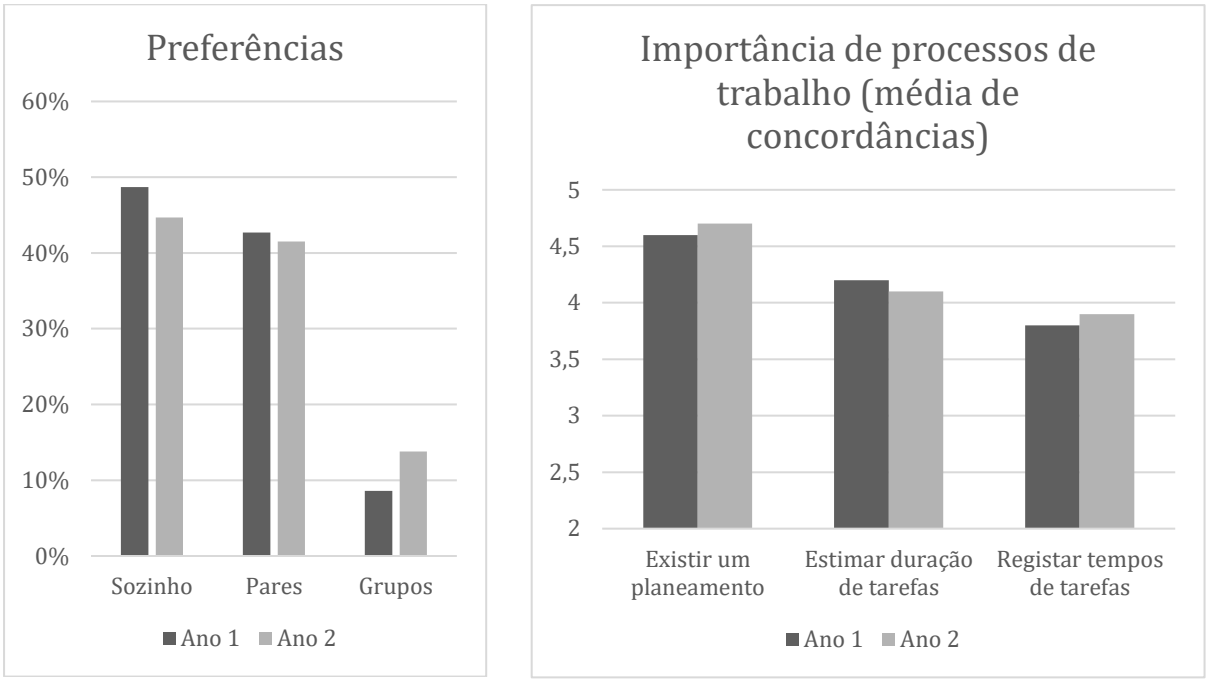


Figura 37 - Preferências dos estudantes quando questionados como é que preferiam trabalhar e grau de importância de alguns processos de trabalho que são utilizados no decorrer do projeto de software

Podemos observar que a percentagem de estudantes é praticamente semelhante nos dois anos letivos para as três preferências, sendo que no ano 2 existe uma diferença mínima na preferência de trabalhar em grupos.

Quanto aos processos de trabalho, os estudantes têm uma ideia semelhante relativamente a algumas práticas/processos na engenharia de software.

Para terminar a análise de processos de trabalho que geralmente os estudantes costumam ter nos trabalhos práticos de outras UC, temos a Figura 38. Os estudantes tinham de atribuir uma classificação de "Nunca", "Raramente", "Às vezes", "Frequentemente" e "Sempre". Para facilitar a demonstração destes dados, esta classificação foi substituída por uma cotação média de 1-5.

Podemos observar que os estudantes do ano 1 utilizavam alguns processos de trabalho com mais regularidade comparado aos estudantes do ano 2, especialmente relativamente aos testes e à utilização de ferramentas de versionamento de código.

Na Figura 39 podemos observar os vários níveis de conhecimento de algumas áreas e temas na gestão de projetos e da engenharia de software.

Analisando o gráfico, é possível concluir que os estudantes do ano 1 aparentemente chegaram à UC mais bem preparados que os estudantes do ano 2 na maior parte dos temas, exceto nos testes unitários, onde se nota claramente que os estudantes de no ano 2 tem um maior nível de conhecimento.

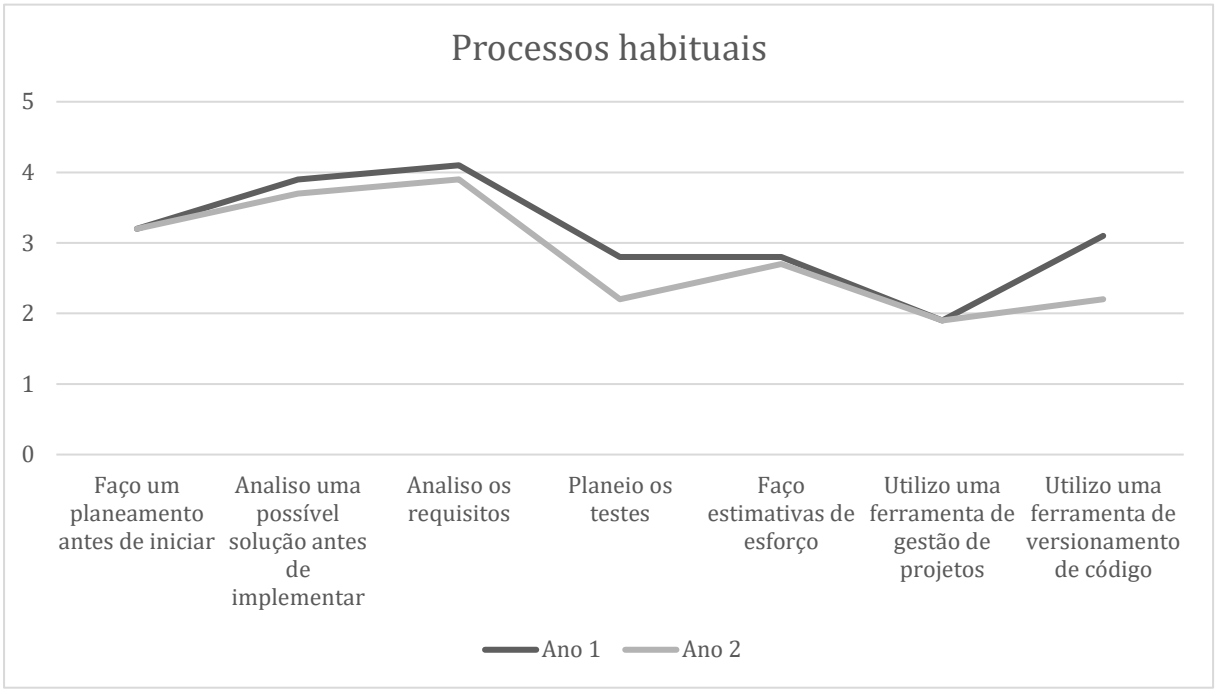


Figura 38 - Processos de trabalho utilizados pelos estudantes noutras UC

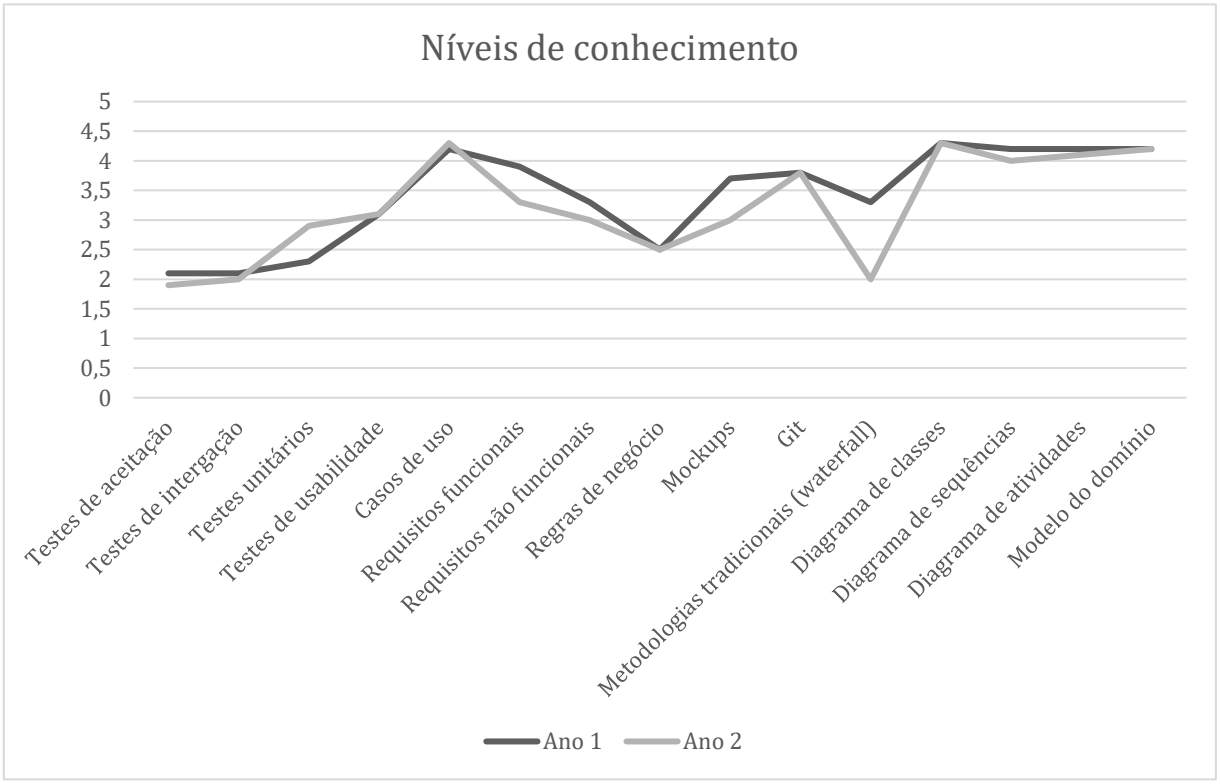


Figura 39 - Níveis de conhecimento dos estudantes relativamente a alguns temas abordados na UC de GPS

6.1.1 Resumo da análise

Em suma, pode-se concluir que:

- No ano 1 os estudantes tinham uma personalidade mais diplomata e menos vigilante, o que pode significar que são estudantes mais proativos. O facto dos estudantes do ano 1 serem mais diplomatas pode resultar num melhor trabalho em equipa, enquanto o facto dos estudantes do ano 2 serem mais vigilantes pode resultar num trabalho mais organizado;
- Relativamente à experiência profissional, os estudantes do ano 1 têm mais experiência profissional relativamente aos estudantes do ano 2, o que pode ser uma vantagem visto que têm um maior nível de senioridade;
- No que diz respeito a processos de trabalho, os estudantes do ano 1 iniciaram o ano letivo com uma pequena vantagem, visto já estarem habituados a alguns dos processos de trabalho utilizados na UC;
- Relativamente aos níveis de conhecimento nalgumas áreas/atividades utilizadas na unidade curricular, os estudantes do ano 1 também têm, em quase todas as áreas/atividades, uma vantagem quando comparados aos estudantes do ano 2. Este facto pode ser benéfico visto já estarem familiarizados com algumas das práticas utilizadas no trabalho prático.

6.2 Alterações aos processos

Com o decorrer do semestre, foi necessário efetuar pequenas alterações nos processos, sendo elas as seguintes:

- Não foi utilizada ferramenta de análise estática de código – optámos por abandonar a utilização do SonarCloud, uma vez que o repositório tinha de estar num grupo do GitLab para que todos os elementos do grupo pudessem analisar os resultados das análises. No entanto, para estar num grupo, o repositório teria de estar privado, sendo que o SonarCloud só permite análises a repositórios privados com subscrição paga.
- Não foi utilizada a *pipeline* CD – os estudantes teriam um gasto de tempo/esforço demasiado grande para as competências que iriam adquirir. Para além disso, demasiados estudantes estavam a ter problemas de recursos com as máquinas virtuais. Desta forma, optámos por substituir a utilização da máquina virtual usada para a demonstração da aplicação, com recurso a um *container* docker, para a execução de um ficheiro jar que é criado de forma automática. Este ficheiro jar é criado quando o MR é aceite para a *branch* de **qa** ou para a *branch* **main**. Desta forma, foi necessário alterar a *pipeline* CI para

criar este jar de forma automática, sendo que é possível observar no Anexo U o estado final da *pipeline* CI utilizada pelos estudantes.

6.3 Resultados

Para avaliar qual o ano letivo onde os estudantes obtiveram os melhores resultados no final dos respetivos semestres, fizemos uma comparação de notas de duas componentes: o exame final e o projeto de software. Além disso, para cada uma destas componentes de avaliação, analisámos o desempenho dos estudantes relativamente aos diferentes RA. Relativamente ao exame, conseguimos fazê-lo relacionando cada uma das questões dos exames com um determinado RA. No que diz respeito ao projeto de software, uma vez que os estudantes tinham reuniões de progresso todas as semanas, as equipas foram avaliadas tendo em conta listas de tópicos que estavam relacionadas com os diferentes RA.

Na Tabela 15 é apresentado qual dos anos letivos obteve melhores resultados em cada componente de avaliação, para cada RA. Esta avaliação teve em conta vários fatores, nomeadamente o feedback obtido durante as reuniões semanais, as notas semanais do trabalho prático e as notas finais do exame. Desta forma, foi feita uma análise meramente comparativa entre os dois anos letivos relativamente a cada um dos RA.

Tabela 15 - Comparação do resultado dos estudantes no final de cada ano letivo

RA	RA–1	RA–2	RA–3	RA–4	RA–5	RA–6
Exame final	Ano 2	Ano 2	Ano 2	Ano 1	Ano 2	Ano 2
Projeto de software	Ano 2	Ano 2	Ano 2	Ano 2	Ano 2	Ano 2

Os estudantes do ano 2, que seguiram as metodologias ágeis, demonstraram melhores resultados em quase todos os RA, apesar de terem demonstrado estar menos preparados quando iniciaram o projeto segundo o estudo feito inicialmente. Vamos tentar dar algumas explicações para os resultados observados.

Relativamente ao RA–1 (Descrever o desenvolvimento de software como uma disciplina de engenharia, utilizando a terminologia apropriada), como o Scrum é um processo mais iterativo do que o Waterfall, fez com que os estudantes assimilassem melhor os termos de engenharia de software.

No que diz respeito ao RA–2 (Realizar um projeto de software em equipa, seguindo processos definidos e boas práticas de engenharia de software), os estudantes do ano 2 tinham regras mais definidas em algumas ferramentas, como a utilização do Git. Havia uma estratégia de *branching* obrigatória (pelo menos três *branches*), revisões de código obrigatórias, entre outras regras.

Em relação ao RA–3 (Planear um projeto de desenvolvimento de software utilizando estimativas, calendarização de tarefas e alocação de recursos), os estudantes do ano

2 passaram pelo processo de estimar e agendar tarefas várias vezes, fazendo com que exercitassem este processo com mais frequência. Desta forma, conseguiram assimilar melhor alguns processos relativamente ao planeamento e gestão do projeto de software.

Quanto ao RA–4 (Aplicar as melhores práticas de gestão da qualidade, gestão de riscos, gestão de equipas e gestão de *stakeholders*), os estudantes do ano 1 obtiveram melhores resultados no exame porque assimilaram melhor, ainda que não tenham aplicado, diferentes estratégias de qualidade comparados aos estudantes do ano 2. Relativamente aos resultados práticos, estes estudantes aplicaram melhores práticas, como a revisão do código e os testes automatizados, que são executados continuamente.

Relativamente ao RA–5 (Gerir um projeto de software, monitorizando e controlando o seu progresso, garantindo a entrega do software dentro dos prazos e custos planeados e gerindo as expectativas dos *stakeholders*), como os estudantes fizeram demonstrações ao cliente ao longo de todo o projeto, foram capazes de negociar com ele e esclarecer dúvidas/pontos que por vezes não eram tão claros desde o início.

Para terminar, no que diz respeito ao RA–6 (Demonstrar conhecimento dos processos e metodologias de desenvolvimento de software, bem como dos princípios e fundamentos da melhoria de processos de software), tal como no RA–1, o facto de o Scrum ser iterativo faz com que os estudantes pratiquem os processos várias vezes, assimilando assim melhor os princípios e fundamentos das metodologias.

Na Figura 40 é possível ver as notas finais dos estudantes nos dois anos letivos. Salienta-se que só foram tidos em consideração os estudantes que concluíram a UC com sucesso. Não houve alunos que tivessem ido a exame que não tivessem obtido aprovação à UC.

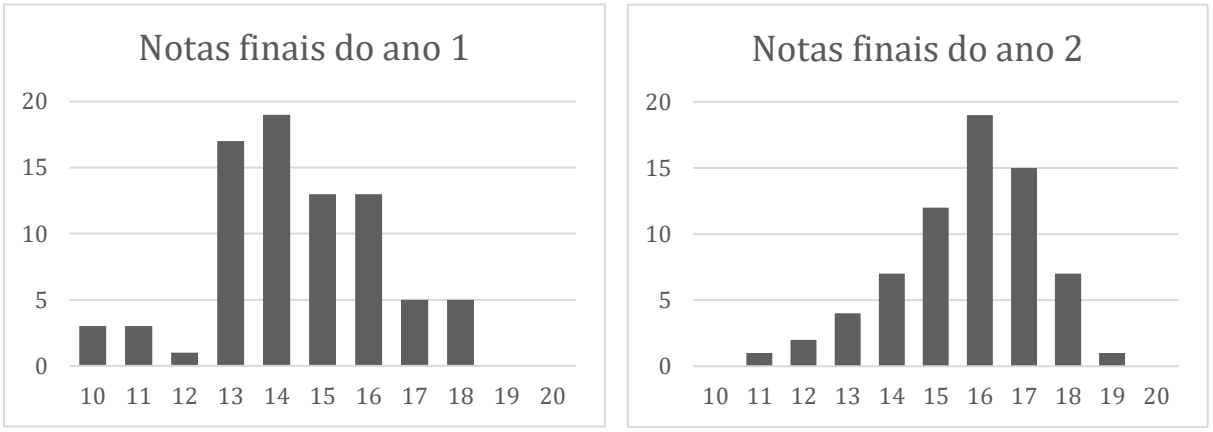


Figura 40 - Notas finais dos estudantes dos anos 1 e 2

Adicionalmente, na Figura 41 é possível observar as notas finais dos últimos doze anos letivos, incluindo o ano letivo 2023/2024.

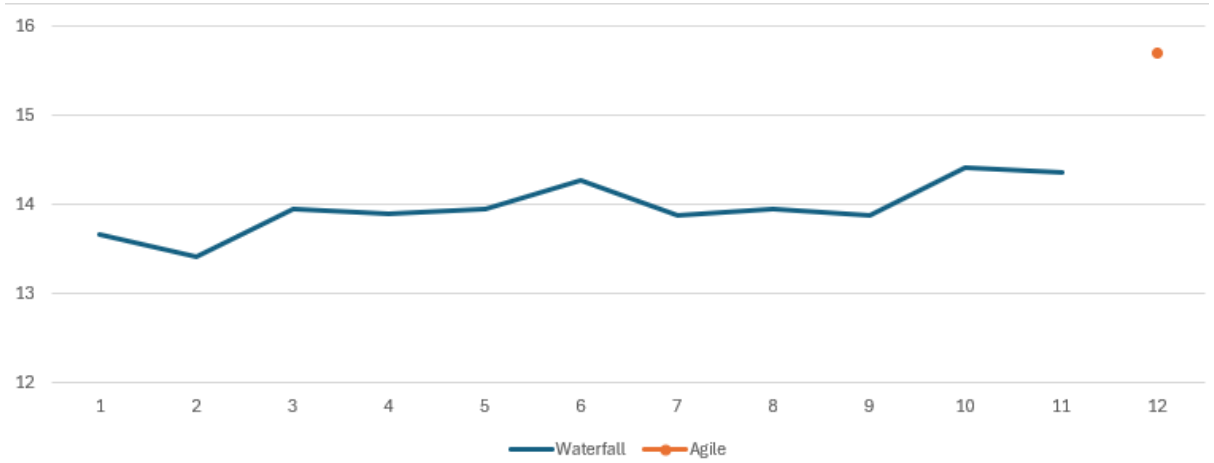


Figura 41 - Notas finais dos últimos doze anos letivos

As médias finais dos anos onde foi lecionada a metodologia Waterfall variaram entre os 13,4 e os 14,4 valores, sendo a média 13,9 valores. Já relativamente a este ano letivo, onde foi seguida uma metodologia ágil, a nota média da UC foi 15,7 valores, mostrando um claro aumento no que diz respeito ao aproveitamento por parte dos estudantes.

