Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu



# Estudo comparativo das Metodologias Ágeis e PMBOK

Março de 2018

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu



# Estudo comparativo das Metodologias Ágeis e PMBOK

**Tese de Mestrado**
Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Jorge Alexandre Albuquerque Loureiro
Professor Mestre João Pedro Menoita Henriques

Março de 2018

Aos meus pais Graça e João, por todo o apoio de sempre.

À minha esposa Marta, por toda a paciência e carinho.

# RESUMO

No mercado atual, as empresas tendem a mudar, repentinamente, os seus requisitos, de forma a se adaptarem às mudanças das áreas de negócio. Devido a essa realidade, as empresas de desenvolvimento de *software* procuram novas metodologias de desenvolvimento para poderem acompanhar a velocidade de mudança dos seus clientes e conseguirem ser competitivas, já que uma das razões mais fortes para a adoção de metodologias ágeis é a velocidade de entrega do produto.

Embora as metodologias ágeis conheçam uma aceitação crecente, a nível mundial especialmente Scrum e híbridas, algumas delas derivadas de Scrum, tal como Scrum/XP (Extreme Programming) e Scrumban, revelam alguns pontos fracos, tal como uma documentação débil e requisitos pouco definidos. Neste contexto, o estudo de um *standard* na gestão de projetos pode contribuir para colmatar essas fraquezas, sendo o PMBOK um dos mais reconhecidos internacionalmente.

Com base na análise das metodologias ágeis e no PMBOK, foi criada a metodologia híbrida XPOKs, que resultou da combinação entre modelos ágeis com as boas práticas recomendadas pelo PMBOK, passível de ser aplicada a pequenas e médias empresas de desenvolvimento de *software,* que pretendem alcançar rigor na execução dos seus projetos, o cliente presente e entregas constantes.

O XPOKs compreende três fases: início, ciclos de desenvolvimento e fecho. O início do projeto, é fundamental para definir o âmbito, onde o cliente e alguns elementos da equipa de desenvolvimento definem as histórias. Os ciclos de desenvolvimento são compostos por vários *sprints*, onde as histórias são codificadas e no término de cada *sprint* é entregue uma nova versão ao cliente. O fecho do projeto serve para estabilizar a aplicação e por fim dar formação ao cliente.

A validação do XPOKs ocorre com um teste piloto aplicando a metodologia a um projeto e com um questionário que foi respondido pela equipa, constatando-se que, caso tivessem de eleger uma metodologia para os seus projetos provavelmente, optariam pelo XPOKs. Dados estes resultados preliminares, pode afirmar-se que esta metodologia pode apoiar a gestão de projetos de *software*.

# ABSTRACT

In the current worldwide market, the companies tend to suddenly change their requirements, in order to adapt to the constant changes in the business world. ue to this reality, the software developing companies are searching for new development methodologies so they can compete on an ever-changing client environment and become highly competitive, since one of the strongest reasons for the adoption of agile methodologies is due to the higher product development delivery speed.

Although the increasing worldwide acceptance of agile methodologies, specially the Scrum and hybrid ones, where some of them are actually Scrum derived, like Scrum/XP (Extreme Programming) and Scrumban, they show some weakenesses, like poor documentation and unclear requirements. In this context, a study of the application of the combination of agiles and standard methodologies in project management, could help filling the gap related to those weaknesses, having the PMBOK as the most well recognized worldwide for this study.

Based on the analysis of the agile methodologies and the PMBOK, a hybrid methodology has been created with the name of XPOKs, resulted from the combination between the agile models and the recommended PMBOK best practices, capable of being implemented on small to medium sized development companies, that intend to have high accuracy in the execution of projects, the client to be frequently involved and to have constant deliveries.

The XPOKs is comprehended by three stages: initiation, development cycles and closing. The initiation of the project is fundamental to define his scope, where the client and some elements of the development team specify the user stories. The development cycles are composed from different sprints, where the stories are codified and in the sprint closure, a new version of the project is delivered to the client. The main purpose of the project closing stage, is to normalize the application, improving the stability, and, finally, providing training to the customer.

The XPOKS validation was performed through a pilot test, applying thus this methodology to a project. As a result of a survey, that was answered by the project development team, it was possible to conclude that if they have to elect a methodology for their projects, they will eventually vote for XPOKs. Given these preliminar results, it is possible to claim that this methodology can sustain the management of software projects.

# PALAVRAS CHAVE

Gestão de projetos

Metodologias Ágeis

Scrum

Extreme Programming

Kanban

PMBOK

XPOKs

# KEY WORDS

Project Management

Agile Methodologies

Scrum

Extreme Programming

Kanban

PMBOK

# AGRADECIMENTOS

Deixo um agradecimento muito especial à minha esposa Marta, porque no momento em que abracei este projeto me apoiou. Quando existiu a necessidade de estar mais ausente da nossa vida, sozinha ter suportado este pilar tão importante.

Não posso esquecer os meus pais Graça e João. Desde cedo, sempre incentivaram a minha vida académica. Sem eles, não seria possível este percurso.

Agradeço ainda a todos os companheiros que estiveram comigo neste projeto, em especial à Vânia Trindade, ao Fernando Santos, ao Vasco Santos e ao António.

Por fim, um muito obrigado aos meus orientadores Jorge Loureiro e João Henriques por todo o apoio prestado e conselhos que se revelaram indispensáveis para o sucesso deste trabalho.

# ÍNDICE GERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABELAS

# ABREVIATURAS E SIGLAS

| | |
|---|---|
| ASD | *Adaptive Software Development* |
| DM | *Delivery Manager* |
| DoD | *Definition of Done* |
| DSDM | *Dynamic Systems Development Method* |
| EAP | Estrutura analítica do projeto |
| EM | *Engagement Manager* |
| FDD | *Feature Drive Development* |
| IEC | *International Electrotechnical Commission* |
| ISO | *International Organization for Standardization* |
| PMBOK | *Project Management Body of Knowledge* |
| PMI | *Project Management Institute* |
| PO | *Product Owner* |
| QA | *Quality Assurance* |
| SM | *Scrum Master* |
| TL | *Team Leader* |
| XP | *Extreme Programming* |
| DevOps | *Development and Operations* |

# 1. Introdução

A atividade de desenvolvimento de *software* é muitas vezes feita de forma caótica, caraterizada pela frase "desenvolvimento e correção". Esta realidade funciona para pequenos projetos, mas à medida que um sistema cresce, torna-se cada vez mais difícil adicionar novas funcionalidades, conduzindo à necessidade de se desenvolverem métodos de trabalho (Fowler, 2001).

Desde muito cedo que se têm tentado aplicar técnicas ao desenvolvimento de *software*. Já nos anos 30 um especialista em qualidade da empresa Bell Labs propôs uma versão simplificada do plan-do-check-act[1] para melhorar a qualidade, sendo mais tarde explorada e promovida por W. Edwards Deming. Já nos anos 60, num projeto de *software* denominado "Mercury", a equipa trabalhou com iterações curtas e todas as mudanças requeriam revisão técnica, aplicando a escrita de testes antes da implementação o que corresponde a uma prática relacionada com as metodologias ágeis (Larman e Basili, 2003).

Nos anos 70 e 80, as empresas que desenvolviam *software* atravessaram um momento de crise, sendo necessário encontrar respostas ao porquê dos projetos não terminarem dentro dos orçamentos, os produtos terem baixa qualidade e não satisfazerem o cliente (Hiwarkar, Doshi e Chinta, 2016). Para Robiolo e Grane, embora nesta fase as empresas começassem a aplicar métodos de desenvolvimento, cedo se aperceberam que os desvios sobre o planeado aconteciam quando existia indefinição nas especificações, o que levava a um *software* que não cumpria as

[1] PDCA – *Plan-do-check-act* é um processo de gestão iterativo em quatro etapas utilizado para controlo e melhoria contínua de processos.

necessidades (Robiolo e Grane, 2014). É igualmente na década de 70 que Bell e Thayer usam pela primeira vez o termo Waterfall, citando o artigo de Royce, onde foi apresentado formalmente o modelo. O *software* era desenvolvido passando por várias etapas que iam desde os requisitos à implementação, podendo apenas iniciar-se uma etapa nova, quando a anterior estivesse concluída (Royce, 1970).

Nos anos 90, começaram a surgir novas metodologias de trabalho, as metodologias ágeis, e com a sua evolução acabaram por se dividir em dois grandes grupos: métodos tradicionais e métodos ágeis. Os métodos tradicionais necessitam que os requisitos estejam totalmente especificados e são construídos com base num planeamento extremamente meticuloso. As organizações onde esta realidade acontece, tipicamente são de grande dimensão e com um alto grau de formalização. Por outro lado, os métodos ágeis baseiam-se na premissa de um *software* de alta qualidade, adaptando-se aos requisitos do cliente com um *feedback* rápido, permitindo fazer precocemente melhorias (Sharma e Kotwal, 2016).

Em 2001, foi criado o Manifesto Ágil, onde se encontram mencionados doze princípios que uma Metodologia Ágil deve cumprir (Agile, 2001). Neste contexto, vários estudos indicam que os Modelos Ágeis têm sido adotados fortemente no mundo do desenvolvimento de *software* (peter, 2013). Se analisarmos os dados do Chaos Report, facilmente verificamos que projetos que adotaram metodologias ágeis, tiveram taxas de sucesso de 42% e somente 9% de insucesso. Por sua vez, os que seguiram Waterfall apenas 14% tiverem sucesso (Hastie e Wojewoda, 2015). O aspeto fulcral das metodologias ágeis é conseguirem adaptar-se às alterações, algo que é claramente exposto por Nikhil Jose Kuttenchery, escrevendo **"***believe in the phrase "Change is the only constant". Agile is here to change Software Development forever."*.

O facto de existirem diversas metodologias tem levado a estudos e ao surgimento de metodologias híbridas (McLaughlin, 2016). Lendo o relatório de 2015 da empresa Versionone, constatamos que 10% dos inquiridos que utilizam métodos ágeis usam Scrum mais XP e 8% utiliza diversas metodologias (Versionone, 2016).

Embora as metodologias ágeis, tenham auxiliado no sucesso dos projetos de *software*, também apresentam alguns pontos fracos. Com base no estudo de Michel dos Santos (Soares, 2004) podemos indicar os seguintes:

- Análise de riscos;
- Controlo dos custos e duração podem ser difíceis de gerir;
- Gestão de equipas grandes.

Sommerville afirma que, em alguns casos, as metodologias ágeis podem não ser a melhor escolha, quando se trata de sistemas muito complexos ou críticos, uma vez que estes necessitam de documentação extensa e formal (Sommerville, 2007).

O Project Management Body of Knowledge (PMBOK), que é uma referência na área de gestão de projetos, contém as melhores práticas que podem contribuir para ultrapassar alguns problemas identificados com a prática de metodologias ágeis, como o controlo de custo através dos seus processos. Além disso continua a ser amplamente usado em todo o mundo, em cerca de 75% dos projetos (Singh e Lano, 2014).