7 CONCLUSÕES E TRABALHO FUTURO

Este trabalho, que teve a duração de dois anos letivos, teve como principal objetivo a modernização da UC de GPS da LEI do ISEC.

Assim, começámos por recolher um conjunto de boas práticas da engenharia de software e da gestão de projetos de software.

Para além disso, recolhemos evidências que as metodologias ágeis são uma tendência atual, utilizada pela indústria no geral para alcançar uma melhor qualidade do produto e um maior alinhamento com as necessidades e expectativas dos *stakeholders*. Estas evidências foram obtidas através de um trabalho de campo junto de empresas influentes a nível regional, nacional e internacional e através da análise de vários artigos científicos, revistas, relatórios, entre outros.

Também foi feito um trabalho de campo junto de várias IES, que mostra que o ensino de gestão de projetos de software está atualmente muito voltado para as metodologias ágeis.

Desta forma, pretendeu-se fazer uma alteração de metodologia, processos, ferramentas e práticas na UC de GPS da LEI do ISEC, de forma a ir ao encontro das necessidades e tendências da indústria e do ambiente académico. Fez-se então uma alteração de metodologia, pelo que ao invés de uma metodologia tradicional (Waterfall), os alunos passaram a seguir uma metodologia ágil (Scrum), com algumas práticas das metodologias tradicionais, como por exemplo a gestão de riscos e algumas práticas DevOps.

De forma a avaliar os resultados, fizemos um questionário inicial aos estudantes dos dois anos letivos, para percebermos o ponto inicial dos mesmos. Isto era uma questão importante uma vez que poderia justificar o melhor ou pior aproveitamento dos estudantes.

Relativamente a resultados, foi possível concluir que os alunos do ano 1 chegaram mais bem preparados à UC de GPS, o que fazia prever que obtivessem melhores resultados. No entanto, contra as expectativas, os alunos do ano 2 obtiveram melhor aproveitamento em praticamente todos os resultados de aprendizagem.

Assim, podemos argumentar que o ensino das metodologias ágeis não só abordou temas atuais e enquadrados com a realidade na indústria, como melhorou as competências dos estudantes em gestão de projetos de software.

Como trabalho futuro, pretendemos obter feedback das empresas que entrevistámos inicialmente, ainda antes da reformulação da UC de GPS, de maneira a confirmar se as empresas têm a mesma opinião. Também seria interessante começar a aplicar ainda mais práticas DevOps e inteligência artificial, uma vez que também são uma tendência. No Ano 2, os alunos já utilizaram um *pipeline* CI pré-desenvolvida, mas seria importante começar a aplicar mais práticas e analisar a sua inclusão.

Para além disso, não foi possível obter nenhum tipo de relação entre as personalidades dos estudantes participantes no estudo. No entanto, seria interessante existir um trabalho multidisciplinar entre as áreas de informática e psicologia. Caso fosse possível identificar os tipos de personalidade que se encaixam melhor em cada umas das metodologias de desenvolvimento de software, podiam ser criados vários grupos de estudantes onde cada grupo seguia uma metodologia diferentes, de forma a obterem melhores aproveitamentos.

No entanto, uma vez que a área da gestão de projetos de software está em constante evolução, poderíamos mencionar inúmeros trabalhos futuros, como por exemplo a melhoria da gestão da qualidade do código (clarificando a revisão e aceitação do código e a utilização de uma ferramenta de análise estática de código, visto que o SonarQube não foi utilizado), a entrega do software (criando um servidor para *deploy* e demonstração do código), a gestão das equipas (criando equipas com competências e personalidades distintas), entre outras.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. Mishra and Y. I. Alzoubi, "Structured software development versus agile software development: a comparative analysis," *International Journal of System Assurance Engineering and Management*, vol. 14, no. 4, pp. 1504–1522, Aug. 2023, doi: 10.1007/S13198-023-01958-5.

[2] A. Aitken and V. Ilango, "A comparative analysis of traditional software engineering and agile software development," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 4751–4760, 2013, doi: 10.1109/HICSS.2013.31.

[3] "State of Agile Report," 2022. Accessed: Apr. 28, 2024. [Online]. Available: https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf

[4] K. Schwaber and J. Sutherland, "The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game," 2020. Accessed: Apr. 28, 2024. [Online]. Available: https://www.scrum.org/resources/scrum-guide

[5] "Writing Course Goals/Learning Outcomes and Learning Objectives – Center for Excellence in Learning and Teaching." Accessed: Jul. 02, 2024. [Online]. Available: https://www.celt.iastate.edu/instructional-strategies/preparing-to-teach/tips-on-writing-course-goalslearning-outcomes-and-measureable-learning-objectives/

[6] IEEE Computer Society and Association for Computing Machinery, "Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," 2015, p. 94.

[7] Association for Computing Machinery and IEEE Computer Society, *Computing Curricula 2020 (CC2020): Paradigms for Global Computing Education*. 2020. doi: 10.1145/3467967.

[8] Project Management Institute, "Project Management Course Specifications," in *Guidelines for Undergraduate Curriculum and Resources*, vol. 1, pp. 1–92.

[9] T. Thesing, C. Feldmann, and M. Burchardt, "Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project," *Procedia Comput Sci*, vol. 181, pp. 746–756, Jan. 2021, doi: 10.1016/J.PROCS.2021.01.227.

[10] W. Rosa, B. K. Clark, R. Madachy, and B. W. Boehm, "Empirical Effort and Schedule Estimation Models for Agile Processes in the US DoD," *IEEE*

Transactions on Software Engineering, vol. 48, no. 8, pp. 3117–3130, Aug. 2022, doi: 10.1109/TSE.2021.3080666.

[11] Y. Beng Leau, W. Khong Loo, W. Yip Tham, and S. Fun Tan, "Software Development Life Cycle AGILE vs Traditional Approaches".

[12] Ian Sommerville, *Software Engineering*, 10th ed. Boston: Pearson Education Limited, 2016.

[13] V. S. Alagar and K. Periyasamy, "Specification of Software Systems," 2011, doi: 10.1007/978-0-85729-277-3.

[14] M. Stoica, M. Mircea, and B. Ghilic-Micu, "Software Development: Agile vs. Traditional," *Informatica Economică*, vol. 17, no. 4, 2013, doi: 10.12948/issn14531305/17.4.2013.06.

[15] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, "Challenges in software evolution," *International Workshop on Principles of Software Evolution (IWPSE)*, vol. 2005, pp. 13–22, 2005, doi: 10.1109/IWPSE.2005.7.

[16] Project Management Institute, *A guide to the project management body of knowledge*, 6th ed. Accessed: May 23, 2024. [Online]. Available: www.PMI.org

[17] E. Masciadra, "Traditional Project Management," *Knowledge Management and Organizational Learning*, vol. 5, pp. 3–23, 2017, doi: 10.1007/978-3-319-51067-5_1/COVER.

[18] F. Bilimleri Dergisi, M. Javanmard, and M. Alian, "Comparison between Agile and Traditional software development methodologies," *Cumhuriyet University Faculty of Science Science Journal (CSJ)*, vol. 36, no. 3, p. 36, 2015, Accessed: Feb. 11, 2023. [Online]. Available: http://dergi.cumhuriyet.edu.tr/cumuscij©2015

[19] S. Shaikh and S. Abro, "Comparison of Traditional & Agile Software Development Methodology: A Short Survey," *International Journal of Software Engineering and Computer Systems*, vol. 5, no. 2, pp. 1–14, 2019, doi: 10.15282/ijsecs.5.2.2019.1.0057.

[20] R. Mall, *Fundamentals of Software Egineering*, Fourth. PHI Learning Private Limited, 2014.

[21] C. Kaur and V. Kumar, "Comparative Analysis of Iterative Waterfall Model and Scrum," *FP-International Journal of Computer Science Research*, vol. 2, no. 1, pp. 11–14, 2015.

[22] R. Pressman, *Software Engineering: A Practitioner's Approach*, Seventh Edition. 2010. Accessed: Feb. 11, 2023. [Online]. Available:

https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf

[23] N. Mohammed, A. Munassar, and A. Govardhan, "A Comparison Between Five Models Of Software Engineering," *IJCSI International Journal of Computer Science Issues*, vol. 7, no. 5, 2010, Accessed: Feb. 11, 2023. [Online]. Available: www.IJCSI.org

[24] A. Mujumdar, G. Masiwal, and P. M. Chawan, "Analysis of various Software Process Models," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 3, pp. 2015–2021, 2012, Accessed: Feb. 11, 2023. [Online]. Available: www.ijera.com

[25] A. Alshamrani, A. Bahattab, and I. Fulton, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model," *IJCSI International Journal of Computer Science Issues*, vol. 12, no. 1, 2015.

[26] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 189–196, 2014, doi: 10.1109/ACCT.2014.73.

[27] A. Anwar, "A Review of RUP (Rational Unified Process)," *Ashraf Anwar International Journal of Software Engineering (IJSE)*, no. 5, p. 8, 2014.

[28] J. Manalil, "RATIONAL UNIFIED PROCESS Seminar Report," 2010.

[29] "What Is the V-Model? (Definition, Examples) | Built In." Accessed: Dec. 27, 2024. [Online]. Available: https://builtin.com/software-engineering-perspectives/v-model

[30] K. Beck *et al.*, "Manifesto for Agile Software Development." Accessed: May 23, 2024. [Online]. Available: https://agilemanifesto.org/

[31] S. Kumar Dora and P. Dubey, "Software development life cycle (SDLC) analytical comparison and survey on traditional and agile methodology," 2013, Accessed: Feb. 11, 2023. [Online]. Available: www.abhinavjournal.com

[32] F. Tsui, O. Karam, and B. Bernal, *Essentials of software engineering*, Third edition. 2014. Accessed: Feb. 13, 2023. [Online]. Available: https://www.worldcat.org/pt/title/essentials-of-software-engineering-third-edition/oclc/864219285

[33] M. Alqudah and R. Razali, "A comparison of scrum and Kanban for identifying their selection factors," *Proceedings of the 2017 6th International Conference on Electrical Engineering and Informatics: Sustainable Society Through*

Digital Innovation, ICEEI 2017, vol. 2017-November, pp. 1–6, Mar. 2018, doi: 10.1109/ICEEI.2017.8312434.

[34] K. Bhavsar, Dr. V. Shah, and Dr. S. Gopalan, "Scrumban: An Agile Integration of Scrum and Kanban in Software Engineering," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 4, pp. 1626–1634, Feb. 2020, doi: 10.35940/ijitee.D1566.029420.

[35] D. Anderson and A. Carmichael, *Essential Kanban Condensed*. 2016. Accessed: May 23, 2024. [Online]. Available: https://www.thescrummaster.co.uk/wp-content/uploads/2019/09/Essential-Kanban-Condensed-v1.0.0.pdf

[36] A. Janes, "A guide to lean software development in action," *2015 IEEE 8th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015 - Proceedings*, May 2015, doi: 10.1109/ICSTW.2015.7107412.

[37] D. Wells, "Extreme Programming: A Gentle Introduction." Accessed: May 23, 2024. [Online]. Available: http://www.extremeprogramming.org/

[38] R. Fojtik, "Extreme programming in development of specific software," *Procedia Comput Sci*, vol. 3, pp. 1464–1468, 2011, doi: 10.1016/J.PROCS.2011.01.032.

[39] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A Survey of DevOps Concepts and Challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, Nov. 2019, doi: 10.1145/3359981.

[40] R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer, "What is DevOps? A systematic mapping study on definitions and practices," *ACM International Conference Proceeding Series*, vol. 24-May-2016, May 2016, doi: 10.1145/2962695.2962707.

[41] DevOps Research & Assessment, "2022 Accelerate State of DevOps Report." Accessed: May 23, 2024. [Online]. Available: https://services.google.com/fh/files/misc/2022_state_of_devops_report.pdf

[42] DevOps Research & Assessment and puppet, "State of DevOps Report 2016." Accessed: Jul. 10, 2024. [Online]. Available: https://dora.dev/publications/pdf/state-of-devops-2016.pdf

[43] DevOps Research & Assessment, "State of DevOps 2019," 2019. Accessed: Jul. 10, 2024. [Online]. Available: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

[44] M. Alshayeb, S. Mahmood, and K. Aljasser, "Moving from Waterfall to Agile Process in Software Engineering Capstone Projects," pp. 107–114, May 2018, doi: 10.5121/CSIT.2018.80808.

[45] R. Wlodarski, A. Poniszewska-Maranda, and J. R. Falleri, "Comparative Case Study of Plan-Driven and Agile Approaches in Student Computing Projects," *2020 28th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2020*, Sep. 2020, doi: 10.23919/SOFTCOM50211.2020.9238196.

[46] V. Kate, S. Bhalerao, and V. Sharma, "Exploring Agile Methodologies in Educational Software Development- A Comparative Analysis and Project Management Insights," *3rd IEEE International Conference on ICT in Business Industry and Government, ICTBIG 2023*, 2023, doi: 10.1109/ICTBIG59752.2023.10456214.

[47] A. Sinha and P. Das, "Agile Methodology Vs. Traditional Waterfall SDLC: A case study on Quality Assurance process in Software Industry," *2021 5th International Conference on Electronics, Materials Engineering and Nano-Technology, IEMENTech 2021*, 2021, doi: 10.1109/IEMENTECH53263.2021.9614779.

[48] D. F. Rico and H. H. Sayani, "Use of agile methods in software engineering education," *Proceedings - 2009 Agile Conference, AGILE 2009*, pp. 174–179, 2009, doi: 10.1109/AGILE.2009.13.

[49] "Earned Value Management Systems Tool (EVMS) | PMI." Accessed: Apr. 29, 2024. [Online]. Available: https://www.pmi.org/learning/library/earned-value-management-systems-analysis-8026

[50] "GitLab." Accessed: Apr. 29, 2024. [Online]. Available: https://gitlab.com/

[51] "Free personality test, type descriptions, relationship and career advice | 16Personalities." Accessed: Oct. 20, 2024. [Online]. Available: https://www.16personalities.com/

[52] V. Yakovyna, M. Seniv, and I. Symets, "The Relation between Software Development Methodologies and Factors Affecting Software Reliability," *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, vol. 1, pp. 377–381, Sep. 2020, doi: 10.1109/CSIT49958.2020.9321937.

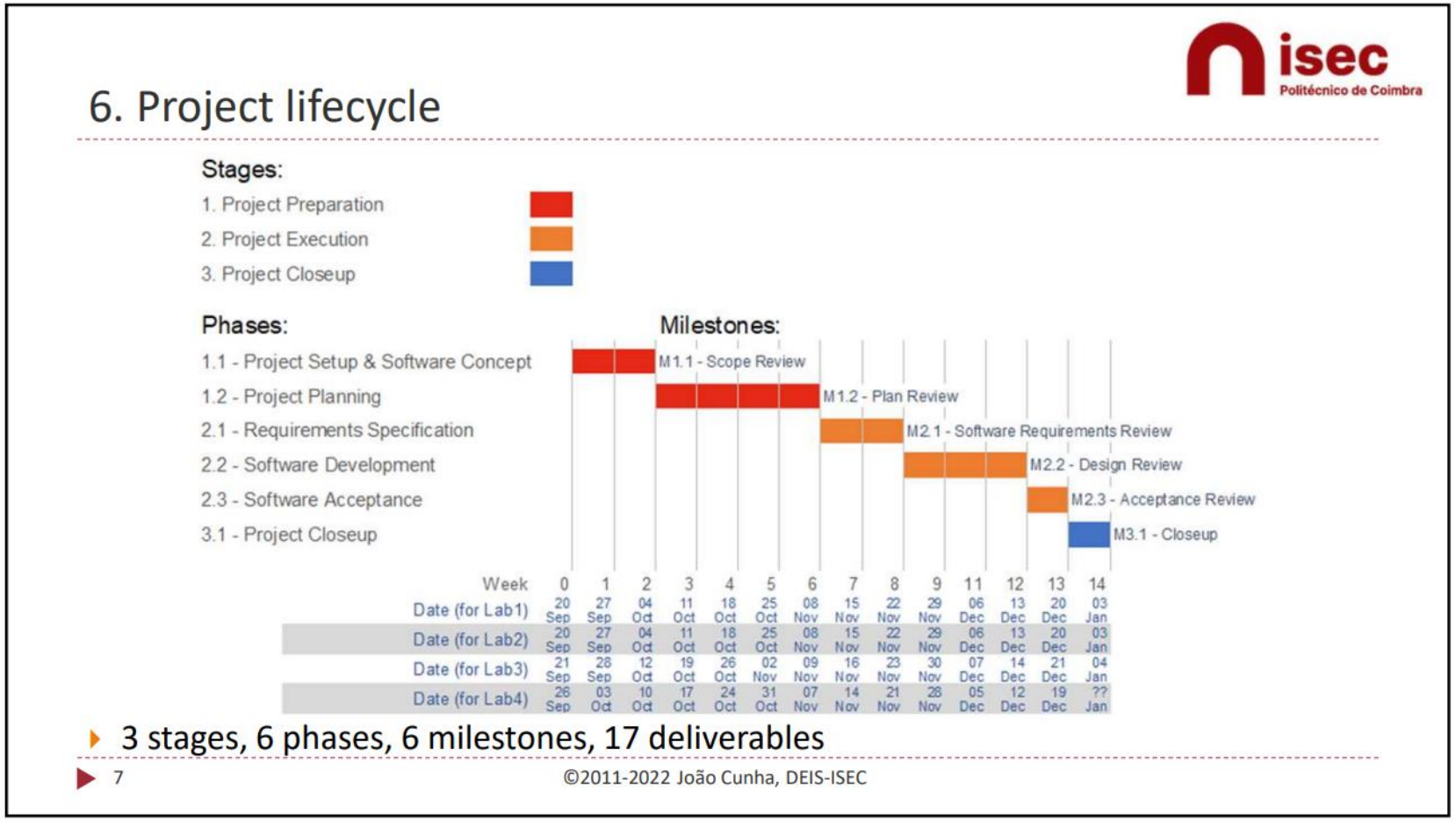
[53] "Earned Value Management Systems Tool (EVMS) | PMI." Accessed: Aug. 07, 2024. [Online]. Available: https://www.pmi.org/learning/library/earned-value-management-systems-analysis-8026

[54] "AESE insight #32 - AESE Business School - Formação de Executivos." Accessed: Oct. 05, 2024. [Online]. Available: https://www.aese.pt/aese-insight-32/

[55] "Periodic Table of DevSecOps Tools | Digital.ai," *https://digital.ai/*, Accessed: Jul. 10, 2024. [Online]. Available: https://digital.ai/learn/devsecops-periodic-table/

ANEXOS

Anexo A. SLIDE APRESENTADO EM GPS COM O CICLO DE VIDA DO PROJETO



7

Anexo B. TEMPLATE DO DOCUMENTO D1.1.1 – VISION AND SCOPE

Software Project Management 2022/2023

«Project Name»

D1.1.1 - Vision and Scope (Visão e Âmbito)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- «version, reviewer name, preferably not in the author's list»

Approver (Aprovado por:)

- «version, approver name, preferably not in the author's list»

Table of Contents (Índice)

«this outline and explanation of vision and scope is from Stellman, 2.1.3. Keep the structure, write the contents, clean the placeholders in "«»"»

1. PROBLEM STATEMENT (PROBLEMA)

1.1 Project background (Descrição do problema)

This section contains a summary of the problem that the project will solve. It should provide a brief history of the problem and an explanation of how the organization justified the decision to build software to address it. This section should cover the reasons why the problem exists, the organization's history with this problem, any previous projects that were undertaken to try to address it, and the way that the decision to begin this project was reached.

1.2 Stakeholders (Entidades envolvidas)

This is a bulleted list of the stakeholders. Each stakeholder may be referred to by name, title, or role ("support group manager," "CTO," "senior manager"). The needs of each stakeholder are described in a few sentences.

1.3 Users (Utilizadores)

This is a bulleted list of the users. As with the stakeholders, each user can either be referred to by name or role ("support rep," "call quality auditor," "home web site user")however, if there are many users, it is usually inefficient to try to name each one. The needs of each user are described.

2. VISION & SCOPE OF THE SOLUTION (VISÃO E ÂMBITO DA SOLUÇÃO)

2.1 Vision statement (Visão)

The goal of the vision statement is to describe what the project is expected to accomplish. It should explain what the purpose of the project is. This should be a compelling reason, a solid justification for spending time, money, and resources on the project. The best time to write the vision statement is after talking to the stakeholders and users and writing down their needs; by this time, a concrete understanding of the project should be starting to jell.

2.2 List of features (Lista de requisitos iniciais)

This section contains a list of features. A feature is as a cohesive area of the software that fulfills a specific need by providing a set of services or capabilities. Any software package, in fact, any engineered product, can be broken down into features. The project manager can choose the number of features in the vision and scope document by changing the level of detail or granularity of each feature, and by combining multiple features into a single one. Sometimes those features are small ("screw-top cap," "holds one liter of liquid"); sometimes they are big ("four-wheel drive," "seats seven passengers"). It is useful to describe a product in about 10 features in the vision and scope document, because this usually yields a level of complexity that most people reading it are comfortable with. Adding too many features will overwhelm most readers.

Each feature should be listed in a separate paragraph or bullet point. It should be given a name, followed by a description of the functionality that it provides. This description does not need to be detailed; it can simply be a few sentences that give a general explanation of the feature. However, if there is more information that a stakeholder or project team member feels should be included, it is important to include that information. For example, it is sometimes useful to include a use case (see Chapter 6), as long as it is written in such a way that all of the stakeholders can read and understand it.

2.3 Features that will not be developed (Requisitos que não serão concretizados)

Features are often left out of a project on purpose. When a feature is explicitly left out of the software, it should be added to this section to tell the reader that a decision was made to exclude it. For example, one way to handle an unrealistic deadline is by removing one or more features from the software, in which case the removed features should be moved into this section. The reason these features should be moved rather than deleted from the document is that otherwise, readers might assume that they were overlooked and bring them up in a review. This is especially important during the review of the document because it allows everyone to agree on the exclusion of the feature (or object to it).

2.4 Assumptions (Pressupostos)

This is the list of assumptions that the stakeholders, users, or project team have made. The team should hold a brainstorming session to come up with a list of assumptions. (See Chapter 3 for more information on assumptions.)

Anexo C. TEMPLATE DO DOCUMENTO D1.2.1 – SOFTWARE DEVELOPMENT PLAN

Software Project Management 2022/2023

«Project Name»

D1.2.1 - Software Development Plan
(Plano de Desenvolvimento de Software)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

Table of Contents (Índice)

4. Technical Process
   4.1. System architecture
   4.2. Methods tools and techniques

«Outline and explanation adapted from Construx - Survival Guide»

This is only a guideline. You may change its structure and contents.

1. INTRODUCTION

This section provides a summary of project purpose, scope, and objective. It may also include background for the project, a summary of the project planning, or other contextual material as appropriate. May also contain a formal or informal collection of assumptions, constraints, external dependencies, etc.

This should not be a copy of Vision&Scope, but may include information from that document.

2. PROJECT ORGANIZATION

This section covers all aspects of managing the project that are not related to estimating, planning, and controlling the work

2.1 Project Lifecycle

Describe the lifecycle for the project, discussing phases and milestones. This should provide a clear view of the project timeline from a process standpoint. This is the baseline plan.

2.2 Deliverables

List the ultimate deliverables that will be provided to the customers of the project. Thake the list from Slides.

2.3 Supporting Documents and Records

Provides a commented list to all the supporting documents and project records or record locations.

2.4 Project Management

Describe the organization structure (management, groups, roles, people, etc.) of the project and relationships to the external organization or other projects.

3 BASELINE PLAN AND CONTROL

This section covers all aspects of managing the project related to estimating, planning, and controlling the work. Most project plans will contain pointers to specialized artifacts and group of artifacts for the items in this section.

3.1 Estimates

Describe estimation process and moments. Provide link to the baseline project estimates.

3.2 Schedule

This section refers to schedules for the project. Provide a top-level summary schedule, and a link to the document with the detailed schedule.

This should include a WBS, resources allocation, effort and start/end dates.

3.3 Tracking and Control

Describe how project cost and schedule will be tracked and controlled throughout the project. Provide a link to the tracking documents. We will use Earned Value Analysis.

4 TECHNICAL PROCESS

Describes or summarizes the top-level technical processes used on the project.

4.1 System architecture

Describe a high-level system architecture.

4.2 Methods tools and techniques

Covers environment, tools, methods, and techniques related to technology and software engineering.

Anexo D. TEMPLATE DO DOCUMENTO D1.2.2 – QUALITY ASSURANCE PLAN

Software Project Management 2022/2023

«Project Name»

D1.2.2 - Quality Assurance Plan
(Plano de Garantia de Qualidade)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

Table of Contents (Índice)

5. Risk Management
   5.1. Risk identification and assessment
   5.2. Risk monitoring and control
6. Coding Standards
7. Change Management

«Outline and explanation adapted from Construx - Survival Guide»

This is only a guideline. You may change its structure and contents.

1. INTRODUCTION

The quality assurance plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes. It also sets out which organizational standards should be applied.

Link to quality assessment report.

2. QUALITY GOALS

The most important external quality attributes identified for this project are the following:

- List 1 or 2 quality attributes and justify each one. Explain how these attributes are measured.

Regarding internal quality, the following attributes have been identified as the most important:

- List 1 or 2 quality attributes and justify each one. Explain how these attributes are measured.

These goals are reported in D2.3.2 - Quality Assessment Report (link to the document), Section 2.

3. REVIEWS

Every document produced must be reviewed and approved. Code may also be reviewed. This section describes two types of reviews.

A Link to the review records (D2.3.2 - Quality Assessment Report Section 3) should be provided.

3.1 Inspections

Explain Inspections. Be clear and concise.

Provide link to folder with inspection records.

3.1 Walkthroughs

Explain Walkthroughs. Be clear and concise.

3.2 Deskchecks

Explain Deskchecks. Be clear and concise.

4. TESTS

Three types of tests shall be applied: unit, integration and acceptance.

Link to D2.3.2 - Quality Assessment Report, Section 4.

4.1 Unit Testing

Describe unit test plan (what will be tested, when, by whom, how).

4.2 Integration Testing

Describe the integration test plan (How, when, by whom).

4.3 Acceptance Testing

Describe acceptance test plan (test plan document, when, by whom, how).

Provide links to test plan and report.

5. RISK MANAGEMENT

Describe how risks will be actively managed on a project.

Link to D2.1.2 – Risk Plan and D2.3.2 - Quality Assessment Report, Section 5.

5.1 Risk identification and assessment

Describe how risks are identified and assessed

Provide a link to the Risk List

Each risk is assigned a impact of:

tolerable(1), serious(3), or catastrophic(5)

and a probability of occurrence with the assigned impact of:

very low(1), low(2), moderate(3), high(4) or very high(5).

For the top risks, with

Probability x Impact ≥ 15

potential indicators and actions have been described.

5.2 Risk monitoring and control

Risks are reviewed every week.

Risks with P x I ≥ 20 must be mitigated.

6. CODING STANDARDS

Code will be written according to code conventions. Provide a link to the conventions document, and to D2.3.2 - Quality Assessment Report, Section 6.

7. CHANGE MANAGEMENT

Explain how change management works.

Link to D2.3.2 - Quality Assessment Report, Section 7.

Anexo E. TEMPLATE DO DOCUMENTO D2.1.1 – SOFTWARE REQUIREMENTS SPECIFICATION

Software Project Management 2022/2023

«Project Name»

D2.1.1 – Software Requirements Specification (Especificação de Requisitos de Software)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

Table of Contents (Índice)

«Outline and explanation adapted from A. Stellman & J.Greene, "Applied Software Project Management"»

1 INTRODUCTION

The introduction serves to orient the reader. It describes both the system and the SRS itself.

1.1 Purpose

This section describes the purpose of the document. Typically, this will contain a brief two- or three-sentence description, including the name of the project. For example: "The purpose of this document is to serve as a guide to designers, developers, and testers who are responsible for the engineering of the (name of project) project. It should give the engineers all of the information necessary to design, develop, and test the software." This is to ensure that the person reading the document understands what he or she is looking at.

1.2 Scope

This section contains a brief description of the scope of the document. If the SRS is a complete description of the software, then it will state something similar to: "This document contains a complete description of the functionality of the (name of project) project. It consists of use cases, functional requirements and nonfunctional requirements, which, taken together, form a complete description of the software." For complex software, the requirements for the project might be divided into several SRS documents. In this case, the scope should indicate which portion of the project is covered in this document.

1.3 System overview

This section contains a description of the system. This is essentially a brief summary of the vision and scope of the project.

1.4 References

Any references to other documents should be included here. These may include other documents in the organization, work products, articles, and anything else that is relevant to understanding the SRS. If there is an organizational intranet, this section often includes URLs of referenced documents.

2 USE CASES

The "Use Cases" section contains each of the use cases.

Divide in subsections as needed.

Present **Use Case Diagrams**.

Example:

Name	**UC-8: Search**
Summary	All occurrences of a search term are replaced with replacement text.

Rationale	While editing a document, many users find that there is text somewhere in the file being edited that needs to be replaced but searching for it manually by looking through the entire document is time-consuming and ineffective. The search-and-replace function allows the user to find it automatically and replace it with specified text. Sometimes this term is repeated in many places and needs to be replaced. At other times, only the first occurrence should be replaced. The user may also wish to simply find the location of that text without replacing it.
Users	All users
Preconditions	A document is loaded and being edited.
Basic course of events	1. The user indicates that the software is to perform a search-and-replace in the document. 2. The software responds by requesting the search term and the replacement text. 3. The user inputs the search term and replacement text and indicates that all occurrences are to be replaced. 4. The software replaces all occurrences of the search term with the replacement text.
Alternative paths	1. In Step 3, the user indicates that only the first occurrence is to be replaced. In this case, the software finds the first occurrence of the search term in the document being edited and replaces it with the replacement text. The postcondition state is identical, except only the first occurrence is replaced, and the replacement text is highlighted. 2. In Step 3, the user indicates that the software is only to search and not replace and does not specify replacement text. In this case, the software highlights the first occurrence of the search term and the use case ends. 3. The user may decide to abort the search-and-replace operation at any time during Steps 1, 2, or 3. In this case, the software returns to the precondition state.
Postconditions	All occurrences of the search term have been replaced with the replacement text.

3 FUNCTIONAL REQUIREMENTS

Example:

Name	**FR-4: Case sensitivity in search-and-replace**
Summary	The search-and-replace feature must have case sensitivity in both the search and the replacement.
Rationale	A user will often search for a word that is part of a sentence, title, heading, or other kind of text that is not all lowercase. The search-and-replace function needs to be aware of that, and give the user the option to ignore it.

Requirements	When a user invokes the search-and-replace function, the software must give the option to do a case-sensitive search. By default, the search will match any text that has the same letters as the search term, even if the case is different. If the user indicates that the search is to be done with case-sensitivity turned on, then the software will only match text in the document where the case is identical to that of the search term. During a search and replace, when the software replaces original text in the document with the replacement text specified by the user, the software retains the case of the original text as follows: • If the original text was all uppercase, then the replacement text must be inserted in all uppercase. • If the original text was all lowercase, then the replacement text must be inserted in all lowercase. • If the original text had the first character uppercase and the rest of the characters lowercase, then the replacement text must reflect this case as well. • If the original text was sentence case (where the first letter of each word is uppercase), then the replacement text must be inserted in sentence case. • In all other cases, the replacement text should be inserted using the case specified by the user.
References	UC-8: Search

4 NONFUNCTIONAL REQUIREMENTS

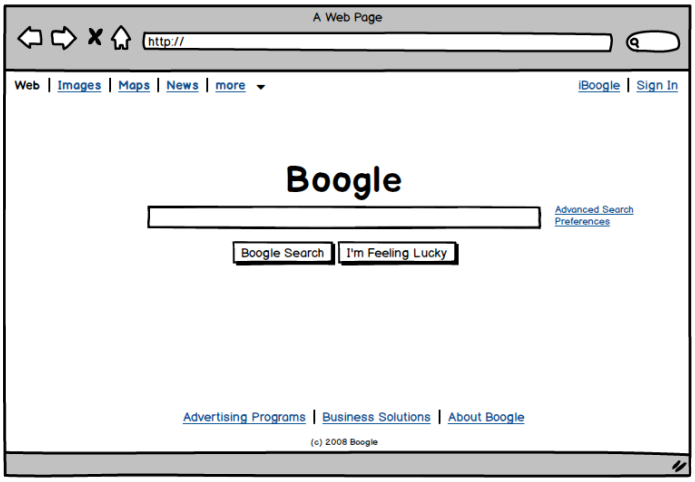
Describe nonfunctional requirements such as availability, performance, reliability, usability, reusability, etc..

Name	**NFR-2: Aesthetics**
Summary	Beautiful design.
Rationale	The users are mainly artists, so this product shall come out from the competition due to a beautiful design.
Requirements	The user must be pleased with the GUI appearance. In a score of 1 to 10, the potential users must give at least an average of 9 to the aesthetics of the GUI.
References	V&S, section 2.2

5 MOCKUPS

Present here some low-fidelity wireframes or mockups of the application. These should be referenced in the functional requirements

5.1.1 MK-1: Search window



Anexo F. TEMPLATE DO DOCUMENTO D2.1.2 – RISK PLAN

Software Project Management 2022/2023

«Project Name»

D2.1.2 – Risk Plan
(Plano de Risco)

Authors (Autores)

- «author name»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

1. THRESHOLD OF SUCCESS / LIMIAR DE SUCESSO

1	By the end of the second quarter, all "Must Have" features are implemented and pass acceptance tests with no known critical defects.
2	All team members give an average score of 5 or better on a job satisfaction survey taken quarterly.

2. RISK LIST/LISTA DE RISCOS

Example of risk list:

Pri	#	P	I	Risk statement		Indicators	Actions
1	RS-2	4	5	The team has no experience in software estimation	might cause underestimation, delaying the project in an uncontrolled way.	Burndown chart revealing low velocity	CP-1
2	RS-1	3	3	Customers fail to understand the impact of requirements changes	requested changes might require major design rework.	Client change requests	Observ.

*Actions may be a **Mitigation Action***

- *Contingency Plan (CP) / Plano de Contingência,*
- *Avoidance Strategy (AS) / Plano de Prevenção or*
- *Minimization Strategy (MS) / Plano de Minimização,*

*detailed in next section, or **Observation**.*

3. MITIGATION ACTIONS

This section may describe the mitigation actions regarding risks with P x I ≥ 20

CP-1 Contingency Plan: Delayed project

If the project gets delayed beyond … then ...

Anexo G. TEMPLATE DO DOCUMENTO D2.1.3 – ACCEPTANCE TEST PLAN

Software Project Management 2022/2023

«Project Name»

D2.1.3 – Acceptance Test Plan (Plano de Testes de Aceitação)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

Table of Contents (Índice)

1. INTRODUCTION

Summary of the application to be tested.

Identify related work products, such as requirements, prototypes, etc.

2. FEATURES TO BE TESTED

A list of the features in the software that will be tested.

3. FEATURES NOT TO BE TESTED

A list of any areas of the software that will be excluded from the test.

4. APPROACH

A description of the strategies that will be used to perform the test.

Examples:

- All tests shall be executed manually by the tester, through mouse and keyboard, and result observation in the monitor.
- Communication with the server shall be tested using a hyperterminal.

Also, a complete description of the test environment or environments. This should include a description of hardware, networking, databases, software, operating systems, and any other attribute of the environment that could affect the test.

5. BASE STATES

Standard procedures to setup testing environment, before running some tests. Notice that these are not tests.

Example:

Name	**BS-12: Load test document TESTDOC.DOC**
Requirements	FR-4 (Case sensitivity in search-and-replace), bullet 2
Preconditions	No user is logged in and no applications are running. The test document TESTDOC.DOC.
Steps	1. Log in as user "joetester" with password "test1234". 2. Launch the application. 3. Select the File/Open menu item. 4. Enter "/usr/home/joetester/TESTDOC.DOC". 5. Click OK.
Expected results	1. Verify that the Open Window dialog box has been dismissed. 2. Verify that the file TESTDOC.DOC is loaded.

	3. Verify that the file TESTDOC.DOC is given focus. 4. Verify that the file TESTDOC.DOC is active.

6. TEST CASES

These are the tests that the tester will execute.

Example:

Name	**TC-01: Verify that lowercase data entry results in lowercase insert**
Requirements	FR-04 (Case sensitivity in search-and-replace), bullet 2
Preconditions	Base state BS-12
Steps	1. Click on the "Search and Replace" button. 2. Click in the "Search Term" field. 3. Enter This is the Search Term. 4. Click in the "Replacement Text" field. 5. Enter This IS THE Replacement TeRM. 6. Verify that the "Case Sensitivity" checkbox is unchecked. 7. Click the OK button.
Expected results	1. The search-and-replace window is dismissed. 2. Verify that in line 38 of the document, the text this is the search term has been replaced by this is the replacement term. 3. Return to base state BS-12.

7. TRACEABILITY MATRIX

Relation between the functional requirements and the test cases.

Example:

TC/FR	01	02	03	04	05											
TC-01				x												
TC-02					x											

Anexo H. TEMPLATE DO DOCUMENTO D2.2.1 – SOFTWARE ARCHITECTURE AND DESIGN

Software Project Management 2022/2023

«Project Name»

D2.2.1 – Software Architecture and Design (Design e Arquitetura de Software)

Authors (Autores)

- «author names»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

Table of Contents (Índice)

This is just a placeholder for your architecture and design. You should present the diagrams as needed for a clear understanding of the software architecture and design.

1. INTRODUCTION

The introduction serves to orient the reader.

2. SYSTEM ARCHITECTURE / CLASS DIAGRAMS / ACTIVITY DIAGRAMS / SEQUENCE DIAGRAMS / STATECHART DIAGRAMS / …

Use the diagrams that better describe your application, so it makes it easier to understand and code.

Anexo I. TEMPLATE DO DOCUMENTO D2.3.1 – ACCEPTANCE TEST REPORT

Software Project Management 2022/2023

«Project Name»

D2.3.1 – Acceptance Test Report
(Relatório de Testes de Aceitação)

Authors (Autores)

- «author name»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

1. FINAL RESULTS

Example:

Test	Date	Tester	P/F	Comments
TC-01	13/12/2020	AV	P	Failed once (12/12/2020)
TC-02	12/12/2020	AV	P	

2. INCIDENTS REPORT

Example:

Test	**TC-01: Verify that lowercase data entry results in lowercase insert**
Date;Tester	**12/12/2020; AV**
Incident	Replaced text in line 38 appeared as *This IS THE Replacement TeRM*
Actions	Check potential bug on replacement routine
Date; Developer	**12/12/2020: FC**
Correction	bug on routine XXX corrected

Anexo J. TEMPLATE DO DOCUMENTO D2.3.2 – QUALITY ASSESSMENT REPORT

Software Project Management 2022/2023

«Project Name»

D2.3.2 - Quality Assessment Report
(Relatório de Avaliação da Qualidade)

Authors (Autores)

- «author name»

Reviewer (Revisto por:)

- « version, reviewer name, preferably not in the author's list »

Approver (Aprovado por:)

- « version, approver's name, preferably not in the author's list »

1. METRICS

The following table resumes all defined metrics, goals and measures for the project.

#	Description/Formula	Goal	Result	Achieved
2.1	External Goal	>8	10	Yes
2.2	Internal Goal	<2	2	No
3	Reviews as planned (real/plan)	100%	66%	No
4	Tests as planned (real/plan)	90%	95%	Yes
5.1	Mitigated Risks	100%	50%	No

5.2	Unanticipated problems	0	0	Yes
6	Coding standards not compliant	0	1	No
7	Documents stability (# change requests)	<2	1	Yes
8	Non-conformities detected	<5	1	Yes
			Achieved/Total	

2. QUALITY GOALS

The following tables describe the metrics related with the Quality Goals.