## 1.1 Motivação e Objetivos do Trabalho

A experiência profissional do autor permitiu o contacto com diversas metodologias ágeis, tais como Outsystems, Kanban e Scrum. Atualmente, a sua atividade laboral é exercida numa empresa de desenvolvimento de *software* e consultoria, que conta com cerca de 60 colaboradores, dos quais 60% são programadores.

Embora a organização discuta a possibilidade de utilização de diversas metodologias ágeis, apenas duas são utilizadas consoante o tipo de projeto. Quando um projeto nasce de um documento com a especificação, onde é possível efetuar um planeamento do volume do trabalho, opta-se por utilizar Scrum, para assim se dividir o trabalho em *sprints*, existirem entregas constantes ao cliente e alocar a equipa ao projeto, havendo uma monitorização do progresso do trabalho. Quando um projeto se encontra geralmente em produção e a realização de trabalho está dependente de pedidos ocasionais do cliente, opta-se por Kanban, pois maioritariamente os pedidos são de duração curta, não permitindo criar um *sprint* de duas semanas como se usa em Scrum. Outra justificação prende-se com a instabilidade na definição das prioridades quando existe mais do que um pedido, pois o cliente pode requerer que algum passe à frente e o projeto ter de ser entregue no imediato, opondo-se à metodologia Scrum.

Embora se usem estas metodologias, têm-se sentido algumas dificuldades como:

- Documentação – A generalidade dos projetos contém um défice de documentação ou está extremamente desatualizada. Enquanto os responsáveis do projeto estão envolvidos com a prática de comentar o código fonte, a ausência de documentação não é muito sentida, embora quando é feito um pedido de alteração a uma funcionalidade antiga, nem sempre é possível, de forma rápida, compreender o seu impacto no negócio. Contudo quando os projetos entram em fase de manutenção e os trabalhadores que a efetuam não estiveram envolvidos anteriormente, obriga-os a lerem o código fonte para perceberem os impactos, o que tem originado entregas com problemas.

- Definição do âmbito do projeto - Pelo facto de usarmos metodologias ágeis, iniciam-se muitos projetos com base num caderno de requisitos inacabados e quando estamos na reta final, torna-se difícil o fecho, uma vez que o cliente consegue argumentar que a extensão a uma funcionalidade estava prevista.

- Práticas de trabalho das equipas de desenvolvimento - Os métodos usados preocupam-se mais com a gestão do projeto do que como fazer a implementação. Assim, tem sido necessário recorrer a alguns métodos externos às metodologias usadas, como o refactoring, que permite a reorganização do código.

O PMBOK é um documento que reúne o corpo de conhecimento na atividade de Gestão de Projetos produzido pelo PMI (Institute, 2013) que pode auxiliar na resolução de alguns problemas, acima referidos, sobretudo na definição do âmbito do projeto. Um dos grupos do PMBOK trata do início do projeto, onde são determinado os objetivos, identificados todos os participantes, documentadas todas as restrições, e é executado um documento importante designado por termo de abertura.

A conjugação do mundo empresarial com o mundo académico originou o desafio de estudar as metodologias ágeis mais aceites no mercado, as suas práticas e conjugar com alguns processos do PMBOK, permitindo assim mitigar as dificuldades identificadas anteriormente.

As metodologias ágeis não substituem as metodologias tradicionais. Pode-se afirmar que os processos híbridos são uma realidade e que, como em muitas outras áreas, assiste-se a uma tendência de hibridação. Contudo, no processo de criação de metodologias, devemos ter em atenção que as equipas de desenvolvimento encontram-se cada vez mais focadas e preocupadas com o desenvolvimento de *software* de alta qualidade (Hayata e Han, 2011).

Portanto, para alcançar os objetivos propostos, a dissertação inclui as seguintes tarefas:

- Estudar e aprofundar conhecimentos das metodologias ágeis existentes e integração com o PMBOK;
- Criar/melhorar uma abordagem híbrida, possibilitando às equipas terem práticas de trabalho e um ciclo de vida do projeto bem definido. Já outros autores combinaram Scrum com XP ou Kanban ou PMBOK (Mushtaq e Qureshi, 2012), (Landry e McDaniel, 2016);
- Efetuar um teste piloto com a metodologia híbrida criada, colocando uma equipa a aplicar a metodologia num desenvolvimento de *software*, no qual será recolhida a sua opinião.

Exposta a motivação e as principais tarefas a desenvolver, pode-se indicar que o trabalho possui dois grandes objetivos. O primeiro inclui a comparação entre as diversas metodologias e o PMBOK, e o segundo consiste em criar uma metodologia híbrida que ajude a colmatar as dificuldades antes expostas.

## 1.2 Metodologia de Investigação

Uma metodologia é um conjunto de métodos e técnicas usadas para se atingir um fim. Assim serão utilizadas metodologias ágeis, já que são uma prática usada no desenvolvimento de *software*, mas que pode ser aplicada também na elaboração de uma dissertação, ajudando portanto a manter um ritmo sustentável e a conseguir entregar o resultado final atempadamente (Alipui *et al.*, 2014).

A citação de Alipui resume, enquadra e fundamenta a utilização de partes da metodologia Scrum, que será apresentada em pormenor na secção 4.1., pois que implicitamente inclui no processo vários constituintes da metodologia, nomeadamente a própria essência da evolução e da rapidez de adaptação à mudança, o uso de *sprints*, reuniões retrospetivas e o envolvimento de um conjunto de *stakeholders* (Scrum Master, cliente) que garantem o feedback imediato, permitindo rapidez na adoção de medidas corretivas e o planeamento das próximas atividades.

> "Dissertations are designed briefly and evolve over the lifetime of the document. Regular check-ins (frequent iterations) ensures the product is of high quality without waste."*(Alipui, 2014)*

Como metodologia de investigação será utilizada a técnica dos seis P, proposta por Oates (Oates, 2005).

Propósito: Estudo comparativo das metodologias ágeis em busca de um modelo híbrido.

Produtos: Um estudo detalhado de cada metodologia investigada, para perceber onde se enquadra, as suas vantagens e desvantagens e apresentação de um modelo híbrido resultante do estudo obtido, apresentado numa dissertação.

Processo: Durante a investigação, será estudada a documentação das diversas metodologias, artigos científicos, nomeadamente aqueles que se focam na comparação de metodologias ágeis com o PMBOK. Serão feitas pequenas iterações de investigação com o intuito de apresentar resultados regularmente e assim ser adicionado valor ao produto final.

Participantes: No decorrer da investigação a participação de diversos agentes, colegas de trabalho e investigação, orientador e co-orientador e professores.

Paradigma: O autor, no âmbito desta investigação, identifica-se com o paradigma interpretativo, uma vez que se pretende analisar e comparar diversas metodologias.

Apresentação: Os resultados da investigação serão apresentados na forma de uma dissertação.

## 1.3 Estrutura da Dissertação

O documento encontra-se dividido em sete capítulos, nos quais se pretende expor o estudo dos diversos métodos de desenvolvimento ágil, assim como o desenvolvimento de uma metodologia híbrida aplicada num teste piloto.

O segundo capítulo contemplará o estudo breve das metodologias de desenvolvimento de *software*, demonstrando o seu estado atual de utilização.

O terceiro capítulo irá analisar mais profundamente o PMBOK, e a sua utilização em abordagens híbridas, de forma a serem verificados que processos podem vir a ser utilizados na metodologia a criar.

O quarto capítulo apresentará, de forma detalhada, o estudo efetuado de cada metodologia ágil, nomeadamente Scrum e XP.

No quinto capítulo serão realizadas diversas comparações entre as metodologias ágeis e com o PMBOK, destacando uma comparação detalhada entre XP com Scrum e Scrum com PMBOK.

No sexto capítulo, a metodologia híbrida será apresentada, fundamentada e aplicada a um projeto. Com o *feedback* obtido por parte dos participantes, será feita uma primeira validação da nova metodologia.

O sétimo e último capítulo irá expor uma reflexão do trabalho desenvolvido, com especial foco na metodologia híbrida proposta. Serão também sugeridos caminhos futuros para a continuação deste trabalho.

# 2. Metodologias de Desenvolvimento de *Software*

Neste capítulo pretende-se obter uma análise geral das diferentes metodologias de desenvolvimento de *software* e avaliar as metodologias que têm sido mais aceites. Este encontra-se dividido em seis secções. A primeira secção mostra o que é uma metodologia de desenvolvimento. A segunda o que é um ciclo de desenvolvimento. A terceira secção aborda as metodologias tradicionais pormenorizando o modelo em cascata por ser o mais antigo e utilizado atualmente. A quarta secção compara dois *standards* de gestão de projetos - o PMBOK e Prince. A quinta demonstra quais as metodologias ágeis mais utilizadas na atualidade, as suas vantagens e desvantagens e por fim é apresentado o sumário do capítulo.

## 2.1 Metodologia de Desenvolvimento de *Software*

Uma metodologia é uma abordagem organizada para se atingir um objetivo, através de um conjunto de métodos que possuem um conjunto de técnicas, relacionadas com aspetos técnicos como a codificação ou aspetos não técnicos como a gestão (Avison e Fitzgerald, 2003), (Marks, 2012), (Aitken e Ilango, 2013), (Snyder, 2014). O grande objetivo das metodologias é aumentar a probabilidade de entrega bem sucedida dos resultados do projeto, através de um melhor controlo do seu âmbito, tempos de entrega e redução de riscos, contribuindo para a satisfação do cliente e para o valor gerado. Nesta sequência, é necessário utilizar uma metodologia

adequada à realidade das organizações (Paulson, 2001), (Kerzner, 2002), (Charvat, 2003), (Chin e Spowage, 2010).

As empresas que desenvolvem *software* têm adotado metodologias para melhorar a gestão de projetos de modo a alcançarem um uso mais eficiente dos seus recursos em ambientes de múltiplos projetos, melhorarem a qualidade da entrega e pela adoção de uma abordagem comum nos seus projetos (Dai e Wells, 2004).

Para Sommerville, existem atividades genéricas comuns a todas as metodologias de desenvolvimento (Sommerville, 2007), sendo elas:

- Especificação – Definição do produto, dos requisitos e das restrições;
- Desenvolvimento – Implementação do *software*, segundo as especificações do cliente;
- Validação – Todo o *software* é testado para garantir a sua qualidade e assegurar que se encontra de acordo com as especificações;
- Evolução – Manutenção evolutiva do *software* para colmatar necessidades novas sentidas pelo cliente.

## 2.2 Ciclo de Vida de Desenvolvimento de *Software*

O ciclo de vida de desenvolvimento de *software* é um processo que ajuda a definir as atividades e o modo como se interligam em cada etapa de desenvolvimento (Alshamrani e Bahattab, 2015). Existe também um padrão internacional, a ISO/IEC 12207 (ISO/IEC e Society, 2008) que tem como objetivo definir todas as atividades necessárias para o desenvolvimento de *software* (Stoica, Mircea e Ghilic-Micu, 2013). Na Figura 2-1 podemos ver as diferentes etapas de um ciclo, que de seguida apresentamos.


**Figura 2-1: Ciclo de vida do desenvolvimento do *software***
**Fonte: (Stoica, Mircea e Ghilic-Micu, 2013)**

Etapas do ciclo:

- Análise e planeamento – A análise de requisitos é a etapa mais importante, uma vez que todo o projeto depende deste levantamento e é realizada por elementos séniores em conjunto com o cliente. A informação recolhida ajuda no planeamento do projeto que procura garantir a sua qualidade e a identificação dos riscos;

- Definição de requisitos – Após a análise é necessário clarificar e documentar todos os requisitos e obter aprovação por parte do cliente;

- Arquitetura do projeto – Com os requisitos especificados, é projetado o produto final de forma a alcançar a melhor arquitetura possível. Deve ser revista por todos os *stakeholders* e ser avaliada com base no risco, robustez, modularidade, orçamento, tempo e a melhor abordagem a ter no projeto;

- Desenvolvimento do produto – Nesta etapa, os requisitos são transformados em código-fonte. As equipas devem seguir diretrizes de codificação definidas pela organização;

- Testes – Nesta fase são realizados testes ao produto pelas equipas de qualidade, validando se cumprem todos os padrões de qualidade definidos, para poderem ser entregues;

- Implementação e manutenção – **N**esta etapa o produto é entregue e configurado no cliente e com o *feedback* recolhido. É necessário proceder com manutenções solicitadas pelo cliente.

## 2.3 Metodologias Tradicionais

Na década de 70, o desenvolvimento de *software* carecia de organização, planeamento e estrutura originando com alguma frequência, produtos de má qualidade que não correspondiam às reais necessidades dos clientes (Boehm e Turner, 2003). Perante este cenário, as empresas sentiram necessidade de melhorar a sua atividade, começando a aplicar métodos de desenvolvimento de *software* (Robiolo e Grane, 2014).

Nesta circunstância, surgiram as metodologias tradicionais, onde todos os processos se encontram muito bem definidos. Cada fase é muito bem especificada e todas as etapas são essenciais para o seu desenvolvimento. É necessário que todas as fases sejam executadas de forma sequencial e só se inicia uma nova quando a anterior se encontra concluída, para que assim esteja garantida a qualidade do produto.

Quando surgiram estas metodologias, o paradigma de desenvolvimento era muito diferente do atual, onde era necessário grande investimento, o que limitava o pedido de alterações a projetos, não havendo sequer ferramentas de apoio ao desenvolvimento. Então, a forma encontrada para minimizar os problemas foi documentar tudo corretamente, antes de se iniciar o desenvolvimento (Pressman, 2005). Todavia, a gestão de projetos com metodologias tradicionais é ainda um método bastante usado (Boehm e Turner, 2003).

Segundo Pressman (Pressman, 2005), os modelos tradicionais mais usados são:

- Modelo em Cascata – Este modelo segue uma abordagem sequencial de fases para o desenvolvimento do produto. Começa pelo levantamento de requisitos do cliente, segue-se o planeamento, o desenvolvimento do produto e, por fim, a manutenção do *software* entregue;

- Modelo Incremental – Surge para melhorar o modelo em cascata, pois permite entregar partes do projeto enquanto este se realiza. Embora seja dividido em incrementos, cada incremento passa pelas fases do modelo em cascata. Por norma, o primeiro incremento

é o núcleo do sistema, implementando os requisitos mais básicos e aprendendo com este para os próximos incrementos (Miguel, 2003);

- Modelos Evolucionários – Estes modelos são iterativos. Inicialmente é realizado um levantamento das necessidades do cliente, e através de diversas iterações são implementados pequenos conjuntos de requisitos, proporcionando ao cliente acompanhar a evolução do projeto. Um modelo muito conhecido é o modelo em espiral, que assenta em seis "atividades" (Miguel, 2003):

    1. Planeamento, para determinar os objetivos;
    2. Análise de risco, para identificar e resolver riscos;
    3. Realização de protótipos, afirmando-se como uma das maiores desvantagens deste modelo, uma vez que o resultado final pode ser diferente e o cliente muitas vezes, julga ter já um produto e afinal tem um protótipo;
    4. Desenvolvimento do produto;
    5. Testes e disponibilização do produto;
    6. Entrega e avaliação feita pelo cliente.

Uma vez que o foco desta dissertação são as metodologias ágeis, não serão descritos em pormenor todos estes métodos, sendo apenas exposto, em particular, o modelo em cascata. Apesar de ser dos primeiros, é o mais conhecido e ainda muito utilizado, sendo paradigmático de uma era (Alshamrani e Bahattab, 2015). Além disso, os restantes métodos já foram descritos anteriormente.

### 2.3.1 Modelo em Cascata

O modelo em cascata (ver "diagrama" na Figura 2-2) foi proposto em 1970 por Winston W. Royce (Royce, 1970), mas o termo Waterfall (cascata) surge mais tarde, sugerido por Bell e Thayer (Bell e Thayer, 1976), citando o artigo anteriormente publicado por Royce onde descreve o modelo formalmente. Este modelo baseia-se num processo sequencial, no qual o projeto só passa para a fase seguinte quando a fase anterior se encontra completamente desenvolvida. Para muitos autores, como Alshamrani & Bahattab, 2015; Mohammed,

Munassar, & Govardhan, 2010; Mujumdar, Masiwal, & Chawan, 2012 é o modelo tradicional mais usado nas empresas (Alshamrani e Bahattab, 2015), (Munassar e Govardhan, 2010), (Mujumdar, Masiwal e Chawan, 2012).

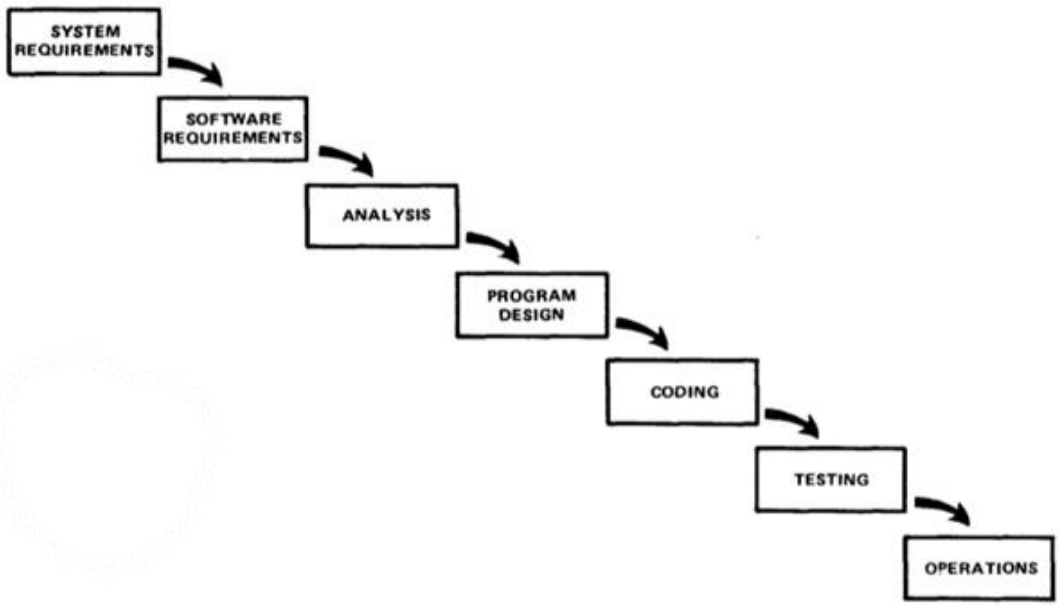


**Figura 2-2: Modelo em cascata**
**Fonte: (Royce, 1970)**

O modelo em cascata é uma sucessão de etapas bem definidas, sendo elas (Miguel, 2003):

- Requisitos do sistema – São recolhidas as informações possíveis junto do cliente, de forma a estabelecer todas as necessidades e os requisitos globais;

- Requisitos do *software* – Foca-se na identificação de todas as funções, restrições e objetivos para a construção do *software*, junto dos utilizadores finais, o que obriga muitas vezes os engenheiros de sistemas a compreenderem a área de negócio;

- Desenho dos programas – Modulação técnica para o problema. Este processo tem diversos passos que se convertem em quatro atributos: estrutura de dados, arquitetura de *software*, interface e algoritmos;

- Codificação – Nesta fase é transformado todo o passo anterior numa linguagem de programação;

- Testes – Servem para certificar que as especificações foram bem implementadas e encontram-se de acordo com os requisitos do negócio. Após esta fase, o produto é entregue ao cliente;

- Manutenção – Depois do produto ser entregue, são muitas vezes detetados erros que não foram visíveis na fase de testes. Também podem surgir novas necessidades do cliente, levando a alterações que são efetuadas no âmbito do suporte e manutenção do sistema. Geralmente, esta é a fase mais longa do ciclo de vida de um projeto.

Segundo Sommerville (Sommerville, 2007), o desenvolvimento de *software* tradicional em cascata assenta em extensa documentação, algo que pode atrasar os projetos e a indefinição dos requisitos pode mesmo comprometer esses projetos (Sommerville, 2007). Porém, pode ser uma mais-valia para projetos onde é necessário um grande controlo de qualidade e a documentação seja uma preocupação constante (Avison e Fitzgerald, 2003).

Atualmente, os projetos de desenvolvimento de *software* decorrem a ritmo muito veloz e com constantes pedidos de modificações. Por isso, o modelo em cascata não se enquadra a 100% nas mudanças. No entanto, foi um modelo que influenciou o surgimento de muitos outros modelos tradicionais (Pressman, 2005).

### 2.3.2 Falhas nas Metodologias Tradicionais

As abordagens tradicionais seguem um conjunto de processos pré-determinados, onde é produzida documentação. O sucesso alcançado com os projetos com estas abordagens, nomeadamente o modelo em cascata, depende do conhecimento de todos os requisitos no início, o que não é simples nos dias de hoje. Um problema apontado reside no facto dos requisitos serem todos fechados antes do início do desenvolvimento, sendo difícil a implementação de mudanças ao longo do ciclo de vida do projeto (Leau *et al.*, 2012), (Williams e Cockburn, 2003).