2.1 External goals

#				
1				
			Result	

2.2 Internal goals

#				
1				
			Result	

3. REVIEWS

The following list describes which work products shall be reviewed, and what kind of procedure will be used.

#	**Work Product**	**Planned review**	**# major defects**	**Done on date**	**Done late**
1	D2.1.1 SRS	Inspection	2	2/12/2021	
2	D2.2.1 - Risk Plan	Deskcheck	5		4/12/2021
3	SW Module A	Inspection	-	-	-

	Total	3	7	1	1

4. TESTS

The following list describes which work products shall be tested, and what kind of procedure will be used.

#	Work Product	Planned test	# errors	Done on date	Done late
1	SW Module A	Unit	0	5/1/2022	
2	SW Module A	Unit	0		8/1/2022
3	Integration of modules A+B	Integration	-	-	-
4	Acceptance	Acceptance	5	12/1/2022	-
	Total	4	5	2	1

5. RISK MANAGEMENT

5.1 Mitigated risks

The following Risks were mitigated because of the risk plan

#	Date	Risk	Mitigation result
1			
			# of mitigated risks: 1

5.2 Unanticipated problems

The following problems were not anticipated

#	Date	Problem	Result
1			
			# of unanticipated problems: 1

6. CODING STANDARDS

The following coding standards were not followed

#	Coding standard	Files	Description
1	1.1		
			# of not conform standards: 1

7. CHANGE CONTROL LIST

The following changes have been requested.

#	Requested change	Proponent	Version	Decision
1	SRS – add requirement 25	Manuel Silva	1.0	Accepted
			# of accepted changes	1

8. NON COMPLIANCE LIST

The following list describes the issues that occurred during the project, which did not comply with the quality plan.

#	Description
1	Change to the test plan without change request submission
	Non-conformities:

Anexo K. TEMPLATE DO DOCUMENTO DX.X.X – MILESTONE MX.X REPORT

Software Project Management 2022/2023

«Project Name»

Dx.x.x – Milestone Mx.x Report
(Relatório de Milestone Mx.x)

Authors (Autores)

- «author name»

Reviewer (Revisor)

- «version, reviewer name, preferably not in the author's list»

Approver (Aprovado por)

- « version, approver name, preferably not in the author's list»

1. GOAL OF STAGE X.X (OBJETIVOS DA FASE X.X)

Describe in one paragraph the goal of this stage.

2. TASKS DONE (TAREFAS REALIZADAS)

(from team log or other sources)

Date	Task	Type*	Effort
	Total (budget: 40)		

* E – engineering; Q – quality; M - management

3. ACHIEVED (RESULTADOS CONSEGUIDOS)

- Dashboard ready, «version»
- D1.1 Vision&Scope published, «version», «link»
- Team Log up-to-date «version», «link»
- ...

4. PLANNED BUT NOT ACHIEVED (PLANEADO MAS NÃO CONSEGUIDO)

- Something - the team decided to postpone to Stage x.x because …

5. ANALYSIS

a. What went well

- We arranged a meeting place and …
- ...

b. What went wrong

- We met online but the network was always down
- ...

c. What we could do differently to improve

- Finding a quiet place to meet every week
- ...

Anexo L. EXEMPLO DO FICHEIRO EXCEL ONDE OS ALUNOS DEVIAM REGISTAR O ESFORÇO

	A	B	C	D	E
1	**Week**	**From**	**To**	**Tasks Done**	**Effort**
2	1	13/out	19/out	Sharepoint and Dashboard preparation.	3
3	2	20/out	26/out		4
4	3	27/out	#####		4,5
5	4	#####	#####		
6	5	#####	#####		
7	6	#####	#####		
8	7	#####	#####		
9	8	01/dez	07/dez		
10	9	08/dez	14/dez		
11	10	15/dez	21/dez		
12	11	22/dez	28/dez		
13	12	29/dez	04/jan		
14	13	05/jan	11/jan		
15	14	12/jan	18/jan		
16				Total	11,5

Anexo M. OBJETIVOS DAS VÁRIAS IES ENTREVISTADAS

IES	Objetivos / Competências pretendidas
ES – FEUP	Familiarizar-se com os métodos de engenharia e gestão necessários ao desenvolvimento de sistemas de software complexos e/ou em larga escala, de forma economicamente eficaz e com elevada qualidade. No final da unidade curricular, os estudantes deverão ser capazes de: descrever os princípios, conceitos e práticas da engenharia de software e do ciclo de vida do software; conhecer e saber aplicar as técnicas e ferramentas necessárias para executar e gerir as várias atividades do processo de desenvolvimento de software de qualidade; explicar os métodos e processos de construção de diferentes tipos de sistemas de software.
ES – FCTUC	O estudante deverá ser capaz de perceber por que razão a complexidade do software exige uma abordagem de engenharia e as diversas formas de organizar as pessoas e atividades para o desenvolvimento de um produto de qualidade, nomeadamente abordagens lineares, iterativas e incrementais. Deverá ainda ser capaz de perceber as diferenças entre elas e escolher a(s) mais indicada(s) em função do contexto concreto do projeto. Deverá também ser capaz de usar técnicas e artefactos genéricos de gestão de projeto (diagramas Gantt e PERT/CPM, análise de riscos...). Finalmente, deverá ser capaz de descrever aspetos centrais do artefacto de software a desenvolver usado a notação UML.

ES – IST	Familiarizar-se com os métodos de engenharia e gestão necessários ao desenvolvimento de sistemas de software complexos e/ou em larga escala, de forma economicamente eficaz e com elevada qualidade. Em particular, nesta UC procura-se transmitir o roteiro do desenvolvimento de software, desde o levantamento de requisitos até à manutenção de programas. Integrar os conhecimentos adquiridos noutras disciplinas no contexto mais alargado do processo de desenvolvimento de software. Motivar para o desenvolvimento de software como uma engenharia, que integra os aspetos tecnológicas da computação com os fatores sociais e humanos da construção de produtos. No final da UC, os estudantes deverão ser capazes de: - descrever os princípios, conceitos e práticas da engenharia de software e do ciclo de vida do software; - conhecer e saber aplicar as técnicas e ferramentas necessárias para executar e gerir as várias atividades do processo de desenvolvimento de software de qualidade.
PSI – FCUL	Esta disciplina representa um dos pontos de consolidação e interligação de forma consistente de diversos temas abordados em unidades curriculares anteriores, concretizadas no desenvolvimento de um sistema de informação baseado na Web. Tem como objetivo apresentar os conceitos fundamentais e diversos tópicos teóricos associados a projetos de desenvolvimento de aplicações e serviços Web, introduzindo-os num contexto de gestão ágil de projetos. Na componente prática a disciplina pretende dotar os estudantes da capacidade de projetar e desenvolver um sistema de informação baseado em tecnologias web.
TAES – ESTG	Objetivos: Conhecimento e compreensão da perspetiva ágil no desenvolvimento de software; Capacidade de relacionar os conceitos das metodologias ágeis de desenvolvimento de software com os conceitos da gestão de projetos tradicional; Capacidade de identificar as principais ferramentas e técnicas utilizadas pelas metodologias ágeis de desenvolvimento de software; Capacidade de aplicar os conceitos, ferramentas e técnicas num ambiente real de desenvolvimento de software segundo uma metodologia ágil;

	Capacidade de gestão de equipas de desenvolvimento software segundo uma perspetiva ágil; Capacidade de identificar, avaliar e resolver os riscos de um projeto de software segundo uma perspetiva ágil.

Anexo N. TUTORIAL DE CONFIGURAÇÃO DO REPOSITÓRIO GITLAB

Gitlab Configuration

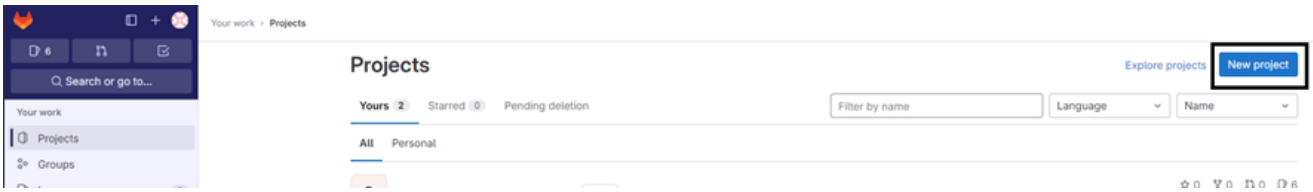
14 September 2023
17:02

1. Install git - https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

2. Register/login at gitlab - https://gitlab.com/

a. If it asks you to create a group, you can. This will create a repository by default. You can delete this repository. To do this, open the repository, click on "Settings -> General", click on "Expand" in the "Advanced" topic and scroll to the bottom. Click on "Delete project".

3. (Just one element of the group) Create the repository:



a. Select “Create blank project”;

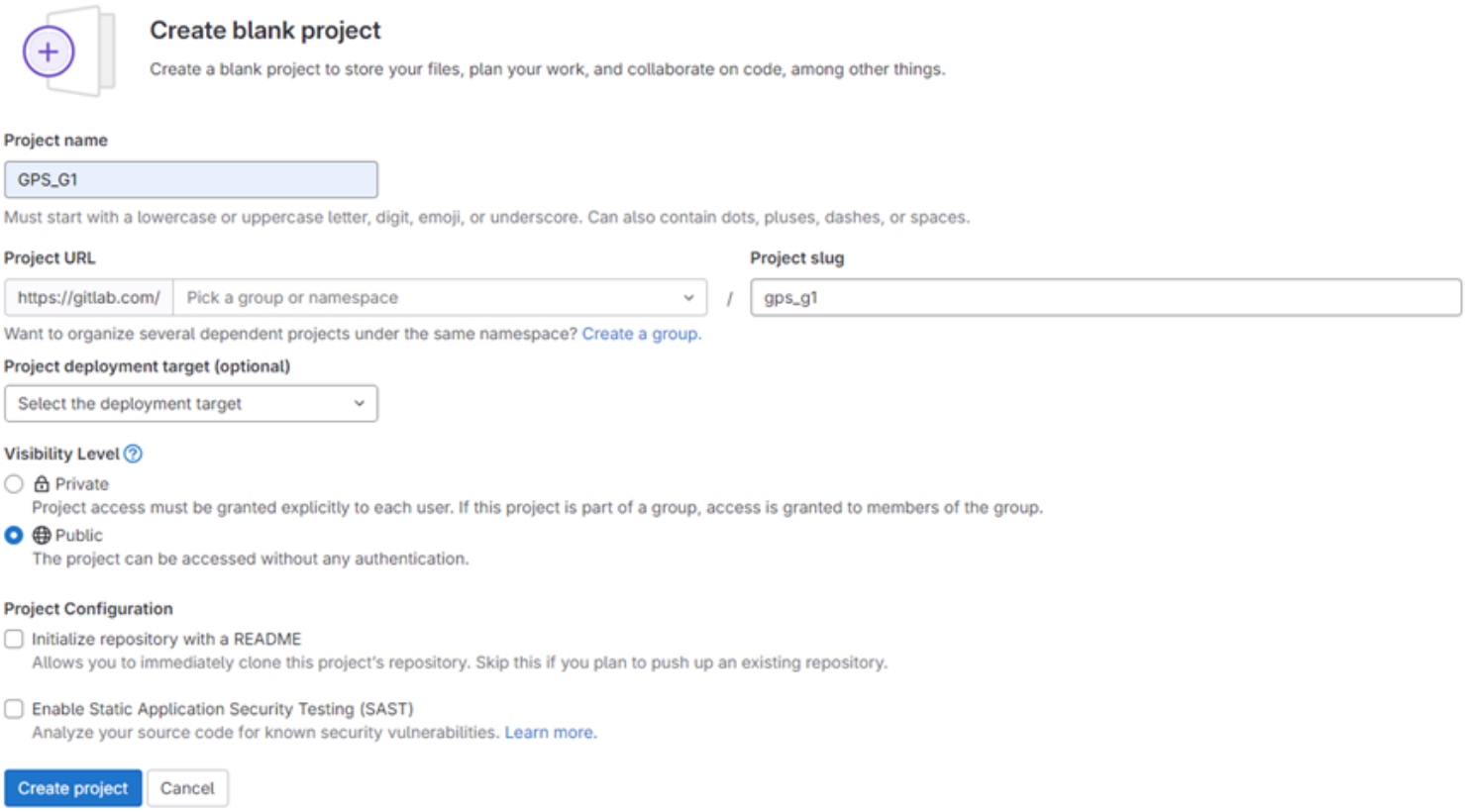
b. At "Pick a group or namespace" dropwdown don’t select your group, select your username.

c. Define repository name. Suggestion: GPS_Gx (group number);

d. Don’t select any “Project deployment target”;

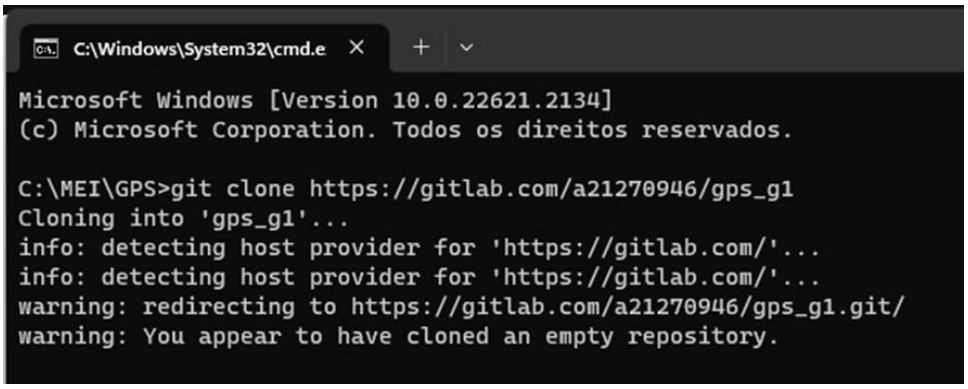
e. Select “Public” in visibility level;

f. Deselect the option “Initialize repository with a README”. The page should be something like this:



4. (Just one element of the group) Now, you should invite all members of the group.

5. Next, clone the project. Open a command line in the folder where you want to save the project and run the command "git clone <url of the repository>"



a. If you have problems doing this, it should be authentication / ssh keys problems

6. Move to the directory of the project.

7. (Just one element of the group) Copy the file “READ.ME” and “.gitignore” that is available in Teams files. Be aware that downloading the .gitignore file from teams can remove the '.' (dot) at the beginning of the filename. The file should really be called ".gitignore"

8. (Just one element of the group) Then, you should push the files to repository:

   a. git add .

   b. git commit -m “read me and gitignore added” (for example)

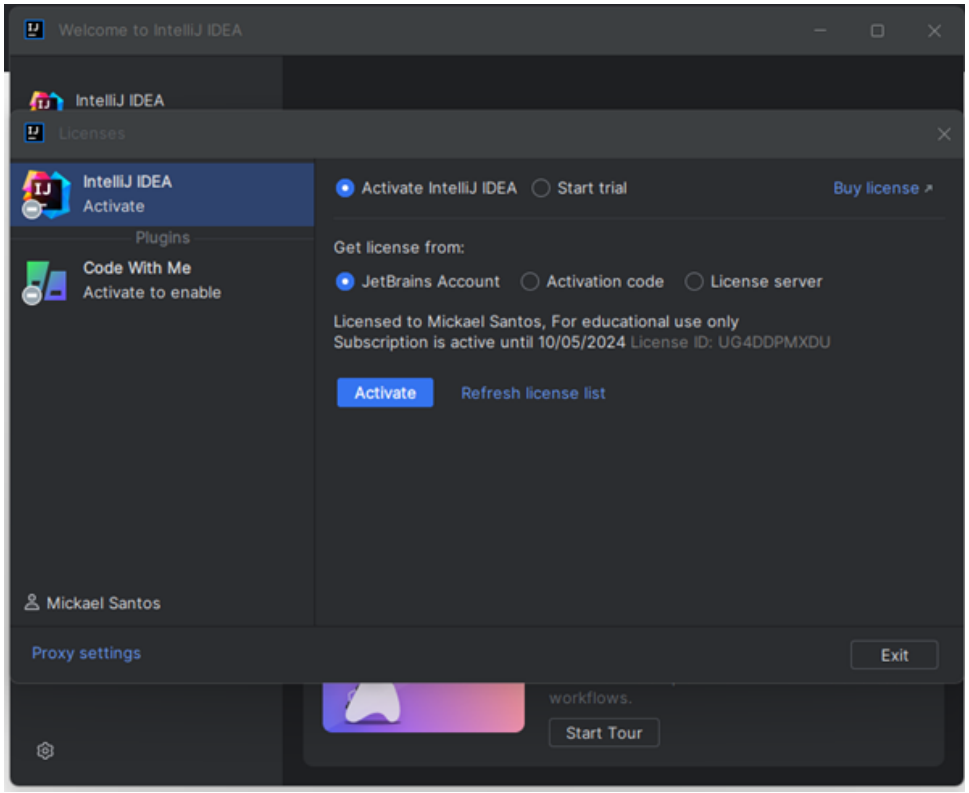
   c. git push origin main

   d. The other elements should do "git pull" (If you have problems doing this, it should be authentication / ssh keys problems)

9. Then, you will have to install IntelliJ Ultimate. If you already have IntelliJ Ultimate installed, go to step 12. You can use the isec email to get the ultimate version for free - https://www.jetbrains.com/idea/download

10. Once you've installed IntelliJ, you'll get a popup asking you to login or register your purchase key. Click on "Log in to Jetbrains Account", and you'll be redirected to a web page.