Segundo Cohn (Cohn, 2005), algumas das razões para as metodologias tradicionais falharem são:

- As atividades não terminam antes do tempo planeado, ou seja, o responsável pela execução tende a estender o tempo inicialmente estipulado para cada atividade. Em algumas organizações, acabar antes do planeado pode levar o gestor de projeto a ponderar que fez um mau planeamento ou que todas as atividades irão terminar antes;

- O atraso de uma atividade implica que as suas sucessoras possam iniciar mais tarde, isto se a sua folga[2] for inferior ao atraso. Mesmo que uma tarefa termine mais cedo, mas o seu início dependa de outras, só pode iniciar quando todas terminarem;

- As atividades não são desenvolvidas por prioridades, uma vez que o cliente só terá acesso ao projeto quando este estiver terminado. Um problema frequente é o momento de fecho de um projeto, ou seja, quando não foi possível realizar todo o âmbito, a tendência é ajustá-lo ao âmbito realizado. Estas situações determinam em muitos casos, a não serem entregues as funcionalidades mais importantes para o cliente;

- A gestão tradicional ignora a incerteza ao assumir que os requisitos, inicialmente definidos, levam a uma satisfação total por parte dos clientes;

- A atribuição de várias tarefas, em simultâneo, a um recurso tende a diminuir a produtividade, uma vez que o foco do colaborador tende a dispersar-se por diversas atividades;

- O cliente não retira partido do término de uma atividade.

## 2.4 *Standards* de Gestão de Projetos

As melhores práticas, padrões e diretrizes, não são só de um setor ou organização e podem ser aplicadas em qualquer projeto. Organizações como o Project Management Institute (PMI) têm criado e compilado o corpo de conhecimento necessário para a gestão de projetos (Meredith e Mantel Jr, 2011).

Os *standards* com maior aceitação são o PMBOK e PRINCE2. De acordo com o PMI existem mais de 500.000 certificações em 180 países diferentes. Já o PRINCE2 é especialmente popular em Inglaterra, Austrália e em alguns países da Europa (Matos e Lopes, 2013), (Karaman e Kurt, 2015), (Jamali e Oveisi, 2016).

[2] Folga- É o tempo que uma tarefa pode atrasar sem que comprometa o caminho crítico (é o caminho mais longo de um projeto). Assim, as tarefas fora do caminho crítico podem ter folga.

Os autores Matos e Lopes conceberam uma análise comparativa de PMBOK vs PRINCE2 onde verificaram os pontos apresentados na Tabela 2-1.

**Tabela 2-1: PMBOK vs PRINCE2**
**Fonte: (Matos e Lopes, 2013)**

| | PMBOK | PRINCE2 |
|---|---|---|
| Documentação | Adaptada ao projeto | Adaptada ao projeto |
| Definição de Projeto | É um esforço temporário realizado para criar um produto, serviço ou resultado singular | É um ambiente de gestão criado com a finalidade de entregar um ou mais produtos de acordo com um dado comercial específico |
| Padrão Internacional | IEEE 1490-2003, fornece fundamentos de gestão de projetos | É um método estruturado, considerado um padrão e reconhecido nos setores público e privado |
| Metodologia | Descritiva, explica as técnicas de gestão de projetos | Perspetiva, detalha como as técnicas de gestão devem ser estruturadas e implementadas |
| Orientação | Orientado aos processos utilizados no desenvolvimento do projeto | Orientado ao produto final, com foco na entrega e qualidade bem sucedida |
| Gestor de projeto | Pessoa responsável pelo cumprimento dos objetivos | O gestor do projeto tem autoridade para "correr" o projeto mas reporta sempre ao conselho de projeto |
| Grupos de processo | 5 Grupos de processo e 47 processos | 7 Processos e 35 atividades |

Os autores Karaman e Kurt, realizaram uma comparação mais detalhada dos processos, que pode ser consultada na Tabela 2-2.

**Tabela 2-2: Processos - PMBOK vs PRINCE2**
**Fonte: (Matos e Lopes, 2013)**

| PRINCE2 Processos | PMBOK Cobre | PMBOK Processos | PRINCE2 Cobre |
|---|---|---|---|
| *Starting Up Project* | *Initiating* (parcialmente) | *Initiating* | *Starting Up Project* |
| *Directing a Project* | - | *Planning* | *Initiating a Project* + *Starting Up Project* (Parte) |
| *Initiating a Project* | *Planning* + *Initiating* (parcialmente) | *Executing* | *Controlling a Stage* + *Managing product delivery* (Partes) |
| *Controlling a Stage* | *Executing* + *Monitoring and Controlling* | *Monitoring and Controlling* | *Controlling a Stage* + *Directing a Project* (monitorização e controlo de atividades) |

| PRINCE2 Processos | PMBOK Cobre | PMBOK Processos | PRINCE2 Cobre |
|---|---|---|---|
| *Managing product delivery* | *Executing* (parcial) | *Closing* | *Managing a stage boundary + Closing a project* |
| *Managing a stage boundary* | *Closing* | - | |
| *Closing a project* | | - | - |

O PMBOK e o PRINCE2 têm diferentes abordagens para a realização de uma tarefa, sendo praticamente impossível estabelecer um padrão entre os processos (Matos e Lopes, 2013).

No âmbito deste trabalho será utilizado o PMBOK, uma vez que é o *standard* com mais adesão internacional. Além disso, uma das suas principais vantagens do PMBOK é ser bastante genérico, podendo ser aplicado à maioria dos projetos (PRIKLADNICKI e ORTH, 2009).

## 2.5 Estado das Metodologias Ágeis

Quando se aborda o tema das metodologias ágeis, é fundamental referir o Manifesto Ágil, onde dezassete pioneiros estabeleceram os doze princípios que fundamentam o desenvolvimento ágil e os quatro valores fundamentais (Agile, 2001).

Analisar e compreender os quatro valores fundamentais representa um grande desafio, uma vez que estes são compostos sempre por duas partes, uma escrita a negrito representando os princípios ágeis e outra sem negrito representando os princípios que as metodologias tradicionais defendem, por exemplo - "**Indivíduos e interações** mais do que processos e ferramentas"(Agile, 2001). Todavia, um passo relevante para podermos proceder ao estudo das metodologias, é atribuir mais importância às palavras a negrito, não esquecendo os valores representados pelas palavras que se encontram sem negrito (Agile, 2001), (*Examining the Agile Manifesto*, 2017).

Os quatro valores fundamentais das metodologias ágeis são os seguintes:

1. "**Indivíduos e interações** mais do que processos e ferramentas" - O *software* é desenvolvido por equipas que englobam programadores, *testers* de *software*, gestores de projetos e clientes. Estes indivíduos necessitam de trabalhar em conjunto, e manter constante comunicação entre eles (*Examining the Agile Manifesto*, 2017), (Ocamb,

2013), sendo este o ponto mais importante, sem descartar as ferramentas a usar (*Examining the Agile Manifesto*, 2017);

2. "***Software* funcional** mais do que documentação abrangente" - Muitos projetos com requisitos extensos falharam (Ocamb, 2013). Porém, a documentação é importante, sendo um guia fundamental para a perceção de como funciona o *software* (*Examining the Agile Manifesto*, 2017). Portanto, a filosofia das metodologias ágeis consiste em efetuar a quantidade necessária de documentação, uma vez que uma documentação exaustiva muda frequentemente aquando da sua implementação e o cliente dá mais valor ao ver o *software* ser entregue incrementalmente (Ocamb, 2013).

3. "**Colaboração com o cliente** mais do que negociação contratual - Este ponto pode ser um dos que mais gera discordância nas metodologias ágeis, uma vez que é necessário existirem contratos. Os contratos devem ser flexíveis, sendo necessário, muitas vezes, explicar ao cliente as necessidades de mudança que podem existir (Ocamb, 2013). Para um cliente é muito mais simples ver o *software* em funcionamento do que ler documentação (*Examining the Agile Manifesto*, 2017).

4. "**Responder à mudança** mais do que seguir um plano" - Durante o decorrer de um projeto, os elementos das equipas vêm frequentemente alteradas as suas prioridades de entrega. O próprio ambiente do cliente muda, ou seja, a mudança é uma realidade na construção de *software*. No entanto, não é habitual um projeto não ter um plano, devendo ser maleável e existir espaço para a mudança (*Examining the Agile Manifesto*, 2017).

Uma vez que existem diversas metodologias ágeis, é necessário entender quais as que têm sido aceites pelas empresas, assim como as que na literatura têm sido recentemente estudadas.

O autor Roger S. Pressman (Pressman, 2005) investigou diversos métodos como XP, Adaptive Software Development, Scrum, Dynamic Systems Development Method, Crystal, Feature Drive Development, Lean Software Development, Agile Modeling e Agile Unified Process. Mas para ele, o modelo mais usado para o desenvolvimento de *software* é XP, que começou a ser expandido no final dos anos de 1980 por Kent Beck. Outros estudos indicam que uma das metodologias mais usadas é Scrum e tem sido a mais aceite pelas empresas (Salameh, 2014), (Tanveer, 2015).

A empresa Versionone tem produzido relatórios ao longo dos anos e neles podemos verificar quais as metodologias usadas nos projetos. Na Figura 2-3 pode-se visualizar que a mais usada é Scrum, seguida de três abordagens híbridas.

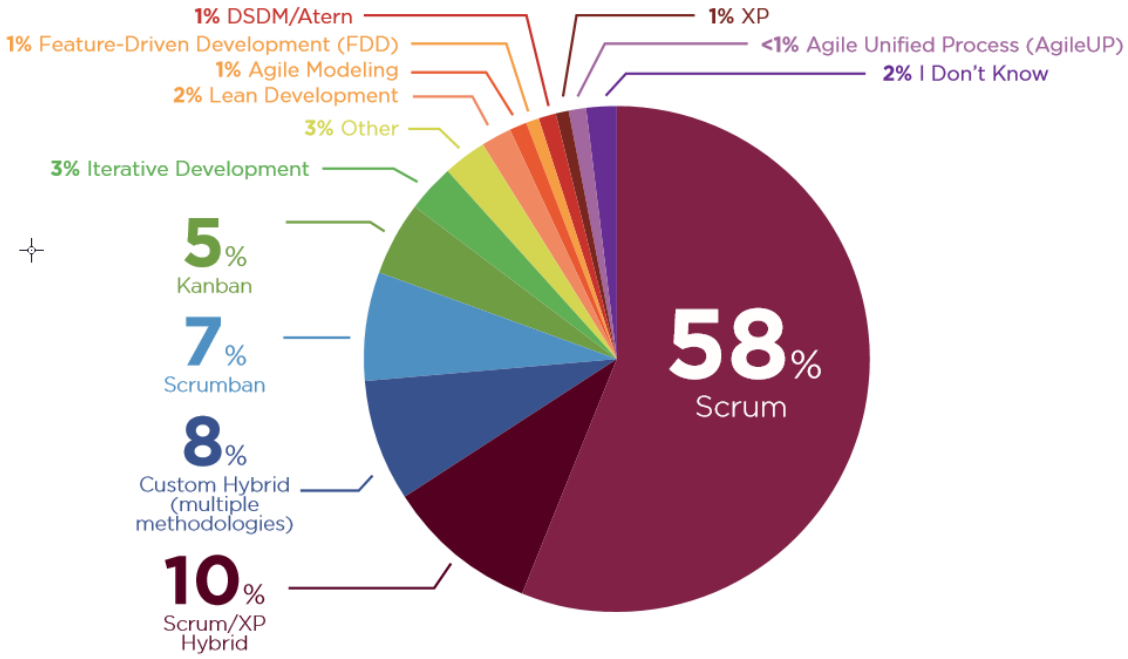


**Figura 2-3: Metodologias ágeis e a sua utilização**
**Fonte: (Versionone, 2016)**

A Tabela 2-3 exibe, resumidamente, as metodologias mais aplicadas desde 2006 até 2015, sendo mencionadas somente aquelas, que durante este período, obtiveram pelo menos 5% de utilização num ano. Facilmente notamos que Scrum foi sempre a mais usada e que os utilizadores de XP diminuíram desde o aparecimento do modelo híbrido Scrum/XP (Mushtaq e Qureshi, 2012), similarmente o modelo Scrumban, que é híbrido (Yilmaz e O'Connor, 2016), tem tido uma forte adesão desde o seu surgimento, sendo mesmo a terceira mais utilizada uma metodologia híbrida (Versionone, 2016).