11. Once you have finished registering in JetBrains, the popup should look like this, and you must click on “Activate” button:



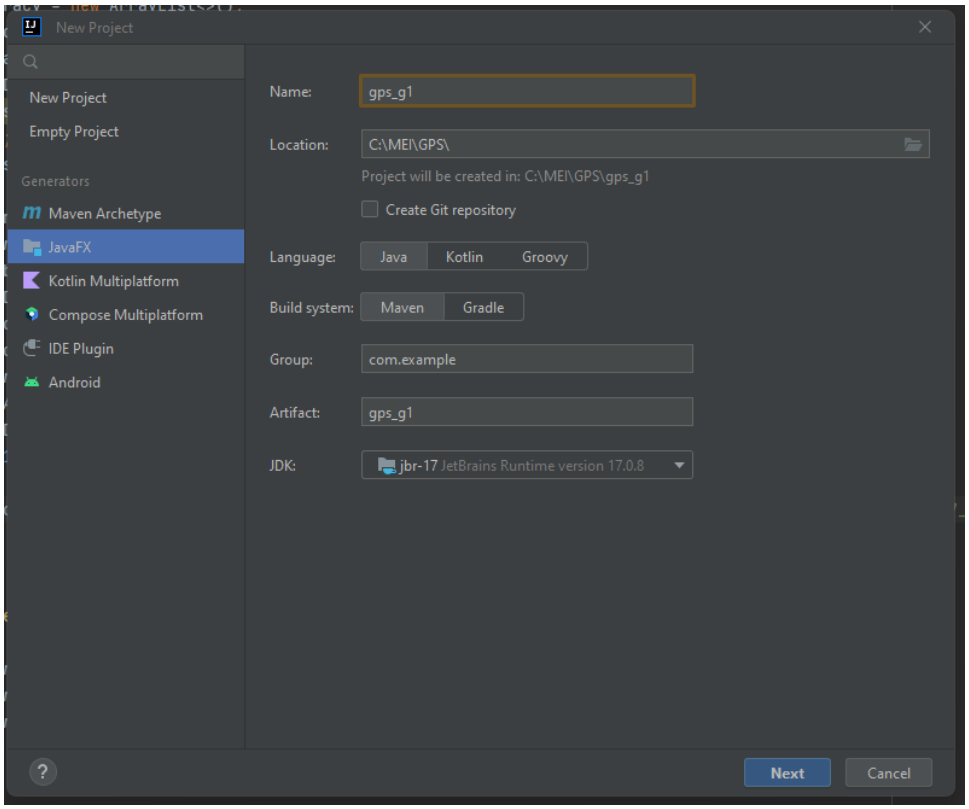
12. (Just one element of the group) Just Then, you must create project:

a. You will have to create the project in the same directory as the repository folder that you saved before. For example, if your repository is called "gps_g1", you have to name your project with "gps_g1" and select the location of that folder (look image at d.);

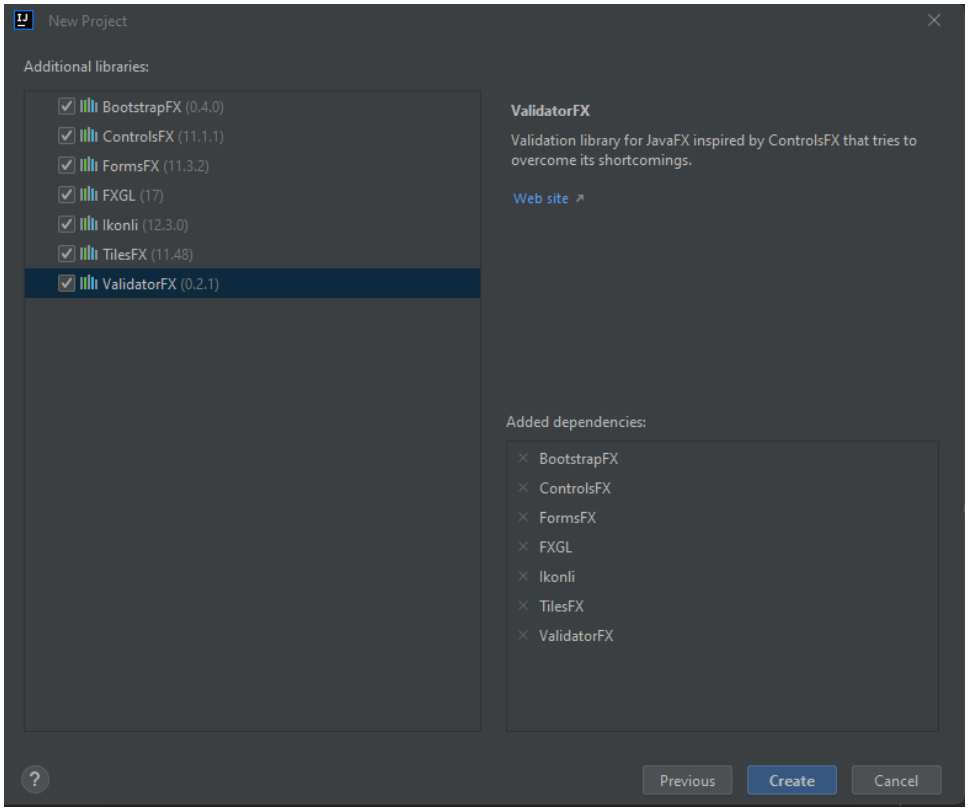
b. Select “Maven”;

c. Select Java version 17. If you don’t have sdk 17, you can download it by clicking on “Add SDK – Download JDK” and select “JetBrains Runtime 17”;

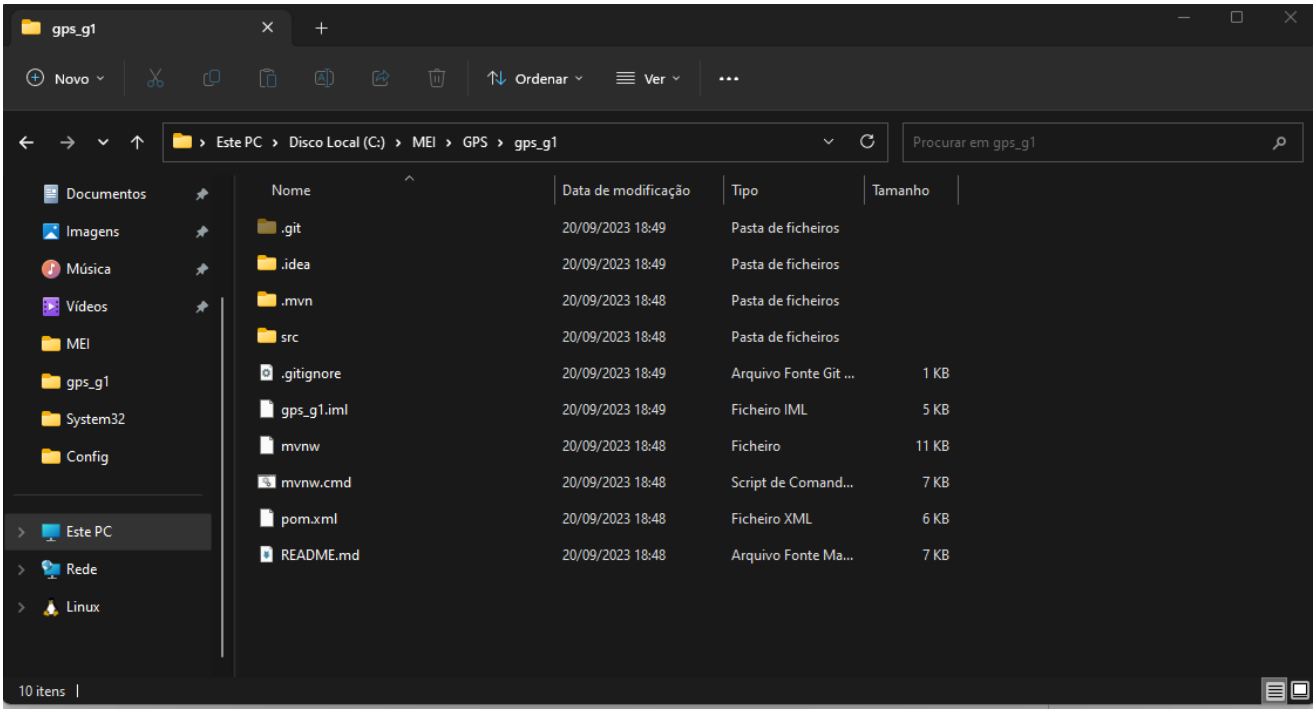
d. It will shows you a popup saying that directory is not empty. Just click “ok”



e. Select all checkboxes and click "Create"



f. After that, your folder should look like this:



13. (Just one element of the group) Copy the folder “javafx-sdk” that is available in Teams files and paste in root of your project.

14. (Just one element of the group) Then, you should push the project into gitlab:

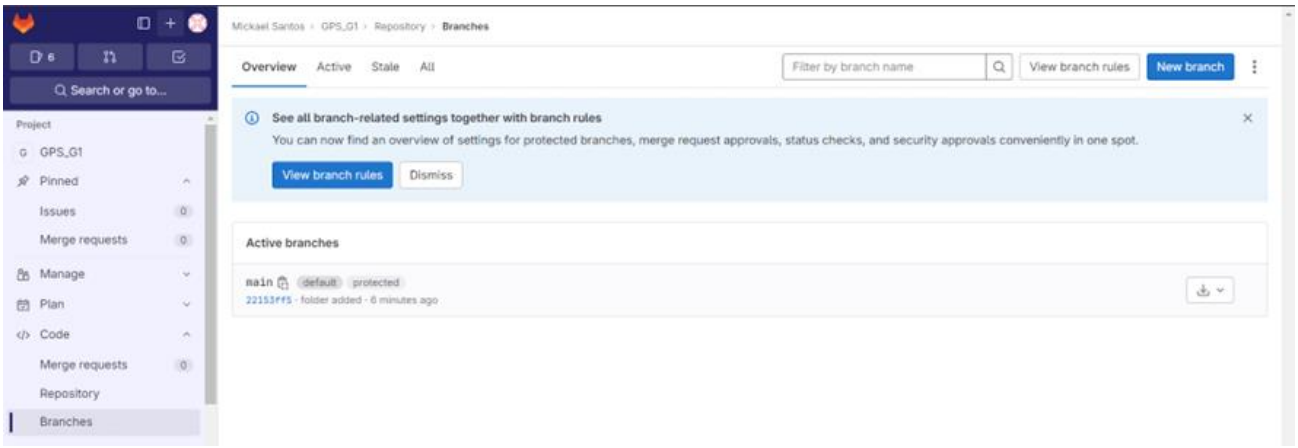
    a. git add .

    b. git commit -m “code added” (for example)

    c. git push origin main

    d. The other elements should do "git pull"

15. (Just one element of the group) Then, you should create a branch named “dev” and another one named "qa". To do that, you can go to your repository, click on “Code –> Branches” and click in “New Branch”.

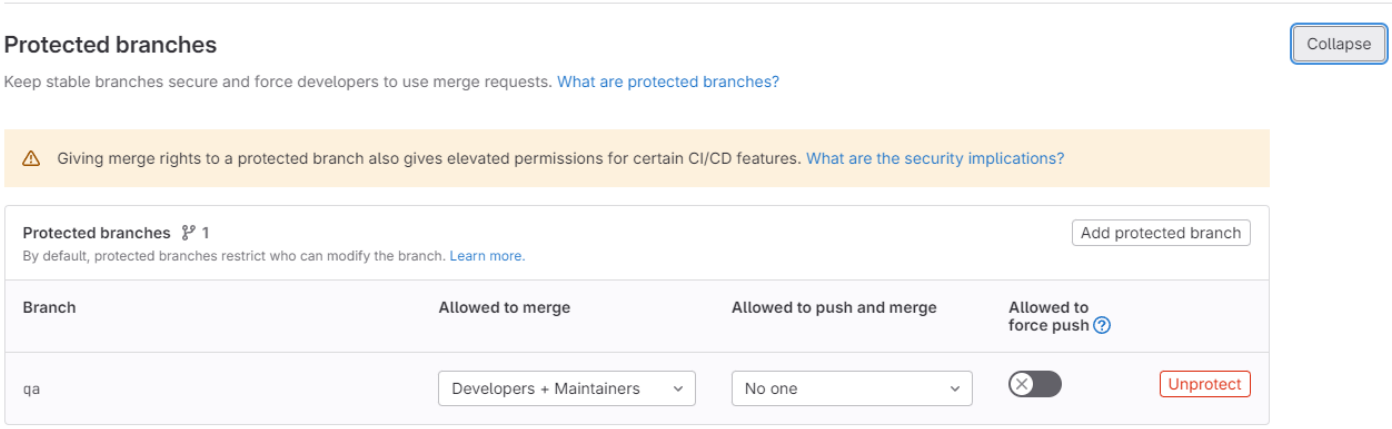


16. (Just one element of the group) Then, you must protect your “qa” branch. To do that, you can click on “Settings -> Repository”, expand “Protected branches”.

    a. Select “Developers + Maintainers” on “Allowed to Merge”;

    b. Select “No one” on “Allowed to push and merge”;

    c. Set “Allowed to force push” switch to false.



17. (Just one element of the group) Then, you must go to “Settings -> Merge requests”, deselect the option “Enable "Delete source branch" option by default” and select "Save Changes" (THIS IS VERY IMPORTANT).

18. Then, you can test your repository. Checkout to the “dev” branch in your terminal, try making a simple change in some file (for example, change the program's initial message to Hello World GPS) and commit:

a. git pull

b. git checkout dev

c. make a simple change

d. git add .

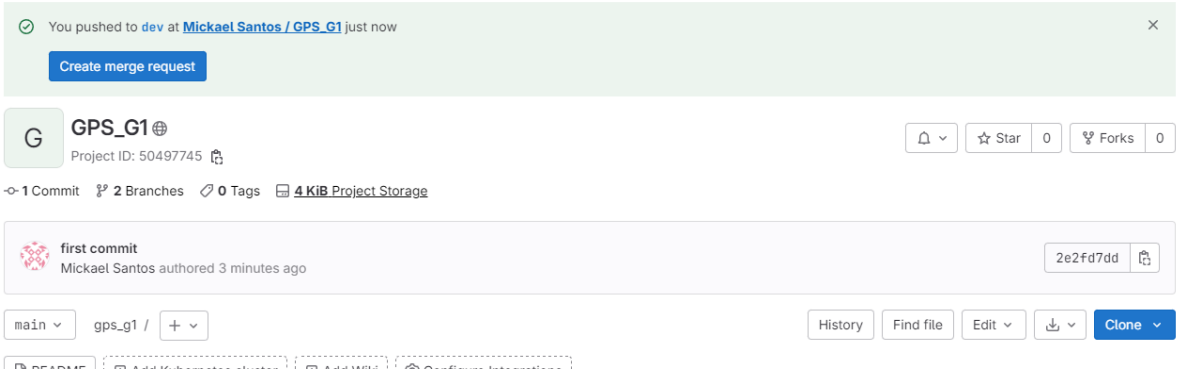
e. git commit -m “test repo”

f. git push origin dev

g. The other elements should do "git pull" and "git checkout dev"

19. Then, create a merge request in Gitlab, assign to someone. That someone should accept and merge the code.

a. You can go to GitLab, and a popup to create a merge request will appear:



b. Click on "Create Merge Request". Merge the code into "qa" branch. Assign to someone else;

c. Make sure that option "Delete source branch when merge request is accepted.." Is **not** checked;

d. Click on "Create merge request".

20. Merge the code:

a. The person who you assigned can merge the code by clicking on "Merge" button.

21. Everyone should checkout to qa branch, do "git pull" and then return to dev branch, doing "git checkout dev"

22. To practice, you can open a new merge request from "dev" branch to "qa" branch.

Anexo O. TUTORIAL DA CRIAÇÃO DA MÁQUINA VIRTUAL

Virtual machine

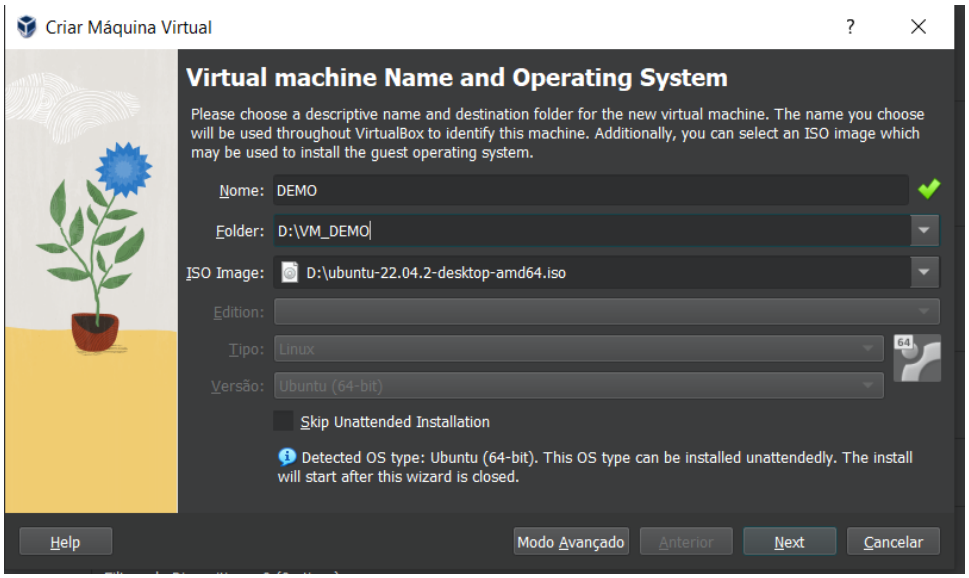
30 de setembro de 2023
13:32

1 - Install virtualbox - https://www.virtualbox.org/wiki/Downloads

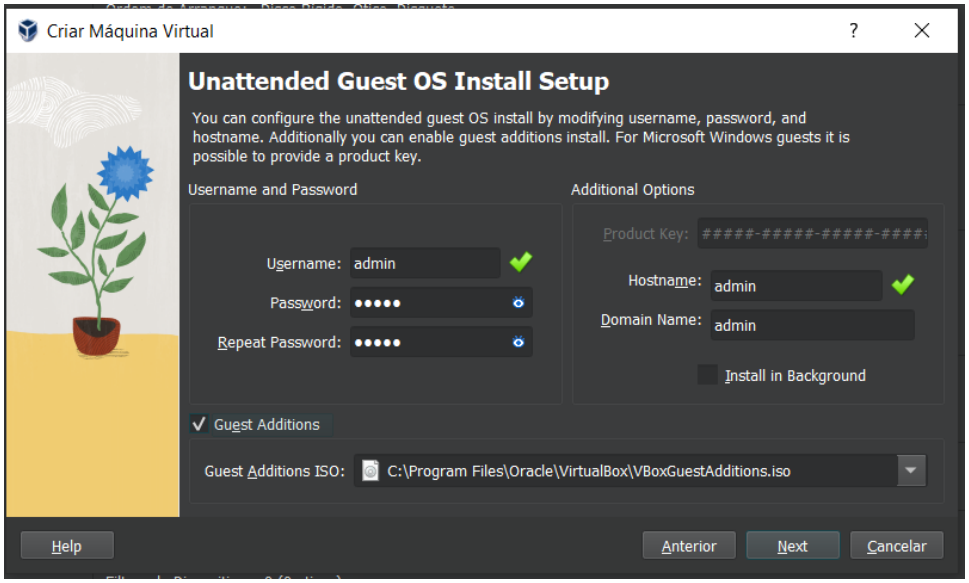
2- Download Ubuntu 22.04 iso file - https://ubuntu.com/download/desktop

3 - Create virtual machine

a. You must select the iso image you downloaded



b. Choose your username
c. Select "Guest Additions"



d. Select default hardware values

4 – Install ansible (after terminating VM installation) by running:

a. sudo apt update
    a. If you have the error "<user> is not in sudoers file..." you must follow this process - https://stackoverflow.com/a/47807036
b. sudo apt install software-properties-common
c. sudo add-apt-repository --yes --update ppa:ansible/ansible
d. sudo apt install ansible
e. ansible –version – just to check if everything is ok

Anexo P. TEMPLATE DE READ.ME A UTILIZAR NO REPOSITÓRIO GITLAB

```
# Project Title

## Contents

- [Team](#team)
- [Vision and Scope](#vision-and-scope)
- [Requirements](#requirements)
    - [Use case diagram](#use-case-diagram)
    - [Mockups](#mockups)
    - [User stories](#user-stories)
- [Definition of Done](#definition-of-done)
- [Architecture and Design](#architecture-and-design)
    - [Domain Model](#domain-model)
- [Risk Plan](#risk-plan)
- [Pre-Game](#pre-game)
- [Release Plan](#release-plan)
  - [Release 1](#release-1)
  - [Release 2](#release-2)
- [Increments](#increments)
  - [Sprint 1](#sprint-1)
  - [Sprint 2](#sprint-2)
  - [Sprint 3](#sprint-3)

## Team

- Student A
- Student B
```

```
- Student C


***


## Vision and Scope


#### Problem Statement


##### Project background


This section contains a summary of the problem that the project will solve. It should provide a brief history of the

problem and an explanation of how the organization justified the decision to build software to address it. This section

should cover the reasons why the problem exists, the organization's history with this problem, any previous projects

that were undertaken to try to address it, and the way that the decision to begin this project was reached.


##### Stakeholders


This is a bulleted list of the stakeholders. Each stakeholder may be referred to by name, title, or role ("support group

manager," "CTO," "senior manager"). The needs of each stakeholder are described in a few sentences.


##### Users


This is a bulleted list of the users. As with the stakeholders, each user can either be referred to by name or role ("
```

```
support rep," "call quality auditor," "home web site user”)
however, if there are many users, it is usually inefficient

to try to name each one. The needs of each user are described.


***


#### Vision & Scope of the Solution


##### Vision statement


The goal of the vision statement is to describe what the
project is expected to accomplish. It should explain what the

purpose of the project is. This should be a compelling reason,
a solid justification for spending time, money, and

resources on the project. The best time to write the vision
statement is after talking to the stakeholders and users and

writing down their needs; by this time, a concrete
understanding of the project should be starting to jell.


##### List of features


This section contains a list of features. A feature is as a
cohesive area of the software that fulfills a specific need

by providing a set of services or capabilities. Any software
package, in fact, any engineered product, can be broken

down into features. The project manager can choose the number
of features in the vision and scope document by changing

the level of detail or granularity of each feature, and by
combining multiple features into a single one. Sometimes

those features are small ("screw-top cap," "holds one liter of
liquid"); sometimes they are big ("four-wheel drive," "

seats seven passengers"). It is useful to describe a product
in about 10 features in the vision and scope document,

because this usually yields a level of complexity that most
people reading it are comfortable with. Adding too many
```

features will overwhelm most readers.