**Tabela 2-3: Resumo das metodologias mais usadas**
**Fonte: (Versionone, 2016)**

| Ano | Scrum | Scrum/ XP | XP | Hybrid/ Custom | DSDM | Scrumban | Kanban |
|---|---|---|---|---|---|---|---|
| 2006 | 40 | | 23 | 14 | 8 | | |
| 2007 | 37 | 23 | 12 | 9 | 5 | | |
| 2008 | 49.1 | 22.3 | 8 | 5.3 | 1.4 | | |
| 2009 | 50 | 24 | 6 | 5 | | | |
| 2010 | 58 | 17 | 4 | 5 | | 3 | |
| 2011 | 52 | 14 | 2 | 9 | | 3 | |

| Ano | Scrum | Scrum/ XP | XP | Hybrid/ Custom | DSDM | Scrumban | Kanban |
|---|---|---|---|---|---|---|---|
| 2012 | 54 | 11 | 2 | 9 | | 7 | 4 |
| 2013 | 55 | 11 | 1 | 10 | | 7 | 5 |
| 2014 | 56 | 10 | 1 | 8 | 1 | 6 | 5 |
| 2015 | 58 | 10 | 1 | 8 | 1 | 7 | 5 |

### 2.5.1 Caraterísticas Comuns das Metodologias Ágeis

Muitas das abordagens ágeis partilham algumas práticas entre elas. Os autores Firas Glaiel, Allen Moulton e Stuart Madnick identificaram sete genes comuns e chamaram a este conjunto de *"Genome of Agile"* (Glaiel, Moulton e Madnick, 2014):

- Histórias (*User Story*) - É a descrição de uma funcionalidade na perspetiva do utilizador final. Esta abordagem permite que, à medida que as histórias vão sendo escritas, possam ser implementadas e demonstradas ao cliente para recolher feedback, reduzindo erros e possibilitando detetar, numa fase inicial, possíveis melhorias;

- Iterativo e Incremental – O desenvolvimento começa antes dos requisitos estarem todos fechados, através de múltiplos incrementos onde cada um adiciona novas caraterísticas ao produto final. Em cada *sprint* é realizado o planeamento, a implementação e os testes, para que se possa entregar um produto acabado.

- Otimização – Representa a natureza adaptativa dos processos ágil. As equipas são constantemente confrontadas com mudanças e são incentivadas a adaptarem-se. Além disso, as equipas realizam frequentemente, a revisão do trabalho realizado.

- *Refactoring* – Trata-se de uma prática comum entre as equipas de programação, que consiste em reestruturar o código-fonte existente em blocos padronizados, para assim poder ser partilhado. Tem um custo associado, que é o esforço necessário para implementar a mudança no *software*.

- Integração contínua – Corresponde à junção do código-fonte produzido por todos os membros da equipa num projeto. Esta prática tem ainda associados os testes automatizados para que os programadores possam garantir que as novas funcionalidades não coloquem em causa as entregas anteriores.

- Equipas dinâmicas – Representam práticas que as equipas usam para aperfeiçoar o desempenho. A maioria das práticas incluem reuniões frequentes para que as equipas se possam organizar e que possam estabelecer a prioridade do trabalho. O facto de em muitos métodos existirem reuniões diárias, acaba por criar pressão que se traduz numa produtividade maior.

- Clientes envolvidos – O cliente encontra-se integrado na equipa do projeto e deve participar nas reuniões diárias de planeamento e demonstrações. A disponibilidade e contribuição do cliente permite diminuir os riscos e identificar, de forma precoce, os problemas ou trabalhos adicionais necessários.

### 2.5.2 Vantagens e Desvantagens das Metodologias Ágeis

Os métodos ágeis apresentam vantagens e desvantagens, apresentadas mais abaixo. Na Figura 2-4 podemos observar um resumo de algumas vantagens indicadas por Kumar e Bathia (Kumar e Bhatia, 2012).

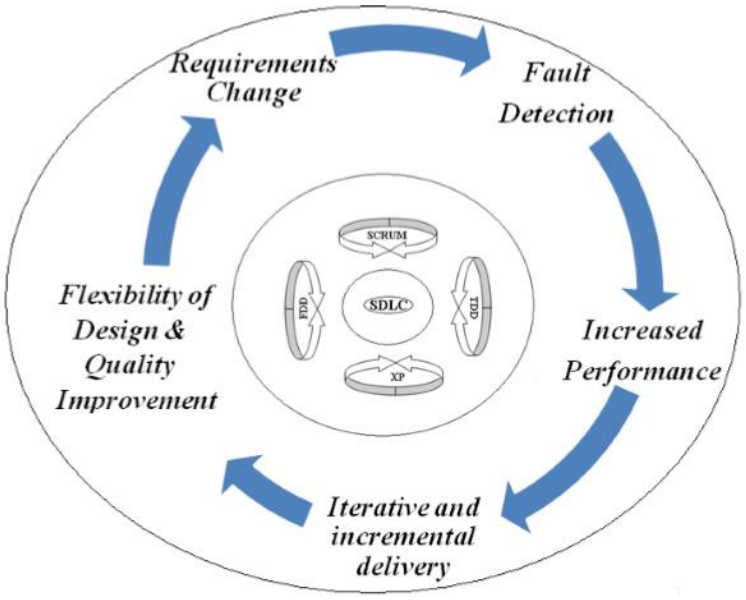


**Figura 2-4: Benefícios das metodologias ágeis**
**Fonte: (Kumar e Bhatia, 2012)**

- Vantagens

  - Alteração de requisitos – No planeamento do *sprint*, todos os requisitos necessários para o seu sucesso são revistos e a participação do cliente permite ajustá-los;

- Redução de falhas - Os testes são realizados durante a iteração, antecipando os erros que são corrigidos mais cedo. Além disso, após cada iteração, é recolhido o *feedback* do produto entregue, contribuindo para o aumento da qualidade;
- Aumento do desempenho – As atividades como as reuniões diárias permitem a troca de informação entre todos os membros. A comunicação representa um ponto-chave para o aumento de desempenho;
- Entregas iterativas e incrementais – O projeto é dividido em pequenas entregas, para reduzir o risco e obter feedback do cliente o mais cedo possível. Cada iteração é caracterizada por incluir as fases de análise, desenvolvimento e testes;
- Satisfação do cliente – Uma vez que o cliente é envolvido em todas as fases, contribui para que as entregas correspondam a efetivas necessidades;
- Adapta-se à mudança – As abordagens ágeis incentivam a mudança em qualquer estado do desenvolvimento.

- Desvantagens
  - Não se adequa a projetos grandes - O desenvolvimento ágil encontra-se adaptado a pequenas equipas e iterações curtas. Pode não ser suficiente para desenvolver funcionalidades, na sua totalidade, numa iteração;
  - Requisitos pouco claros – Geralmente, os requisitos só são especificados na iteração onde serão implementados, o que dificulta o controlo de orçamento, já que o planeado foi feito com base em dados iniciais, que não contêm todo o detalhe. De forma similar, é apontado o facto de os requisitos serem discriminados à medida que são necessários. Isto dificulta a entrada de novos elementos na equipa, pois não entendem que funcionalidades existem e como funcionam;
  - Mudança de requisitos – Uma das maiores vantagens é também apontada como desvantagem, na medida que é necessário a equipa dispor de flexibilidade para fazer mudanças, conforme o pedido, para garantir a entrega do produto certo. Este

princípio pode criar projetos sem fim, já que se torna muito mais difícil determinar o requisito de entrega, uma vez que o *definition of done*3 muda constantemente;

- Pouca documentação – Embora desenvolver pouca documentação represente menores custos para o projeto, dificulta a integração de novos membros na equipa do projeto;

- Recursos experientes – As decisões são maioritariamente tomadas por elementos séniores. Quando são tomadas por elementos juniores, é fundamental a presença de um sénior.

### 2.5.3 Desafios das Metodologias Ágeis

No âmbito do desenvolvimento de *software*, as metodologias ágeis são relativamente recentes e têm sido apresentados alguns desafios para estas crescerem, sendo de destacar o trabalho desenvolvido pelo autor David Rico (Rico, Sayani e Sone, 2009).

- Formação – Os métodos ágeis são simples, mas não são fáceis de concretizar, pois é necessário alterar os hábitos de trabalho. Apesar de serem simples de iniciar, é um desafio a sua correta utilização, porque são baseados na colaboração com o cliente, trabalho de equipa, desenvolvimento iterativo e adaptável. A maioria destes valores são intangíveis, como construir relacionamentos e ter uma atitude flexível. Por isso, é necessária formação em pontos chaves como a resistência à mudança, como responder a perguntas ou corrigir mal entendidos;

- Métodos ágeis híbridos – Existem muitos tipos de metodologias ágeis, o que origina perguntas como "Quais as melhores práticas?" "O melhor método?" "São todos eficazes?" E a resposta deve ser: usar a ferramenta certa para o trabalho certo. Michel dos Santos (Soares, 2004), na conclusão de um estudo sobre a comparação de metodologias, afirmou que é necessário encontrar formas de eliminar alguns pontos fracos das metodologias, resumindo a ideia na seguinte citação:

[3] *Definition of Done* (DoD) – Lista de atividades (código, testes unitários, testes de integração, etc.), que adicionam valor demonstrável para o projeto

*"O desafio futuro das metodologias ágeis é encontrar formas de eliminar alguns dos seus pontos fracos, como a falta de análise de riscos, sem torná-las metodologias pesadas."(Soares, 2004)*

- Complexidade e escalabilidade – Alguns autores como David Rico, Stoica, Mircea e Ghilic-Micu afirmam que os métodos ágeis não se encontram desenhados para projetos grandes e complexos. Segundo David Rico esta afirmação é um mito. Ainda que seja um desafio manter as equipas organizadas e coordenadas durante um projeto grande, é preciso estabelecer e manter estratégias para a integração do código das diversas equipas, coordenação e gestão do risco para o produto (Glazer *et al.*, 2008);

- Documentação – Um ponto negativo apontado aos métodos ágeis é a ausência de documentação, mas na realidade são produzidos documentos com o essencial. No lugar de longos documentos explicativos, o código deve estar documentado, para que as equipas de manutenção possam desenvolver o seu trabalho.