Each feature should be listed in a separate paragraph or bullet point. It should be given a name, followed by a

description of the functionality that it provides. This description does not need to be detailed; it can simply be a few

sentences that give a general explanation of the feature. However, if there is more information that a stakeholder or

project team member feels should be included, it is important to include that information. For example, it is sometimes

useful to include a use case (see Chapter 6), as long as it is written in such a way that all of the stakeholders can

read and understand it.

Features that will not be developed

Features are often left out of a project on purpose. When a feature is explicitly left out of the software, it should be

added to this section to tell the reader that a decision was made to exclude it. For example, one way to handle an

unrealistic deadline is by removing one or more features from the software, in which case the removed features should be

moved into this section. The reason these features should be moved rather than deleted from the document is that

otherwise, readers might assume that they were overlooked and bring them up in a review. This is especially important

during the review of the document because it allows everyone to agree on the exclusion of the feature (or object to it).

Assumptions

This is the list of assumptions that the stakeholders, users, or project team have made. The team should hold a

```
brainstorming session to come up with a list of assumptions.
(See Chapter 3 for more information on assumptions.)


***


## Requirements


### Use Case Diagram


![Use case diagram](imgs/UML_use_case_example-800x707.png)


***


### Mockups


***


### User Stories


- User story 1 (link to issue card)
- User story 2 (link to issue card)
- User story 3 (link to issue card)


***


## Definition of done


(This section is already written, do not edit)

It is a collection of criteria that must be completed for a
User Story to be considered “done.”
```

```
1. All tasks done:
  - CI – built, tested (Junit), reviewed (SonarCloud)
  - Merge request to qa (code review)
2. Acceptance tests passed
3. Accepted by the client
4. Code merged to main



***


##### User Story 2


***


##### User Story 3


***


## Architecture and Design


#### Domain Model


A domain model should be here. You can see in (#use-case-
diagram) how to import an image.


***


## Risk Plan
```

```
##### Threshhold of Success

- By the 2nd release date, 30% of all the User Stories from
the Product Backlog are closed. (for example)

- Goal2

- Goal3


##### Risk List

- RSKn – PxI: axb=c; Risk statement

- RSK2 – PxI: 4x5=20; The team has no experience in software
estimation, therefore there might cause underestimation,
delaying the project in an uncontrolled way (for example)




##### Mitigation Actions (threats>=20)

- RSKn – CP/AS/MS; Contingency Plan (CP), Avoidance Strategy
(AS) or Minimization Strategy (MS)

- RSK2 – MS; At the middle of each sprint, compare the
progress of the project with the estimation and, if necessary,
review the sprint plan. (for example)


***


## Pre-Game

### Sprint 0 Plan


- Goal: description

- Dates: from 10-13/Oct to 24-27/Oct, 2 weeks

- Sprint 0 Backlog (don't edit this list):

  - Task1 – Write Team

  - Task2 – Write V&S

  - Task3 – Write Requirements
```

```
  - Task4 – Write DoD

  - Task5 – Write Architecture&Design

  - Task6 – Write Risk Plan

  - Task7 – Write Pre-Gane

  - Task8 – Write Release Plan

  - Task9 – Write Product Increments

  - Task10 – Create Product Board

  - Task11 – Create Sprint 0 Board

  - Task12 – Write US in PB, estimate (SML), prioritize
(MoSCoW), sort

  - Task13 – Create repository with “GPS Git” Workflow


***


## Release Plan


### Release 1


- Goal: MVP - description

- Dates: [teams 0] 21-24/Nov | [teams1] 28-30/Nov

- Release: V1.0


***


### Release 2


- Goal: Final release – description

- Date: [teams 0+1] 12-15/Dec

- Release: V2.0
```

```
***

## Increments

### Sprint 1
##### Sprint Plan

- Goal: what's the goal for this sprint

- Dates: from 24-27/Oct to 7-10/Nov | 14-17/Nov, 2 | 3 weeks

- Roles:
  - Product Owner: name
  - Scrum Master: name

- To do:
  - (list of US or Tasks from the PB)
  - US1: As … I want … so that …
  - Task1: Some task

- Story Points: 2S+3M+3X+2H

- Analysis: short analysis of the planning meeting

##### Sprint Review

- Analysis: what was not done or what was added (Link to US or
Task from the PB)
```

```
- Story Points: 2S+1M+2X+2H

- Version: 0.1

- Client analysis: client feedback

- Conclusions: what to add/review

##### Sprint Retrospective

- What we did well:
    - A
- What we did less well:
    - B
- How to improve to the next sprint:
    - C

***

#### Sprint 2

***

#### Sprint 3

***
```

Anexo Q. Template de pipeline CI a utilizar no repositório GitLab

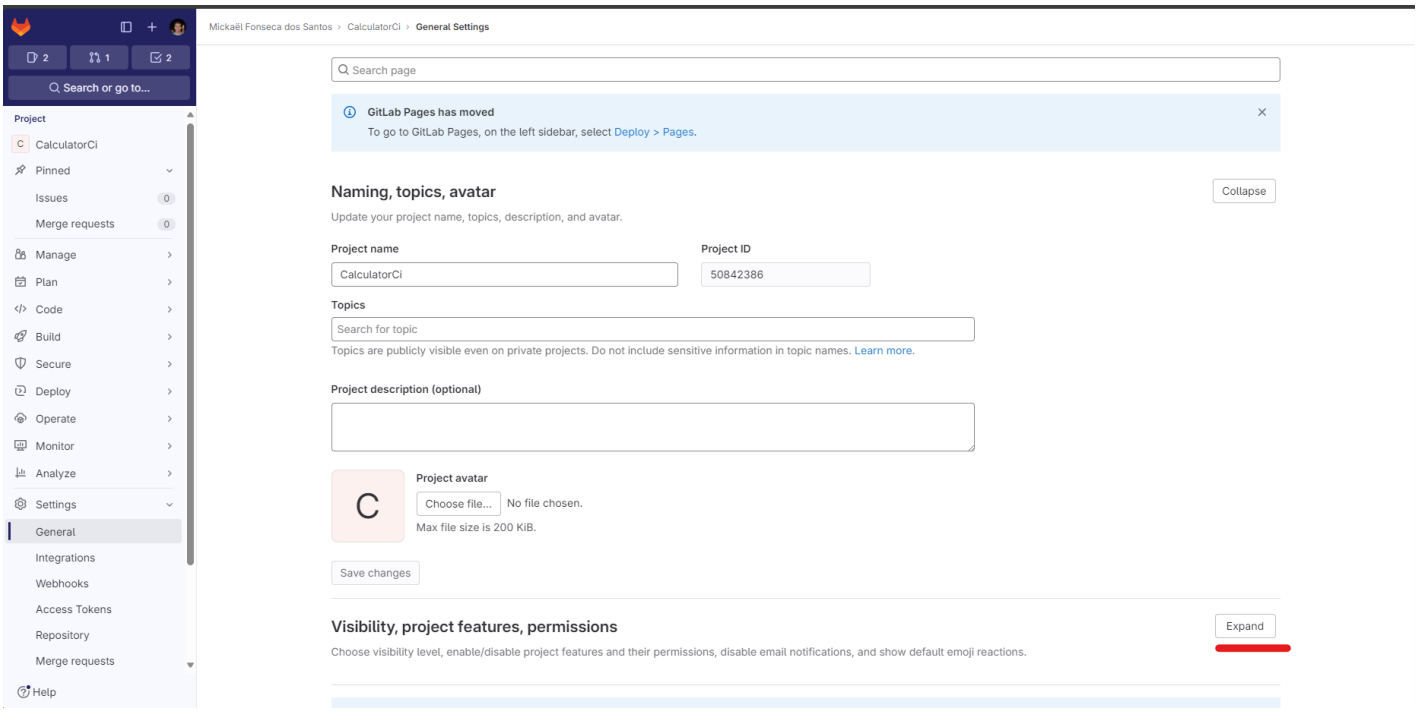
```
stages:
  - test
  - build

variables:
  CI_REGISTRY_IMAGE: "<repository_name>"
  CI_REGISTRY_USER: "<user>"

test:
  stage: test
  script:
    - echo "Testing"

sonarcloud-check:
  stage: test
  image: maven:3.8.4-openjdk-17-slim
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - mvn verify sonar:sonar -
Dsonar.projectKey=mickaelfsantos_cicd

build_and_deploy_image:
  stage: build
  image: docker:stable
  services:
    - docker:dind
  script:
    - docker login -u $CI_REGISTRY_USER -p
$CI_REGISTRY_PASSWORD
    - export DOCKER_TAG=$CI_COMMIT_REF_SLUG-$CI_PIPELINE_ID
    - docker build -t $CI_REGISTRY_IMAGE:$DOCKER_TAG .
    - docker tag $CI_REGISTRY_IMAGE:$DOCKER_TAG
$CI_REGISTRY_IMAGE:latest
    - docker push $CI_REGISTRY_IMAGE:$DOCKER_TAG
    - docker push $CI_REGISTRY_IMAGE:latest
```

Anexo R. TUTORIAL DA CONFIGURAÇÃO DO SONARCLOUD

SonarCloud

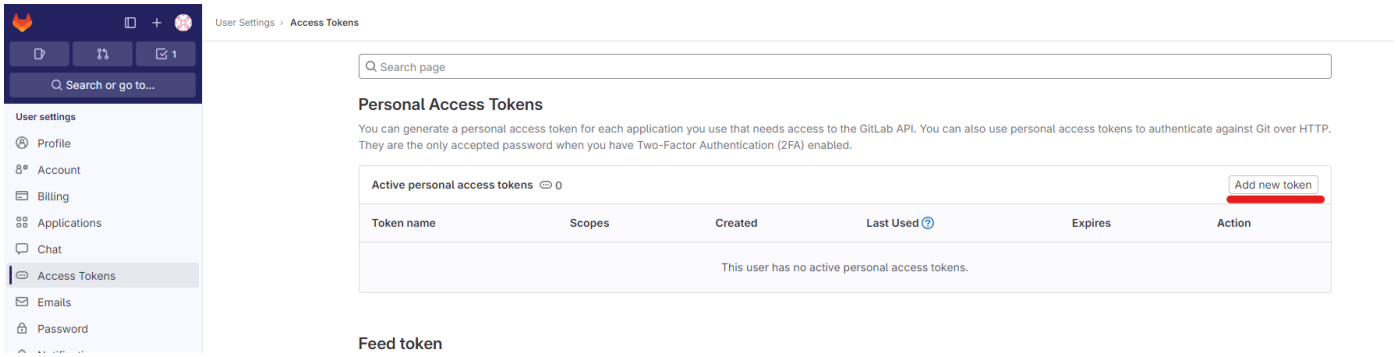
4 de outubro de 2023
19:24

1. For those who created the private repository in class, to practice continuous integration, you should make it public
    a. Open your repository
    b. Click on Settings – General
    c. Expand tab "Visibility, project features, permissions"

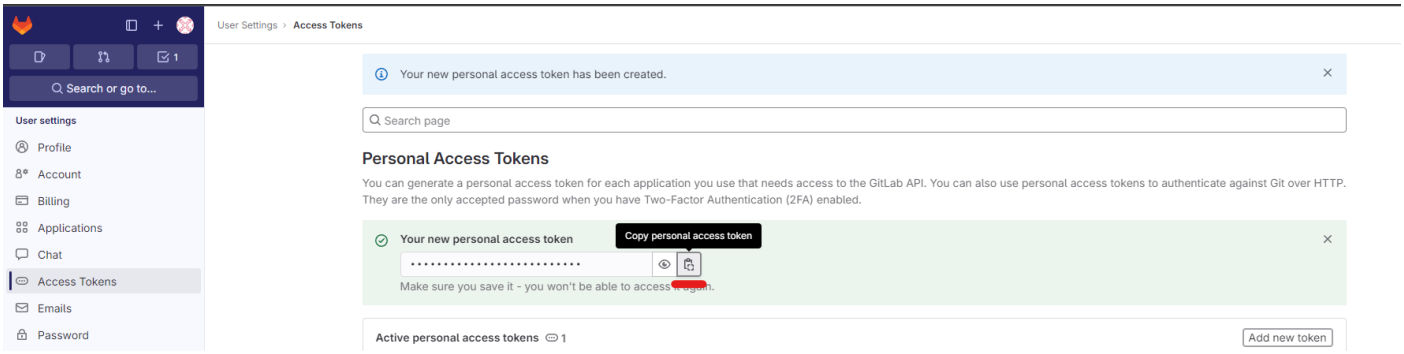


    d. Set project visibility to Public
    e. Scroll down and click on "Save changes" button

2. Navigate to https://sonarcloud.io/login
3. Login with Gitlab
4. Click on red button "Authorize"
5. Click on "Import an organization from GitLab"
6. Select "Import my personal Gitlab group"
7. Click on "Personal Access Token"
8. In the new Tab, click on "Add new token"



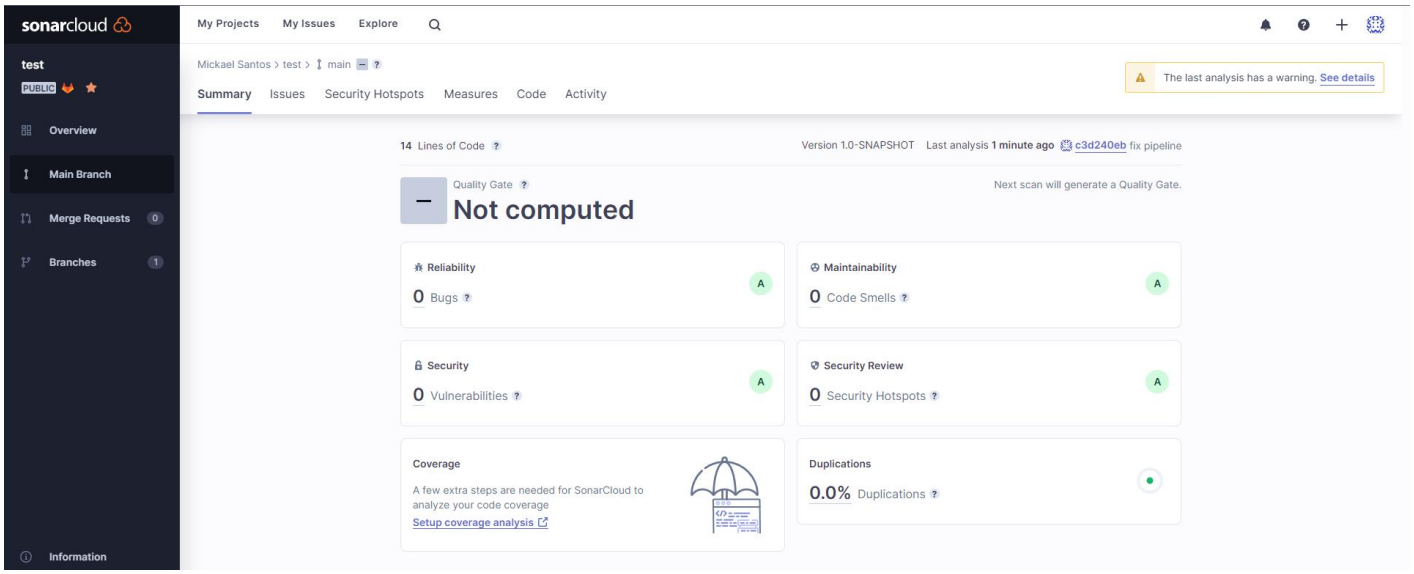
9. Give the name "SonarCloud" (for example) and remove expiration time
10. Check "api"
11. Click on "Create personal access token"
12. Click on the button to copy access token. You can not copy than later!



13. Paste the token on SonarCloud and click on "Continue"
14. Select "Free Plan" and click on "Create organization"
15. Now click on "Analyze a new Project"
16. Select the repository that you created in your lab and click on "Set Up"
17. Select "Previous version" and "Create Project"
18. Select "With GitLab CI/CD Pipeline"
19. Now just follow the tutorial that SonarCloud shows you
20. Just do a litle change in you .gitlab-ci.yml file:
    a. Add stage "test" to sonar cloud job
    b. Remove "only" tag

```yaml
sonarcloud-check:
  image: maven:3-openjdk-11 # Or newer
  stage: test
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - mvn verify sonar:sonar -Dsonar.projectKey=[redacted]
```

21. Then commit changes and check the results on SonarCloud



Anexo S. TEMPLATE DA PIPELINE CD A UTILIZAR NO REPOSITÓRIO GITLAB

Playbook.yml:

```
---
- name: Install/Update App
  hosts: studentsHosts
  connection: local
  gather_facts: false
  become: true

  tasks:
    - name: Install Docker
      become: true
      become_user: "{{ become_user }}"
      apt:
        name: docker.io
        state: present
      when: not docker_installed.stat.exists

    - name: Add user to docker group
      user:
        name: "{{ ubuntu_user }}"
        groups: docker
        append: yes
      become: true
      become_user: "{{ become_user }}"

    - name: Restart Docker service
      service:
        name: docker
        state: restarted
      become: true
      become_user: root

    - name: Install Xorg
      apt:
        name: xorg
        state: present
      when: not xorg_installed.stat.exists

    - name: Run xhost + command
      command: xhost +
      become: true
      become_user: "{{ become_user }}"
      become_method: su

    - name: Install pip3
```

```
      apt:
        name: python3-pip
        state: present
      become: true

    - name: Install Docker Python library
      pip:
        name: docker
        executable: pip3
        state: present
        extra_args: "--upgrade"
      become: true
      become_user: "{{ become_user }}"

    - name: Run Docker container
      docker_container:
        name: gps_app
        image: "{{ docker_image }}"
        detach: yes
        network_mode: host
        env:
          DISPLAY: ":1"

Hosts.inv:
[studentsHosts]
localhost
```

Anexo T. TUTORIAL DA CRIAÇÃO DAS PIPELINES DO PROJETO

Project Pipeline

25 de outubro de 2023
21:12

Updated date: November 18th

Just one element of the group:

1. In your pom.xml, add this inside tag <build>, <plugins>

```
<plugin>
     <groupId>org.codehaus.mojo</groupId>
     <artifactId>versions-maven-plugin</artifactId>
     <version>2.8.1</version>
     <configuration>
         <generateBackupPoms>false</generateBackupPoms>
     </configuration>
</plugin>
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-shade-plugin</artifactId>
     <version>3.2.4</version>
     <executions>
          <execution>
          <phase>package</phase>
          <goals>
               <goal>shade</goal>
          </goals>
          <configuration>
               <transformers>
                                   <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestReso
urceTransformer">
                                            <mainClass>'your main
class'</mainClass>
                                   </transformer>
                          </transformers>
                  </configuration>
                   </execution>
     </executions>
</plugin>
<plugin>
     <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
                  <release>17</release>
                        <source>17</source>
                     <target>17</target>
     </configuration>
</plugin>
```

```
<plugin>
     <groupId>org.openjfx</groupId>
     <artifactId>javafx-maven-plugin</artifactId>
     <version>0.0.8</version>
     <configuration>
          <mainClass>'your main class'</mainClass>
     </configuration>
</plugin>
```

2. In your pom.xml, add this inside <project> tag:

```
<repositories>
     <repository>
          <id>gitlab-maven</id>
          <!--suppress UnresolvedMavenProperty -->
          <url>https://gitlab-ci-
token:${CI_JOB_TOKEN}@gitlab.com/api/v4/projects/${CI_PROJECT_ID}/pa
ckages/maven</url>
     </repository>
</repositories>
<distributionManagement>
     <repository>
          <id>gitlab-maven</id>
          <!--suppress UnresolvedMavenProperty -->
          <url>https://gitlab-ci-
token:${CI_JOB_TOKEN}@gitlab.com/api/v4/projects/${CI_PROJECT_ID}/pa
ckages/maven</url>
     </repository>
     <snapshotRepository>
          <id>gitlab-maven</id>
          <!--suppress UnresolvedMavenProperty -->
          <url>https://gitlab-ci-
token:${CI_JOB_TOKEN}@gitlab.com/api/v4/projects/${CI_PROJECT_ID}/pa
ckages/maven</url>
     </snapshotRepository>
</distributionManagement>
```

3. Download the files ".gitlab-ci.yml" and "ci_settings.xml" that is available at "Files -> Internal Resources -> Templates"

4. Copy it and paste into project folder

5. Create Personal Access Token:
    a. Click on your picture in the top left corner
    b. Click on "Preferences"
    c. Click on "Access Tokens"
    d. Click on "Add new token"
    e. Give a name (for example Project Pipeline)
    f. Remove Expiration time
    g. Check all the permissions
    h. Click on "Create personal access token"
    i. Don't close window because you will need that token
6. Open your repository in another tab

7. Create variables:
    a. Click on Settings
    b. Click on CI/CD
    c. Expand "Variables"
    d. Click on "Add variables"
    e. Input "CI_JOB_TOKEN" on key
    f. Input the token that was created in 4.)
    g. Check "Mask variable" and "Expand variable reference"
    h. Click on "Add variable"

8. Now you can commit the changes to your repository. Then, create a merge request into qa and verify that pipeline is triggered.

9. At the end of executing the pipeline, go to "Deploy -> Package Registry" and check if jar file deployment was successful

10. Download the jar file and run the application by running the command java --module-path /path/to/javafx/lib --add-modules javafx.controls -jar your-app-name.jar

Note: you can delete Dockerfile, image in container registry and javafx-sdk folder
If you have any doubts about some file, you can check this repository:
https://gitlab.com/mickaelfsantos/gitlabregistry

Anexo U. TEMPLATE DA SEGUNDA PIPELINE CI A UTILIZAR NO REPOSITÓRIO GITLAB

```
stages:
  - build
  - test

build:
  stage: build
  image: maven:3.8.1-openjdk-17-slim
  script:
    - export NEW_VERSION=${CI_COMMIT_SHA:0:8}-
${CI_COMMIT_REF_NAME}
    - mvn versions:set -DnewVersion=1-$NEW_VERSION
    - mvn -s ci_settings.xml deploy
  only:
    - qa
    - main

unit-test-job:
  stage: test
  script:
    - echo "Replace with the command to run unit tests"
```

Ficheiro ci_settings.xml utilizado na pipeline acima:

```
<settings xmlns=http://maven.apache.org/SETTINGS/1.1.0
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd">
    <servers>
        <server>
            <id>gitlab-maven</id>
            <configuration>
                <httpHeaders>
                    <property>
                        <name>Job-Token</name>
                        <value>${CI_JOB_TOKEN}</value>
                    </property>
                </httpHeaders>
            </configuration>
        </server>
    </servers>
</settings>
```

Anexo V. ARTIGO CIENTÍFICO

From Traditional to Agile Methodologies in Software Project Management Education: A Case Study

Mickaël Santos

Polytechnic University of Coimbra, Rua da Misericórdia, Lagar dos Cortiços, S. Martinho do Bispo, 3045-093 Coimbra, Portugal, mickael.f.santos@isec.pt

Ricardo Filipe

Polytechnic University of Coimbra, Rua da Misericórdia, Lagar dos Cortiços, S. Martinho do Bispo, 3045-093 Coimbra, Portugal, ricardo.filipe@isec.pt

João Cunha

Polytechnic University of Coimbra, Rua da Misericórdia, Lagar dos Cortiços, S. Martinho do Bispo, 3045-093 Coimbra, Portugal, jcunha@isec.pt

Nowadays, agile methodologies are being increasingly adopted in the software development industry, replacing traditional methodologies. In this way, software engineering courses have been following the industry, and are therefore increasingly teaching students to follow agile methodologies and practices rather than traditional ones. This paper describes and analyzes this transition in a Software Project Management course at a higher-education institution. This experiment took place over two academic years, with Waterfall being used in the first year and Scrum in the second. The Learning Outcomes in both years are the same: to gain competences in managing software projects in small teams; but the steps to reach these competences changed according to the current trends in the area. The results obtained by the students show that, by following Scrum, the students demonstrated being more capable of developing software in teams, focused on the clients, and acquired more knowledge in fundamental areas of software development.

CCS CONCEPTS • **Software and its engineering~Software creation and management~Software development process management~Software development methods~Agile software development / Waterfall model**

Additional Keywords and Phrases: Software Development Methodologies, Software Project Management Education, Traditional to Agile

ACM Reference Format:

1 INTRODUCTION

Agile methodologies have been a trend on the raise over the last 20 years in an attempt to solve the fact that traditional methodologies, such as Waterfall, are too rigid [1] and do not adapt to the rapid changes in markets and users, since changing requirements are not welcome once the contract has been signed [2]. According to the State of Agile report [3], organizations increased their adoption of Scrum from 57% to 87% from 2019 to 2022. In addition, in 2020, Scrum underwent minor changes to its processes and practices, showing that it is a methodology that is globally accepted by the industry, and continuously evolving [4].

At the Polytechnic University of Coimbra, a Software Project Management (SPM) course of a first-cycle programme in computer science still addressed traditional methodologies due to pedagogical reasons, since the students had no experience in managing more complex projects, and they would benefit from more rigorous

processes, well-defined requirements and budgets, from the start of the project [52]. So, to meet the expectations of the industry about the competencies of the graduates, a decision was made to reformulate the course, adopting an agile methodology, Scrum in this case due to its popularity. At the same time, it became necessary to verify if such a transition would not compromise the main competencies that the students should acquire in software project management.

In our work, we first conducted a small survey about the software development methodologies mostly used by nine of the most influential companies regional and national wide, and the results unanimously pointed to the use of agile over traditional. A parallel survey focused nine high-education schools, and again all of them privileged agile as the software development methodology for the students to learn.

We started with the students from the first semester of the academic year of 2022/2023 (“Year 1”, from now on), who still used traditional methodologies (Waterfall), by studying their work and achievements. In the second semester we prepared methods and tools for the students of the first semester of the academic year of 2023/2024 (“Year 2”, from now on) to execute their software development projects.

The rest of the paper is organized as follows: Section 2 describes how the Software Project Management course was conducted using traditional (Year 1) and agile (Year 2) methodologies and lists the expected learning outcomes of all students. Section 3 describes the assessment of the student's knowledge. In Section 4 we show and discuss the results. Section 5 presents and compares the results from related work to ours. Section 6 concludes the paper and describes future work.

2 THE SOFTWARE PROJECT MANAGEMENT COURSE

The SPM takes place at the end of a bachelor’s in computer science. As aforementioned, students must develop a software project. The goal is to provide the students with the capacity to plan, execute and control a simple software project as a team (groups of five), using appropriate engineering practices, and deliver it within the expected costs, deadlines, quality, and expectations. This project lasts the whole semester. The students must make a weekly effort of four hours per member of the group. The students record their individual and collective weekly efforts to monitor the progress of the project.

Throughout the semester, students have theoretical and practical classes. In the theoretical classes, concepts and knowledge relating to good engineering practices, software project management, and software development processes and methodologies are transmitted to the students. The practical classes have the objective for weekly follow-ups with each of the groups, in the form of progress meetings with the teacher. The four hours of weekly effort do not include weekly meetings with the teacher.

In this paper we study the transition from Waterfall, adopted in the Year 1, to Scrum, adopted in Year 2.

2.1 Learning outcomes

The Learning Outcomes (LO) of this course are listed in Table 1. Despite the change of the methodology adopted from one to the other academic year, the expected outcomes of the students are the same, but reached through different steps or objectives. Table 1 shows the general learning outcomes, and the main learning objectives for each of the years.

Table 1 - Learning outcomes and learning objectives

Learning outcomes	Learning objectives (Year 1 - Traditional methodology.)	Learning objectives (Year 2 - Agile methodology.)
LO1 – Describe software development as an engineering discipline, using appropriate terminology	- Explain key terms and concepts related to software engineering, such as requirements analysis, design patterns, and quality assurance (for both learning objectives). - Communicate effectively about software development topics using precise and accurate terminology (for both learning objectives).	
LO2 – Develop a software project in a team, following defined processes and good software engineering practices	- Interpret and follow the steps of software processes in a Waterfall methodology - Demonstrate, through involvement in a team, the fundamental elements of team building and team management	- List the scope of a software project - Define a branching strategy - Organize a team

LO3 – Plan a software development project using estimates, task scheduling, and resource allocation	- Create a Software Development Plan (SDP) Work Breakdown Structure, and Quality Plan - Allocate resources and estimate the software development effort - Extract tasks from functional requirements - Create an EVA plan	- Extract software requirements, using use cases and user stories - Estimate and prioritize user stories - Define the project releases - Plan the sprint and releases
LO4 – Apply best practices in quality management, risk management, team management, and stakeholder management	- Define a quality plan, i.e., quality assurance plan and assessment report - Review documents, using inspections, desk checks and walkthroughs - Define and follow a risk plan - Plan software testing at user, integration, and unit levels, and execute the tests	- Define a product and code level quality plan - Define a code review strategy - Define the test plan - Know how to analyze the results of test execution via the pipeline - Use a version management tool and a jars registry - Identify and define a risk plan
LO5 – Manage a software project by monitoring and controlling its progress, ensuring software delivery within the planned timeframes and costs, and managing stakeholder expectations	- Manage and control software execution progress using EVA - Run milestone analysis to manage progress and stakeholder expectations	- Plan the sprints and estimate tasks - Review the sprints, compare budget vs. costs - Negotiate with the client
LO6 – Demonstrate knowledge of software development processes and methodologies, as well as the principles and fundamentals of software process improvement	- Interpret the distinct stages and phases of a project following Waterfall - Understand all deliverables of a project - Improve the processes of a project according to various aspects such as team, client, scope, etc.	- Interpret the different processes and ceremonies of Scrum - Understand the need to use DevOps - Improve the processes of a project according to several aspects such as team, client, scope, etc.

2.2 Waterfall methodology (Year 1)

In Year 1, the students followed a Waterfall methodology, with six phases, grouped in three stages, as can be seen in Figure 1. During the first stage, the students prepared the project, by choosing the team, setting up the development environment, defining the scope and quality, planning the deliverables, defining, and writing a SDP, etc. During the second stage, three phases are performed: requirements specification, software development, and software acceptance. This is the phase where all the product requirements are described, where the final product is implemented and where all the tests are executed. Finally, there is the stage dedicated to completing the project, which only has one phase.

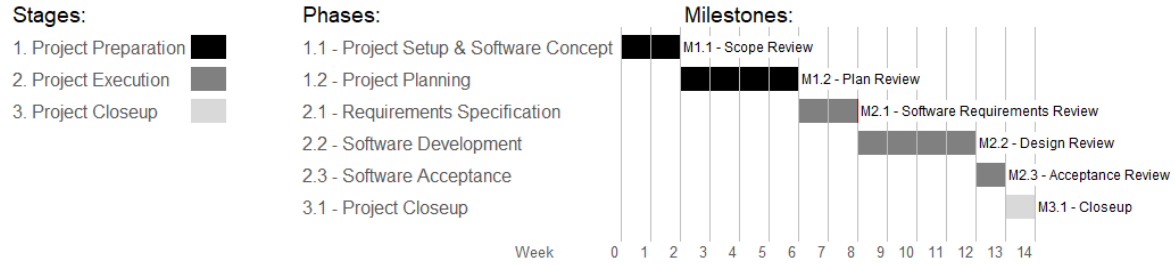


Figure 1 - Project life cycle using Waterfall, followed in Year 1

Since Waterfall is extremely focused on documentation, the students have sixteen deliverables throughout the project.

Regarding quality of the project, the students conduct quality control at both document and code levels. The students must write a document called a “Quality Assurance Plan”, which addresses various points, such as quality objectives, reviews, and tests to be carried out, risk plan, code standards, and evaluation metrics.

To monitor the progress of the project, the students used an Earned Value Analysis Chart [53].

2.3 Scrum methodology (Year 2)

For the experimental evaluation using Scrum we divided the project into three stages: preparation, implementation, and completion, as seen in Figure 2.

The first stage is dedicated to preparing the project. It is divided into two phases, the first of which is aimed at setting up the project, where the students learn the main principles and practices of agile and scrum, and prepared the tools, like GitLab repository (https://gitlab.com/). The second is a pre-game/sprint 0, where the students gather the requirements and write a product backlog with prioritized user stories, organize the team, plan the releases, among other tasks. The second stage, dedicated to implementing the product, is divided into three sprints. Although the sprints ate timeboxed they have different durations to fit the duration of the academic environment.

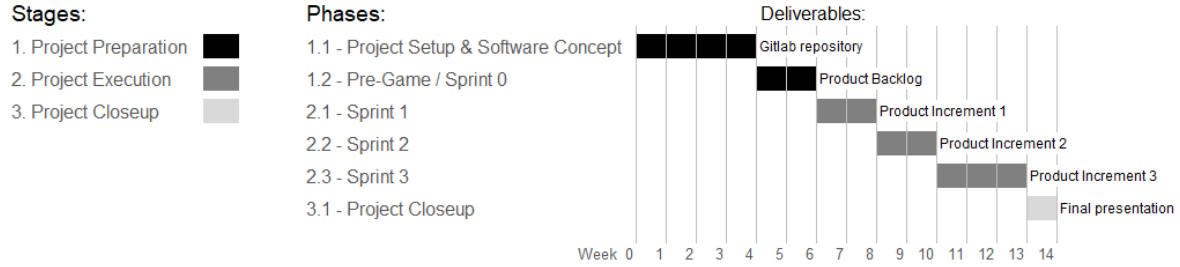


Figure 2 – Project life cycle using Scrum, followed in Year 2

Finally, there is the phase dedicated to completing the project, the phase 3.1. In this phase, the students give a demonstration to the whole class, presenting their project and how they planned, executed, and managed it throughout the semester.

The project consists of two releases. Release 1.0 aim is to create the Minimum Viable Product, which is delivered at the end of sprint 2. Release 2.0 (at the end of Sprint 3) is delivered with the final version of the application.

Regarding the Scrum ceremonies, the students practice Sprint Planning, Sprint Review and Sprint Retrospective, but don't practice Daily Scrum meeting (they don't have "days" of work but keep in sync during their hours out-of-class) and Backlog Refinement (the projects are too small in terms of features and time, hence, the students made small backlog refinements during the Sprint Reviews).

As far as the quality of the project is concerned, some changes were made from Year 1 to Year 2. First, now students have a defined branching strategy on git (unlike Waterfall, which had no branching rules). They must use at least three branches: "dev", "demo", and "main". Secondly, they must run unit tests automatically, using a pre-built continuous integration pipeline. Third, they are required to do code reviews when merge requests are made to the "demo" branch. They must also make a risk plan. The progress of the project is evaluated by sprint and release, comparing the planned product increment versus the delivered one.

3 CHARACTERIZATION OF STUDENTS

At the beginning of each Year 1 and Year 2 we conducted student characterizations so we could have comparable baselines for each version of the course. The students answered a short questionnaire at the beginning of the semester about their previous knowledge and experience related with software engineering and project management. Since it was an optional form, in Year 1 we obtained 36 answers out of 78 students, while in Year 2 we obtained 29 responses out of 70.

The questionnaire addressed the students' preferences and the importance they attach to some of the processes used, an analysis of work processes used in other projects, and the levels of knowledge in some areas, tools, and technologies of software engineering.

Regarding work processes, Figure 3 shows the students' preferences when asked whether they prefer to work in software projects alone, in pairs, or in groups (three to five). We can see that the percentage of students is nearly similar in the Year 1 and Year 2 for the three preferences, and in Year 2 there is a minimal difference in the preference for working in groups.

Figure 3 also shows the level of importance of some of the processes used. This question aimed to find out whether the students were conscious of the importance of these processes. In the questionnaire, the students had to give a rating from "Completely disagree" to "Completely agree", which was replaced by a rating from 1-5 to present the data, where 1 means “Completely disagree” and 5 means “Completely agree”.

We also presented a list of processes used in this work and asked which processes the students had used in other courses. We can see the answers to this question in Figure 4. The students had to give a rating of “Never”, “Rarely”, “Sometimes”, “Often”, and “Always”. To make it easier to show this data, this classification has been replaced by a score of 1-5, i.e., 1 to “Never” and 5 to “Always”. We can see that Year 1 students used some work processes more regularly than Year 2 students, especially in testing and code versioning tools.

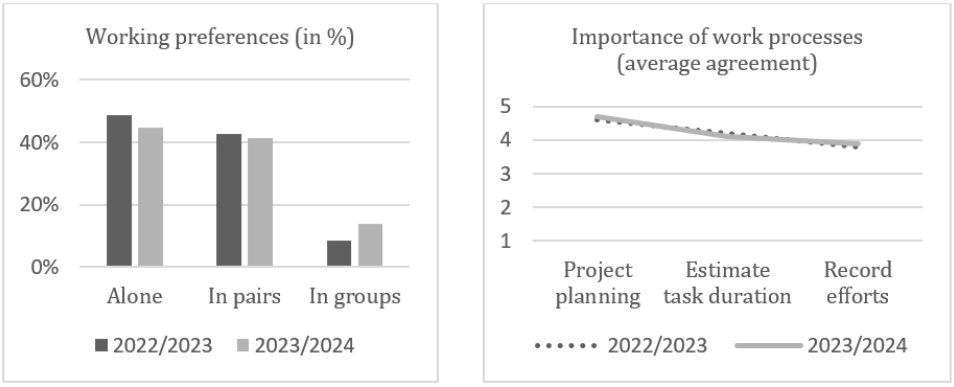


Figure 3 - Working preferences and importance of work processes

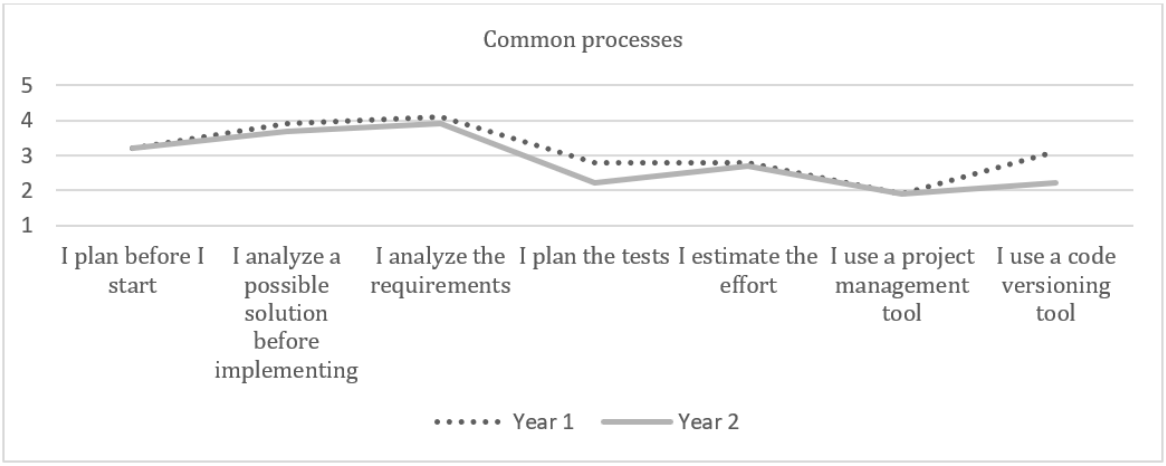


Figure 4 - Processes already used in other curricular units

Figure 5 shows the levels of knowledge of certain areas and topics in project management and software engineering. Although quite similar, we can observe that Year 1 students did arrive at this course a little better prepared than students from Year 2 in most topics.

Considering these results, and lacking additional data, it appears Year 1 students were marginally more prepared than their Year 2 counterparts before starting the Software Project Management course.