- Equipas virtuais – O facto de as equipas estarem dispersas geograficamente é o oposto de um dos valores das metodologias ágeis, pois indicam que as interações devem ser feitas cara a cara. Contudo, as metodologias ágeis têm nas suas raízes a produção de *software* de alta qualidade, a custos baixos. Muitas vezes, torna-se preferível ter equipas noutras localizações para superar dificuldades.

## 2.6 Sumário

Existem muitas diferenças entre as metodologias tradicionais e ágeis, nomeadamente na forma como lidam com o projeto. As tradicionais usam uma abordagem preditiva e as equipas têm de seguir um plano predefinido para todas as tarefas e atividades do ciclo de vida. Por sua vez, as metodologias ágeis são adaptativas e os requisitos encontram-se em constante mudança, sendo fechados muito próximo da sua implementação.

Uma vez que o PMBOK é bastante utilizado e reconhecido mundialmente, é fundamental aprofundar os conhecimentos deste no capítulo seguinte, e abordar alguma possibilidade de abordagens hibridas entre o PMBOK e metodologias ágeis.

# 3. PMBOK – Project Management Body of Knowledge

O guia do Project Management Body of Knowledge (PMBOK) reúne um conjunto de grupos de processos, melhores práticas e diretrizes que são genericamente aceites como padrão na gestão de projetos, sendo gerido pelo Project Management Institute (PMI).

Os conteúdos do presente capítulo baseiam-se na quinta edição do livro do PMBOK (Institute, 2013). Este capítulo encontra-se repartido em cinco secções. A primeira aborda os 5 grupos de processos; a segunda as 10 áreas de conhecimento. A terceira efetua uma comparação entre as práticas do PMBOK com as metodologias ágeis, a quarta aborda algumas metodologias híbridas e por fim, é apresentado o sumário do capítulo.

## 3.1 Grupos de Processos

O PMBOK sugere cinco grupos de processos, representados na Figura 3-1. Um grupo de processos inclui um conjunto de processos, sendo um processo um conjunto de ações e atividades ligadas entre si para alcançar um resultado. Geralmente, as saídas de um processo são as entradas de outro processo.

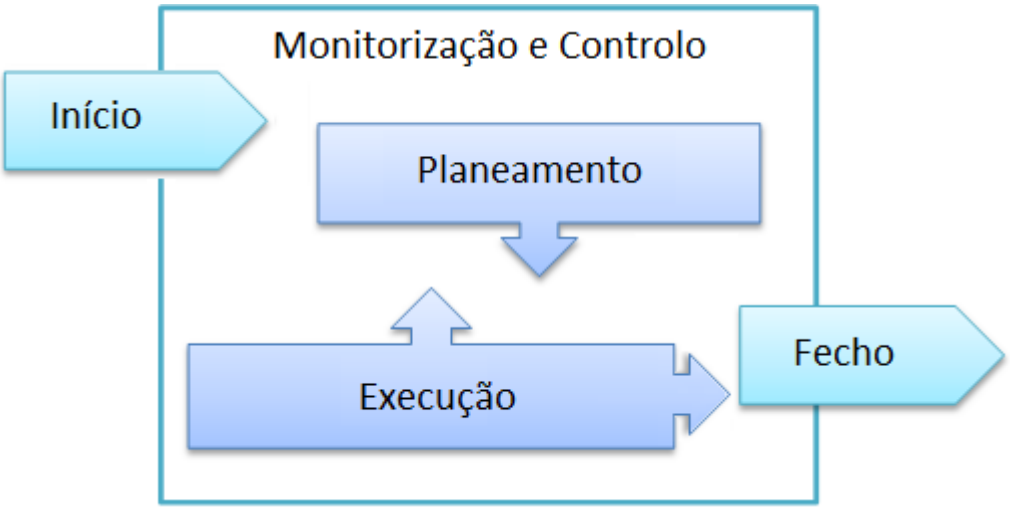


**Figura 3-1: Grupos de Processos do PMBOK**
**Fonte: (*Institute*, 2013)**

- Início – Este grupo de processos é constituído por todos os processos envolvidos no início de um projeto ou de uma fase. Neste grupo de processo, é autorizado o início do projeto, sendo o âmbito definido e os *stakeholders* identificados. Um dos aspetos mais significativos que resulta deste grupo relaciona-se com a nomeação do gestor de projeto, o que lhe confere a autoridade necessária para a gestão do projeto. Logo, os principais aspetos são: determinar os objetivos do projeto; definir o gestor do projeto; identificar os *stakeholders*; documentar as premissas e restrições e elaborar o termo de abertura do projeto;

- Planeamento – Neste grupo de processos, o projeto é planeado, de forma a detalhar e refinar todos os objetivos e melhores alternativas para os alcançar. Para o efeito, são produzidos documentos de grande detalhe, tal como uma lista discriminada de requisitos; estimativas e custos; qualidade e riscos. Os principais aspetos são: elaborar e divulgar a especificação do âmbito do projeto; desenvolver a estrutura analítica do projeto (EAP); desenvolver o cronograma; determinar a necessidade de recursos; definir como serão as aquisições; desenvolver o orçamento e desenvolver e divulgar o plano de gestão do projeto;

- Execução – O trabalho planeado anteriormente é executado. Enquanto as equipas se preocupam com a entrega do projeto, cumprindo todos os requisitos, o gestor de projeto tem de efetuar toda a coordenação de recursos. É nesta fase que grande parte do orçamento é gasto. Durante a execução, pode existir a necessidade de mudança, a qual

requer aprovação. Caso sejam necessárias mudanças substanciais, é indispensável rever o plano projetado, assim como a documentação. Os principais aspetos são: alocar e desenvolver a equipa de execução do projeto; alocar os recursos necessários para o trabalho; executar o plano de gestão do projeto e executar o trabalho do projeto;

- Monitorização e Controlo – Este grupo de processos ocorre em todo o projeto, já que é necessário para garantir o cumprimento dos objetivos, revendo e acompanhando o seu progresso. O grupo de processos de controlo do progresso permite antecipar problemas e implementar ações corretivas. Este método ocorre até ao fecho. Os principais aspetos a destacar são: medir o desempenho do projeto comparando-o com a linha de base; determinar as variações e suas recomendações de ações corretivas ou preventivas; avaliar as ações corretivas que foram aplicadas; auditar os riscos e realizar relatórios de desempenho;

- Fecho – Este grupo de processos é responsável por concluir todas as atividades no decorrer de uma fase. É fundamental a aceitação do cliente. Prevê também a possibilidade de ser feita uma revisão ao trabalho desenvolvido, para aplicar lições aprendidas em futuros projetos. Toda a documentação é arquivada. Nesta fase, os principais aspetos são: obter a aceitação formal do produto do projeto pelo cliente; arquivar os registos do projeto; documentar as lições aprendidas; formalizar o encerramento das aquisições e do projeto.

Na Figura 3-2 é possível observar as fases típicas de um ciclo de vida de um projeto, podendo ser verificada a intensidade do esforço ao longo do tempo, através da linha representada a preto. Os grupos de processos encontram-se expostos a verde claro, o início, a azul claro, o planeamento, a azul escuro, a execução, a vermelho, a monitorização e controlo e a verde o fecho. Estes repetem-se nas diversas fases em diversos níveis de intensidade ao longo de todo o projeto. No princípio, as atividades mais intensas são as de iniciação. Com o decorrer do tempo, os processos de planeamento, execução e por fim o de fecho consomem mais recursos. Já o grupo de processos, monitorização e controlo encontra-se com uma presença mais uniforme ao longo de todo o ciclo de projeto.

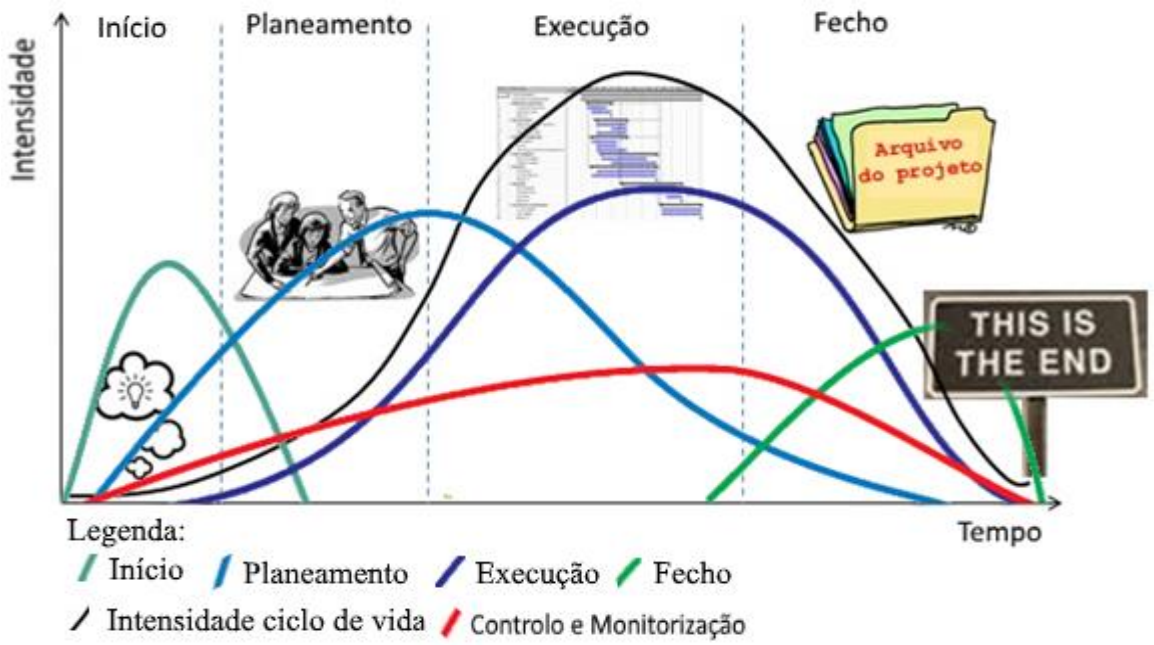


**Figura 3-2: Interação de grupos de processos num projeto**
**Fonte: baseado em Gerenciamento de projetos segundo o guia PMBOK (Bomfin, Nunes, Hastenreiter, 2012)**

## 3.2 Áreas de Conhecimento e Processos

O PMBOK define dez áreas de conhecimento, como se pode constatar, consultando o Anexo 1. Para a gestão de projetos, da conjugação dos grupos de processos com as áreas de conhecimento resultam quarenta e sete processos. Seguidamente, será apresentada uma breve descrição das áreas de conhecimento e dos seus processos.