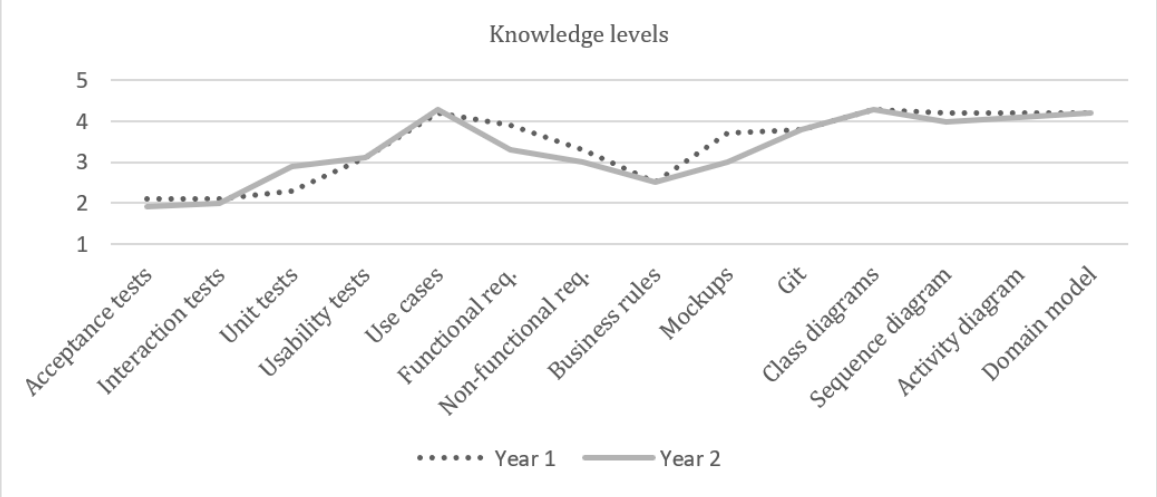


Figure 5 - Level of knowledge in some areas and topics of software engineering

4 RESULTS AND DISCUSSION

To assess which of the students, from Year 1 or from Year 2, had the best results at the end of the corresponding semesters, we compared their scores in two components: the final exam and the software project. Furthermore, for each of these assessment components, we observed the students' performance regarding their LO. Regarding the exam, we could do this by relating each of the questions of the exams to the particular LOs it was assessing. Regarding the software project, since the students has a progress meeting each week, their team was assessed regarding a list of topics that could be related with the LOs. Table 2 shows which academic year demonstrated better results for each of the learning outcomes. This assessment considered the grades of the students related to each of the LOs, both in the practical and theoretical parts.

Table 2 – Comparison of students' assessment at the end of each of the academic years

Learning outcomes	LO1	LO2	LO3	LO4	LO5	LO6
Better theoretical results	Year 2	Year 2	Year 2	Year 1	Year 2	Year 2
Better practical results	Year 2	Year 2	Year 2	Year 2	Year 2	Year 2

Students from Year 2, that followed the agile methodologies, demonstrated better results in almost all learning outcomes, even though they showed to be less prepared when started the project. Let us try to give some explanations to the observed results.

Regarding LO1 (please see Table 1) as Scrum is a more iterative process than Waterfall, it made the students better assimilate software engineering terms. About LO2, Year 2 students had more defined rules in some tools, such as the use of git. There was a mandatory branching strategy (at least three branches), and mandatory code reviews, among other things. Regarding LO3, Year 2 students went through the process of estimating and scheduling tasks several times, making them exercise this process more often. As for LO4, Year 1 showed better results in the exam because they had better assimilated, even if not applied, different quality strategies that Year 2 students. Regarding the practical results, these students applied better practices such as code review and automated tests, which are run continuously. Concerning LO5, as the students demonstrated to the stakeholders throughout the project, they were able to negotiate with him and clarify doubts/points that were sometimes not so clear from the start. About LO6, as with LO1, the fact that Scrum is iterative means that students practice the processes several times, hence assimilating better the principles and foundations of the methodologies. Figure 6 shows the final grades for each year's course. In Portugal, higher education institutions use a 20-point grading scale, and students need to have at least a grade of 10 to pass. Therefore, only those students who scored more than 10 points were counted.

Scrum is an iterative methodology meaning that students go through the processes several times, assimilating them better. For example, Waterfall estimates are done only once. Students do not have experience in estimating, so they are highly likely to fail when estimating tasks. On the other hand, in Scrum, they go through this process several times. Although at the end of Sprint 1, the students managed to deliver almost no user stories (as expected), in the last sprint they managed to deliver almost everything they set out to.

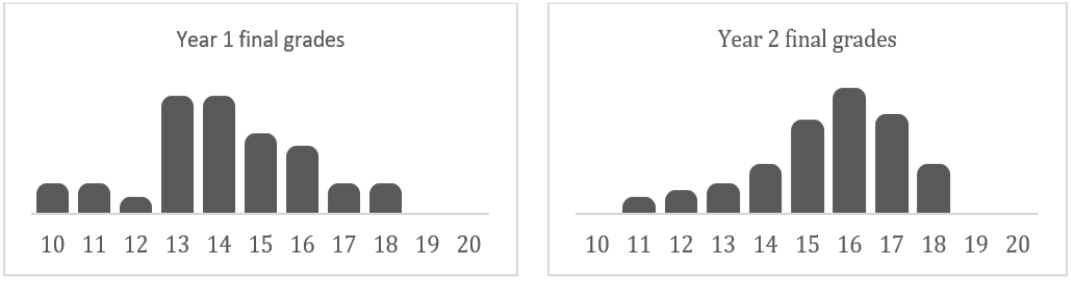


Figure 6 - Students final grades in each academic year of the study

5 RELATED WORK

Around the globe other high-education institutions have made similar changes from teaching traditional to agile software development. In this Section we walkthrough some papers with similar objectives to our paper.

In [44], software engineering students made a product, following traditional and agile methodologies. The aim of the paper is to report their experience in moving from the Waterfall process to the Agile process in realizing this software engineering project of a duration of two semesters. To evaluate whether there were any improvements in students' learning, 28 students answered a 9-question survey. The results "showed that the agile approach helped the students to have an overall better design and to avoid the mistakes they made in the initial design completed in the first phase of the finalization project". The conclusions, based on students' questionnaires, align with ours, emphasizing the advantages of agile. However, the fact that this study was carried out in a capstone project may influence the results, because the students already had more knowledge about software project management.

In [45], an experiment was carried out during one semester in two engineering schools. The paper aimed to evaluate the impact of the development method on the success of student (plan-driven, iterative, and no-process approach). The evaluation was based on metrics of product quality, team productivity, and teamwork quality. The results show that, regarding product quality, the no-process approach had more internal quality and the plan-driven approach had more external quality. In terms of team productivity, no approach stood out. In terms of team quality, the iterative approach won. The differences in this paper for ours lie in the fact that the study was conducted in the same year, but in different schools. It is possible that students were prepared differently for the experience due to variations in their curricula and instructors. Additionally, a notable distinction lies in the evaluation process, which focused solely on quality and productivity metrics. Similarly to our experience, we also noticed that the quality of the team has improved. However, against all expectations, the no-process approach managed to have better internal quality.

In [46], a study aimed to compare the testing phase in each of the methodologies. Four points were considered: "Divide and conquer", "Changes are always welcome", "Time and cost estimation" and "Customer Satisfaction". The results show that agile wins compared to traditional. We also came to this conclusion by analyzing why our LO4 had better results in Year 2. The continuous testing, and negotiate with the client throughout the project, increases the quality. The differences in this paper are that the study only considered the testing phase and was not developed only in an academic environment.

[47] explored the practical application of Agile in the context of educational institutions and project management. They concluded that agile methodologies bring many benefits, but that freedom in processes can be a disadvantage for them. For this reason, they state that "educators and students should strive for a balanced approach, considering both Agile and traditional methodologies". This was the main reason we had not yet made the transition to agile – i.e., without a rigorous process as Waterfall could make the project fail.

Finally, [48] conducted a study with just three teams. The teams mentioned positive points, such as having an operational product from an early stage and better communication with customers. However, they also mentioned disadvantages. Despite having better relations with clients, they point out that having to contact them frequently is a negative point in terms of slower programing. Regarding the advantages, we share the same opinion. However, in our study, we did not find frequent contact with customers to be a disadvantage.

6 CONCLUSION AND FUTURE WORK

Agile methodologies are a nowadays trend used by overall industry to achieve a better product quality and consequently an excellent quality-of-experience for the product consumer. The standard approach in this academic Computer Science course was to use a traditional methodology. Hence, we elaborated a two-year experiment to understand the benefits and limitations of a traditional vs agile methodology in an academic field. As a result, we have a good insight of the best approaches and what Learning Outcomes benefit from the Waterfall or Agile methodology.

The evidence presented in this paper shows that although the Year 1 students were better prepared due to pre-acquired knowledge, evaluated students following an agile methodology had better results in most of the Learning Outcomes.

As future work, we want to get feedback from the companies we initially interviewed. One of the less positive aspects was that the students did not have much knowledge of agile methodologies. Therefore, it would be important to get feedback from these companies again, to check that the Year 2 students have gained more skills. It would also be interesting to start applying DevOps practices. In Year 2, the students already used a pre-built continuous integration pipeline, but we cloud start applying more practices and analyze their inclusion. In addition, it would also be important to start including artificial intelligence, since it is constantly growing trend.

REFERENCES

[1] A. Mishra and Y. I. Alzoubi, "Structured software development versus agile software development: a comparative analysis," *International Journal of System Assurance Engineering and Management*, vol. 14, no. 4, pp. 1504–1522, Aug. 2023, doi: 10.1007/S13198-023-01958-5.

[2] A. Aitken and V. Ilango, "A comparative analysis of traditional software engineering and agile software development," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 4751–4760, 2013, doi: 10.1109/HICSS.2013.31.

[3] "State of Agile Report," 2022. Accessed: Apr. 28, 2024. [Online]. Available: https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf

[4] K. Schwaber and J. Sutherland, "The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game," 2020. Accessed: Apr. 28, 2024. [Online]. Available: https://www.scrum.org/resources/scrum-guide

[5] "Writing Course Goals/Learning Outcomes and Learning Objectives – Center for Excellence in Learning and Teaching." Accessed: Jul. 02, 2024. [Online]. Available: https://www.celt.iastate.edu/instructional-strategies/preparing-to-teach/tips-on-writing-course-goalslearning-outcomes-and-measureable-learning-objectives/

[6] IEEE Computer Society and Association for Computing Machinery, "Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," 2015, p. 94.

[7] Association for Computing Machinery and IEEE Computer Society, *Computing Curricula 2020 (CC2020): Paradigms for Global Computing Education*. 2020. doi: 10.1145/3467967.

[8] Project Management Institute, "Project Management Course Specifications," in *Guidelines for Undergraduate Curriculum and Resources*, vol. 1, pp. 1–92.

[9] T. Thesing, C. Feldmann, and M. Burchardt, "Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project," *Procedia Comput Sci*, vol. 181, pp. 746–756, Jan. 2021, doi: 10.1016/J.PROCS.2021.01.227.

[10] W. Rosa, B. K. Clark, R. Madachy, and B. W. Boehm, "Empirical Effort and Schedule Estimation Models for Agile Processes in the US DoD," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 3117–3130, Aug. 2022, doi: 10.1109/TSE.2021.3080666.

[11] Y. Beng Leau, W. Khong Loo, W. Yip Tham, and S. Fun Tan, "Software Development Life Cycle AGILE vs Traditional Approaches".

[12] Ian Sommerville, *Software Engineering*, 10th ed. Boston: Pearson Education Limited, 2016.

[13] V. S. Alagar and K. Periyasamy, "Specification of Software Systems," 2011, doi: 10.1007/978-0-85729-277-3.

[14] M. Stoica, M. Mircea, and B. Ghilic-Micu, "Software Development: Agile vs. Traditional," *Informatica Economică*, vol. 17, no. 4, 2013, doi: 10.12948/issn14531305/17.4.2013.06.

[15] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, "Challenges in software evolution," *International Workshop on Principles of Software Evolution (IWPSE)*, vol. 2005, pp. 13–22, 2005, doi: 10.1109/IWPSE.2005.7.

[16] Project Management Institute, *A guide to the project management body of knowledge*, 6th ed. Accessed: May 23, 2024. [Online]. Available: www.PMI.org

[17] E. Masciadra, "Traditional Project Management," *Knowledge Management and Organizational Learning*, vol. 5, pp. 3–23, 2017, doi: 10.1007/978-3-319-51067-5_1/COVER.

[18] F. Bilimleri Dergisi, M. Javanmard, and M. Alian, "Comparison between Agile and Traditional software development methodologies," *Cumhuriyet University Faculty of Science Science Journal (CSJ)*, vol. 36, no. 3, p. 36, 2015, Accessed: Feb. 11, 2023. [Online]. Available: http://dergi.cumhuriyet.edu.tr/cumuscij©2015

[19] S. Shaikh and S. Abro, "Comparison of Traditional & Agile Software Development Methodology: A Short Survey," *International Journal of*

Software Engineering and Computer Systems, vol. 5, no. 2, pp. 1–14, 2019, doi: 10.15282/ijsecs.5.2.2019.1.0057.

[20] R. Mall, *Fundamentals of Software Egineering*, Fourth. PHI Learning Private Limited, 2014.

[21] C. Kaur and V. Kumar, "Comparative Analysis of Iterative Waterfall Model and Scrum," *FP-International Journal of Computer Science Research*, vol. 2, no. 1, pp. 11–14, 2015.

[22] R. Pressman, *Software Engineering: A Practitioner's Approach*, Seventh Edition. 2010. Accessed: Feb. 11, 2023. [Online]. Available: https://www.mlsu.ac.in/econtents/16_EBOOK-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf

[23] N. Mohammed, A. Munassar, and A. Govardhan, "A Comparison Between Five Models Of Software Engineering," *IJCSI International Journal of Computer Science Issues*, vol. 7, no. 5, 2010, Accessed: Feb. 11, 2023. [Online]. Available: www.IJCSI.org

[24] A. Mujumdar, G. Masiwal, and P. M. Chawan, "Analysis of various Software Process Models," *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 3, pp. 2015–2021, 2012, Accessed: Feb. 11, 2023. [Online]. Available: www.ijera.com

[25] A. Alshamrani, A. Bahattab, and I. Fulton, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model," *IJCSI International Journal of Computer Science Issues*, vol. 12, no. 1, 2015.

[26] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," *International Conference on Advanced Computing and Communication Technologies, ACCT*, pp. 189–196, 2014, doi: 10.1109/ACCT.2014.73.

[27] A. Anwar, "A Review of RUP (Rational Unified Process)," *Ashraf Anwar International Journal of Software Engineering (IJSE)*, no. 5, p. 8, 2014.

[28] J. Manalil, "RATIONAL UNIFIED PROCESS Seminar Report," 2010.

[29] "What Is the V-Model? (Definition, Examples) | Built In." Accessed: Dec. 27, 2024. [Online]. Available: https://builtin.com/software-engineering-perspectives/v-model

[30] K. Beck *et al.*, "Manifesto for Agile Software Development." Accessed: May 23, 2024. [Online]. Available: https://agilemanifesto.org/

[31] S. Kumar Dora and P. Dubey, "Software development life cycle (SDLC) analytical comparison and survey on traditional and agile methodology," 2013, Accessed: Feb. 11, 2023. [Online]. Available: www.abhinavjournal.com

[32] F. Tsui, O. Karam, and B. Bernal, *Essentials of software engineering*, Third edition. 2014. Accessed: Feb. 13, 2023. [Online]. Available: https://www.worldcat.org/pt/title/essentials-of-software-engineering-third-edition/oclc/864219285

[33] M. Alqudah and R. Razali, "A comparison of scrum and Kanban for identifying their selection factors," *Proceedings of the 2017 6th International Conference on Electrical Engineering and Informatics: Sustainable Society Through Digital Innovation, ICEEI 2017*, vol. 2017-November, pp. 1–6, Mar. 2018, doi: 10.1109/ICEEI.2017.8312434.

[34] K. Bhavsar, Dr. V. Shah, and Dr. S. Gopalan, "Scrumban: An Agile Integration of Scrum and Kanban in Software Engineering," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 4, pp. 1626–1634, Feb. 2020, doi: 10.35940/ijitee.D1566.029420.

[35] D. Anderson and A. Carmichael, *Essential Kanban Condensed*. 2016. Accessed: May 23, 2024. [Online]. Available: https://www.thescrummaster.co.uk/wp-content/uploads/2019/09/Essential-Kanban-Condensed-v1.0.0.pdf

[36] A. Janes, "A guide to lean software development in action," *2015 IEEE 8th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2015 - Proceedings*, May 2015, doi: 10.1109/ICSTW.2015.7107412.

[37] D. Wells, "Extreme Programming: A Gentle Introduction." Accessed: May 23, 2024. [Online]. Available: http://www.extremeprogramming.org/

[38] R. Fojtik, "Extreme programming in development of specific software," *Procedia Comput Sci*, vol. 3, pp. 1464–1468, 2011, doi: 10.1016/J.PROCS.2011.01.032.

[39] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A Survey of DevOps Concepts and Challenges," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, Nov. 2019, doi: 10.1145/3359981.

[40] R. Jabbari, N. Bin Ali, K. Petersen, and B. Tanveer, "What is DevOps? A systematic mapping study on definitions and practices," *ACM International Conference Proceeding Series*, vol. 24-May-2016, May 2016, doi: 10.1145/2962695.2962707.

[41] DevOps Research & Assessment, "2022 Accelerate State of DevOps Report." Accessed: May 23, 2024. [Online]. Available: https://services.google.com/fh/files/misc/2022_state_of_devops_report.pdf

[42] DevOps Research & Assessment and puppet, "State of DevOps Report 2016." Accessed: Jul. 10, 2024. [Online]. Available: https://dora.dev/publications/pdf/state-of-devops-2016.pdf

[43] DevOps Research & Assessment, "State of DevOps 2019," 2019. Accessed: Jul. 10, 2024. [Online]. Available: https://services.google.com/fh/files/misc/state-of-devops-2019.pdf

[44] M. Alshayeb, S. Mahmood, and K. Aljasser, "Moving from Waterfall to Agile Process in Software Engineering Capstone Projects," pp. 107–114, May 2018, doi: 10.5121/CSIT.2018.80808.

[45] R. Wlodarski, A. Poniszewska-Maranda, and J. R. Falleri, "Comparative Case Study of Plan-Driven and Agile Approaches in Student Computing Projects," *2020 28th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2020*, Sep. 2020, doi: 10.23919/SOFTCOM50211.2020.9238196.

[46] V. Kate, S. Bhalerao, and V. Sharma, "Exploring Agile Methodologies in Educational Software Development- A Comparative Analysis and Project Management Insights," *3rd IEEE International Conference on ICT in Business Industry and Government, ICTBIG 2023*, 2023, doi: 10.1109/ICTBIG59752.2023.10456214.

[47] A. Sinha and P. Das, "Agile Methodology Vs. Traditional Waterfall SDLC: A case study on Quality Assurance process in Software Industry," *2021 5th International Conference on Electronics, Materials Engineering and Nano-Technology, IEMENTech 2021*, 2021, doi: 10.1109/IEMENTECH53263.2021.9614779.

[48] D. F. Rico and H. H. Sayani, "Use of agile methods in software engineering education," *Proceedings - 2009 Agile Conference, AGILE 2009*, pp. 174–179, 2009, doi: 10.1109/AGILE.2009.13.

[49] "Earned Value Management Systems Tool (EVMS) | PMI." Accessed: Apr. 29, 2024. [Online]. Available: https://www.pmi.org/learning/library/earned-value-management-systems-analysis-8026

[50] "GitLab." Accessed: Apr. 29, 2024. [Online]. Available: https://gitlab.com/

[51] "Free personality test, type descriptions, relationship and career advice | 16Personalities." Accessed: Oct. 20, 2024. [Online]. Available: https://www.16personalities.com/

[52] V. Yakovyna, M. Seniv, and I. Symets, "The Relation between Software Development Methodologies and Factors Affecting Software Reliability," *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, vol. 1, pp. 377–381, Sep. 2020, doi: 10.1109/CSIT49958.2020.9321937.

[53] "Earned Value Management Systems Tool (EVMS) | PMI." Accessed: Aug. 07, 2024. [Online]. Available: https://www.pmi.org/learning/library/earned-value-management-systems-analysis-8026

[54] "AESE insight #32 - AESE Business School - Formação de Executivos." Accessed: Oct. 05, 2024. [Online]. Available: https://www.aese.pt/aese-insight-32/

[55] "Periodic Table of DevSecOps Tools | Digital.ai," *https://digital.ai/*, Accessed: Jul. 10, 2024. [Online]. Available: https://digital.ai/learn/devsecops-periodic-table/

Instituto Superior
de Engenharia
Politécnico de Coimbra