- Integração – A gestão da integração descreve os processos que integram todos os elementos de gestão do projeto que são identificados, definidos, unificados e coordenados dentro dos grupos de processo. Para Gido e Clements, a gestão de projetos tem início com o planeamento do trabalho, para posteriormente ser executado o plano de trabalho (Gido e Clements, 2008).

    - Definir o termo de abertura – Criação do documento que formaliza a existência do projeto e atribuição de poderes ao gestor para conceder os recursos necessários às atividades do projeto;

    - Desenvolver o plano de gestão do projeto – Processo para definir, preparar e coordenar todos os planos e integrá-los na gestão do projeto;

- Executar o plano do projeto – É o processo de realizar o trabalho definido no plano de gestão do projeto e a implementação das mudanças aprovadas;
- Monitorizar e controlar o projeto – Refere-se ao acompanhamento do progresso do projeto para atingir os objetivos (deve ser sempre registado);
- Controlo de mudanças – Verifica todos os pedidos de alterações, aprova as modificações e efetua a gestão dos documentos do projeto;
- Encerrar o projeto - Encerra formalmente o projeto, finalizando todas as atividades de todos os grupos de processos.

- Âmbito – A gestão do âmbito garante que o projeto inclui todo o trabalho necessário a desenvolver, para que todos os requisitos sejam cumpridos e o projeto seja concluído com sucesso. A grande premissa desta área de conhecimento é garantir que tudo é cumprido, conforme o plano. No caso de não se estabelecer corretamente o âmbito, pode resultar em trabalho desnecessário e provocar derrapagens, quer em custos quer em calendarização (Webster, 2000).
  - Planear a gestão do âmbito – Estabelece o âmbito, validado e controlado;
  - Reunir os requisitos – Define e documenta as necessidades e requisitos das *stakeholders*;
  - Definir o âmbito – Apresenta a descrição detalhada do produto;
  - Criar a EAP (Estrutura Analítica do Projeto (WBS - *Work Breakdown Struture*)) – Lista detalhada das entregas necessárias do projeto e das atividades;
  - Validar o âmbito – Formaliza a aceitação das entregas do projeto;
  - Controlar o âmbito – Monitorização do estado do projeto e gestão das alterações à base do âmbito.
- Tempo – A gestão do tempo inclui todos os processos que permitam atingir o fim do projeto no prazo definido no planeamento. A maioria dos processos são de planeamento e somente um é de controlo.

- Planear a gestão do tempo – Estabelece os procedimentos e documentação para planear, criar, gerir, executar e controlar o cronograma;
- Definir as atividades – Identifica e documenta as ações para a realização da entrega ao cliente;
- Sequenciar as atividades – Identifica a relação entre as diversas atividades;
- Estimar os recursos das atividades – Estima os tipos e quantidades de recursos necessários para cada atividade;
- Estimar a duração das atividades – Estima o tempo necessário para a conclusão de cada atividade;
- Desenvolver o cronograma – Criação do cronograma com base na análise da sequência das atividades, duração, recursos e restrições;
- Controlar o cronograma – Controlar as alterações ao cronograma e atualizar o seu progresso.

- Custos – A gestão dos custos envolve os processos correspondentes de planeamento, estimativas, orçamentos, financiamento e controlo dos custos de forma que o projeto possa terminar dentro do orçamento.
  - Planear a gestão dos custos – Estabelecer políticas, procedimentos e documentos para planear, gerir, executar e controlar os custos dos projetos;
  - Estimar custos – Estima os custos dos recursos para cada atividade;
  - Determinar o orçamento – Junta os custos estimados para todas as atividades para criar uma linha base;
  - Controlar os custos – Monitorização do projeto. Atualiza o orçamento e efetua as alterações na linha base dos custos.
- Qualidade - A gestão da qualidade tem como objetivo garantir que o projeto satisfaz todas as necessidades para o qual foi desenhado. Para tal, devem determinar-se as políticas de qualidade, objetivos, requisitos e responsabilidades.

  - Planear a qualidade – Identificar os requisitos de qualidade do projeto e documentar para demonstrar a conformidade;
  - Realizar a garantia da qualidade – Conjunto de processos/tarefas, cuja realização permite ter uma razoável expectativa de qualidade do produto final;
  - Controlo da qualidade – Verificação da conformidade do produto obtido com as especificações aprovadas. Deve ser efetuado o registo dos resultados da execução das atividades de qualidade para avaliar a *performance* e recomendar as mudanças necessárias.

- Recursos Humanos – A gestão de recursos humanos descreve os processos que organizam e gerem a equipa do projeto. Os processos têm como objetivos compreender o tipo de perfil de pessoas necessárias, de como motivá-las, atribuir a cada elemento as suas responsabilidades e as ferramentas necessárias para a execução do projeto.
  - Planear a gestão dos recursos humanos – Identifica e documenta as responsabilidades, funções, habilidades e cria um plano de gestão de pessoas;
  - Adquirir a equipa de projeto – Valida a disponibilidade dos elementos para constituir a equipa necessária à conclusão das atividades do projeto;
  - Desenvolver a equipa de projeto – Desenvolver e melhorar as competências dos recursos da equipa para melhorar o desempenho do projeto;
  - Gerir a equipa de projeto – Acompanhar o desempenho dos recursos, fornecer *feedback* e resolver os problemas decorrentes do projeto.

- Comunicação – A gestão da comunicação descreve os processos relativos à geração, armazenamento, coleta, disseminação e destino da informação a quem de direito e em tempo oportuno. Os processos definem como as comunicações ocorrem, qual a sua periodicidade, como é distribuída a informação, como se gerem as expetativas dos interessados e se mede o seu grau de satisfação. Esta área compreende a geração dos relatórios que permitem executar o acompanhamento do projeto. Segundo Phillips, a comunicação é a chave entre pessoas, ideias e informação (Phillips, 2007).

  - Planear a gestão das comunicações – Identifica as necessidades de informação e requisitos dos *stakeholders*;
  - Gerir as comunicações - Disponibiliza as informações necessárias para os *stakeholders*. É da sua responsabilidade reunir, criar, distribuir, armazenar, recuperar e descartar a informação;
  - Controlar as comunicações – Controlar e monitorizar as informações no decorrer de todo o projeto para que todas as partes sejam atendidas.

- Riscos – A gestão dos riscos inclui os processos de planeamento, identificação, análise, planeamento de respostas e monitorização de riscos. A premissa desta área é identificar todos os riscos que podem afetar os projetos, procurando soluções para mitigar, ou até mesmo, eliminar o impacto das situações negativas e obter uma vantagem total das oportunidades geradas pelos riscos positivos.
  - Planear a gestão dos riscos - Define como conduzir todas as atividades de gestão do risco;
  - Identificar riscos – Determinar os riscos que podem afetar o projeto e documentar as suas caraterísticas;
  - Realizar a análise qualitativa dos riscos – Analisar as probabilidades do risco, assim como o seu impacto;
  - Realizar a análise quantitativa dos riscos – Investigar quantitativamente o impacto dos riscos para o projeto;
  - Planear a resposta ao risco – Desenvolve opções e ações para reduzir as ameaças e portanto cumprir os objetivos do projeto;
  - Controlar os riscos – Implementa o plano de resposta aos riscos, monitoriza e controla os riscos durante todo o projeto.

- Recursos – A gestão de aquisições inclui todos os processos de aquisição de produtos, serviços e partes externas à equipa de desenvolvimento. Os processos desta área têm

como objetivos determinar a quem se pode comprar, como selecionar o fornecedor, a gestão de contratos e pagamentos.

  - Planear a aquisição de recursos – Documenta as decisões de compra/aquisição, especifica a abordagem e identifica potenciais fornecedores;
  - Conduzir a procura – Obtém as respostas dos fornecedores, seleciona um fornecedor e elabora o contrato;
  - Controlar as aquisições - Gere as relações de aquisição, monitoriza a *performance* dos contratos e realiza alterações, se necessário;
  - Encerra aquisições – Finaliza as aquisições do projeto.

- *Stakeholders* – Mais recentemente, esta foi uma área de conhecimento reconhecida como tal no PMBOK e encontra-se relacionada com a priorização de todas as pessoas que possam ter influência no projeto, direta ou indiretamente. A correta identificação de todas as pessoas pode garantir o sucesso do projeto.
  - Identifica os s*takeholders* – Identifica pessoas, grupos ou organizações que podem ter interesse nos resultados do projeto;
  - Planear a gestão dos *stakeholders* – Desenvolve estratégias para garantir o envolvimento e compromisso de todos os *stakeholders* no decorrer do projeto;
  - Gerir os *stakeholders* – Monitoriza o relacionamento entre os *stakeholders* e ajusta as estratégias para que todos estejam empenhados.

## 3.3 PMBOK com Metodologias Ágeis

Os grupos de processos do PMBOK podem ser comparados relativamente às correspondentes abordagens das metodologias ágeis. De seguida, são descritas algumas práticas comuns identificadas por Schwalbe (Schwalbe, 2012):

- Início – Determinação dos papéis/ decisão do número de *sprints*;

- Planeamento – Criação das histórias e colocação por prioridade em *Backlog*, reuniões diárias, planeamento detalhado por *sprin*t, definição de concluído;
- Execução – Finalização de tarefas durante o *sprint*, incremento produzido por sprint;
- Monitorização e Controlo – Resolução de problemas, demos no fecho de cada *sprint*, reunião de revisão de *sprint*;
- Fecho – Reflexão sobre o *sprint*, para identificar melhorias futuras.

Os autores Jeffery Landry e Rachel McDaniel (Landry e McDaniel, 2016), com base no trabalho de Schwalbe (Schwalbe, 2015), conceberam uma comparação entre as áreas do PMBOK e alguns pontos-chave das metodologias ágeis que se apresentam na Tabela 3-1. Neste quadro, é explicado para cada área de conhecimento do PMBOK os pontos-chave correspondentes das metodologias ágeis.

**Tabela 3-1: Tópicos das metodologias ágeis por áreas de conhecimento do PMBOK**
**Fonte: (Landry e McDaniel, 2015)**

| Áreas de conhecimento do PMBOK | Pontos chave das metodologias ágeis |
|---|---|
| Integração | Valores do Manifesto Ágil e jogos de planeamento |
| Âmbito | *User Stories* / *Backlog* |
| Tempo | *Sprints*, Jogos de planeamento, gráfico de *Burndwon*, quadros Kanban |
| Custos | Métodos de avaliação Financeira (NPV, ROI) |
| Qualidade | Definição de concluído (*Definition of Done*), |
| Recursos Humanos | Equipas Scrum |
| Comunicação | Reuniões diárias |
| Riscos | Ajustes de *Backlog*, entregas progressivas |
| Recursos | Métodos de contratação ágeis |
| *Stakeholders* | Donos do produto, *Scrum Master* |

- Integração – Para as metodologias ágeis, destacam-se os quatro valores fundamentais e os doze princípios que foram definidos no Manifesto Ágil (Beck, 2000). Uma das ferramentas mais utilizada nas metodologias ágeis é o jogo do planeamento, onde são identificados e priorizados os pedidos do cliente, sendo fundamental a participação de todos os *stakeholders*. Esta abordagem é fundamental na maioria das metodologias ágeis e integra quase todas as áreas, como âmbito, tempo, qualidade, custos e riscos (Lindstrom e Jeffries, 2004).

- Âmbito – A gestão do âmbito nas metodologias ágeis passa pela identificação iterativa dos requisitos, descritos em *user stories* e escritas na linguagem que o cliente usa no seu quotidiano. O conjunto de todas as histórias forma o *Backlog*, as quais devem estar ordenadas por ordem de importância. Contrariamente à gestão tradicional, o âmbito é altamente flexível.

- Tempo – Nas metodologias ágeis é difícil encontrar um conceito que tenha o mesmo sentido que no PMBOK. Em vez de existir um planeamento temporal de tempos para todo o trabalho, é dividido em *sprints* de duas a quatro semanas. A equipa de desenvolvimento compromete-se na concretização de uma entrega com algumas histórias.

- Custos – Como se pode observar na Figura 3-3, nas metodologias ágeis, o tempo e os custos são fixos e o âmbito é variável podendo ser ajustado ao longo do tempo, nomeadamente a cada iteração.

- Qualidade - Os projetos desenvolvidos com metodologias ágeis preocupam-se, em particular, com a satisfação do cliente. Em todas as iterações é fundamental o cliente realizar testes de aceitação. O principal objetivo é que os possíveis *Bugs* sejam detetados durante o desenvolvimento (o número de *Bugs* é a medida de qualidade).

- Recursos Humanos – A equipa que compõe o projeto é o principal recurso para o seu sucesso. Para tal, o gestor do projeto requer capacidades ao nível das relações pessoais, levando-as a superar os seus desafios.

- Comunicação – Nas metodologias ágeis, os *stakeholders* trabalham em conjunto com as equipas de desenvolvimento, sendo importante que todos estejam reunidos em todas as reuniões, como, por exemplo, no planeamento de um *sprint*, quando nos referimos à metodologia Scrum.

- Riscos – As abordagens ágeis preveem entregas regulares, de forma a reduzir o risco de aceitação do cliente. Além disso, este participa nas especificações de todas as histórias e na sua priorização, de modo a reduzir a incerteza e a ir ao encontro das necessidades reais.

- Recursos – Uma vez que nas metodologias ágeis os requisitos apenas se clarificam aquando da sua implementação, não é possível efetuar um planeamento das aquisições necessárias nem de as acompanhar, à semelhança do que é feito no PMBOK.

- *Stakeholders* – Para as metodologias ágeis, a satisfação do cliente encontra-se sempre em primeiro lugar e existem alguns papéis, como o de Scrum Master, que têm como função facilitar a interação entre o cliente e a equipa de desenvolvimento.

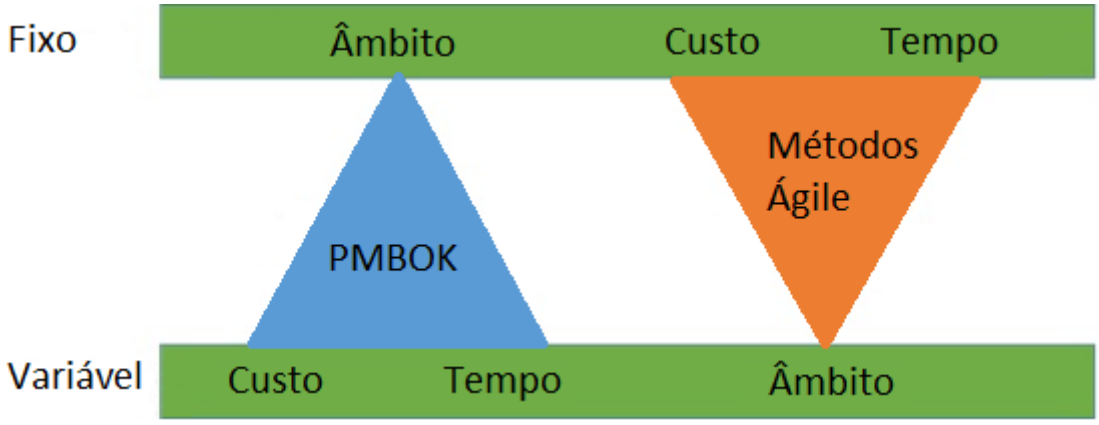


**Figura 3-3: Diferença entre PMBOK e Métodos Ágeis**
**Fonte: (Adjei e Rwakatiwana, 2010)**

## 3.4 Abordagens Híbridas entre PMBOK e Métodos Ágeis

Os autores Siddique e Hussein produziram um estudo, no qual se concentraram na gestão do risco em projetos de *software*, concluindo que se encontram a ser usadas duas abordagens: a implícita, com recurso a comunicação e colaboração, iterações curtas, entregas frequentes e *feedback* constante; a explícita, com recurso a análise SWOT, matriz de risco e gráfico de *burn down*[4] (Siddique e Hussein, 2017).

No estudo apresentado por Uikey e Suman ficou assente que as abordagens tradicionais e ágeis possuem princípios e valores similares, uma vez que ambas são direccionadas para alcançar o objetivo, liderando a equipa e obtendo a satisfação do cliente. É enfatizada a necessidade de comunicação entre o cliente e a equipa, a documentação e os testes de *software*. Neste âmbito,

[4] É um gráfico que representa o trabalho em falta face ao tempo

os autores propuseram um quadro de gestão de projetos ágeis juntamente com o PMBOK, como se pode observar na Figura 3-4 (Uikey e Suman, 2012).

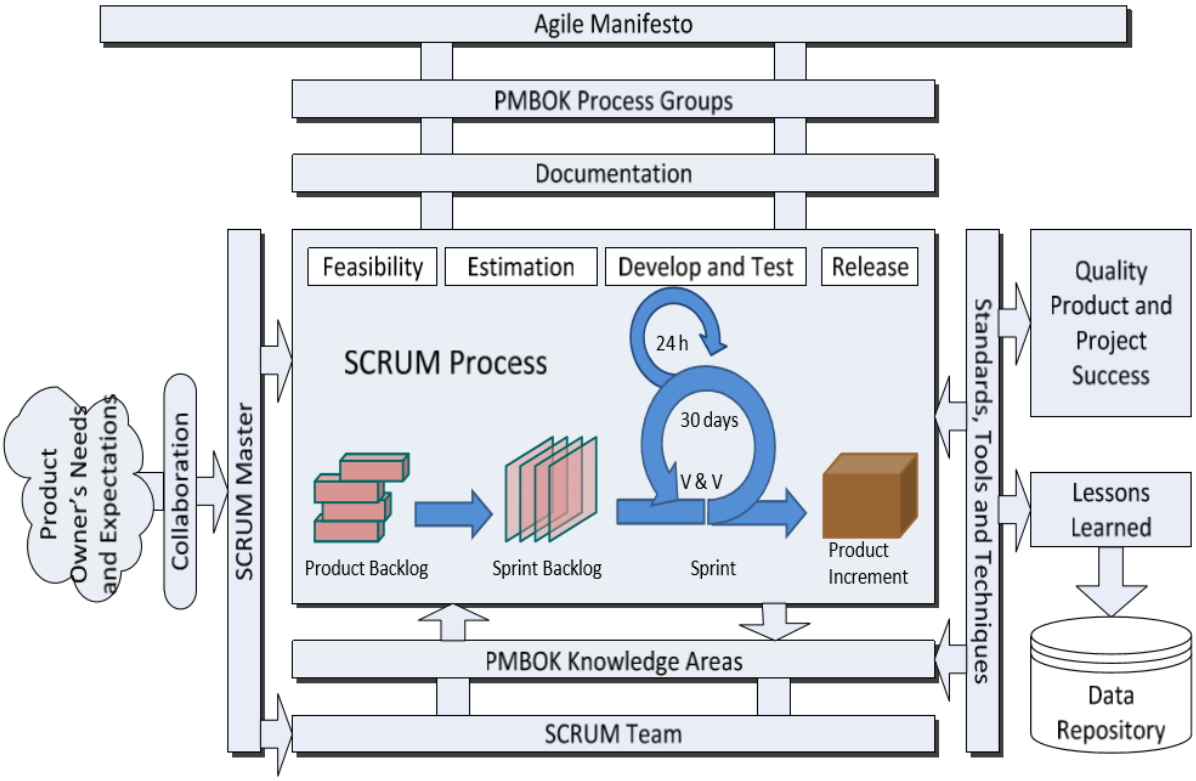


**Figura 3-4: *Framework* PMBOK e Scrum**
**Fonte: (Adjei e Rwakatiwana, 2010) (Uikey e Suman, 2012)**

O artigo de Batra (Batra *et al.*, 2010) demonstra um método híbrido utilizando Scrum com *sprints* de duas semanas*; Backlog*: satisfação do cliente e entregas frequentes com *software* sem *bugs*, medindo assim o progresso do projeto e retrospetivas. Do PMBOK foi usado o ciclo de vida do projeto, etapas de levantamento de requisitos, análise e *design* (Batra *et al.*, 2010).

Os autores Noureddine, Damodaram e Younes (Noureddine, Damodaran e Younes, 2009) criaram uma *framework* híbrida para ser aplicada a projetos de *software*, onde dividem o ciclo de vida em quatro fases. Não é focado nada no encerramento do projeto, deixando em aberto para trabalhos futuros. As quatro fases são:

- Planeamento – Recebe como *input*, o plano do projeto, a definição do gestor do projeto e todas as *stakeholders*. Tem como objetivo a construção de uma visão geral do projeto, definir os requisitos básicos e criar as estimativas;

- Inovação – Baseado no XP, visa inicialmente produzir uma versão do *software* baseado nos requisitos básicos;

- Organização – Tem como objetivo refinar a primeira versão do *software* adicionando funcionalidades específicas. Nesta fase é usado Scrum para que a equipa se possa auto organizar e ser o mais produtiva possível;

- Definição – Visa resolver os problemas de última hora e produzir a documentação necessária.

## 3.5 Sumário

Neste capítulo foram descritos os processos e áreas de conhecimento do PMBOK. Foi efetuada uma análise comparativa entre as metodologias ágeis com o PMBOK, tendo-se verificado que apesar de terem abordagens diferentes, ambas têm o mesmo objetivo final satisfazer o cliente e entregar o produto de acordo com as necessidades.

No capítulo seguinte serão aprofundadas algumas das metodologias ágeis, nomeadamente Scrum e XP por serem as mais usadas, tal como podemos constatar no capítulo 2. Desta forma, serão descritos os princípios de funcionamento, valores e práticas de diversas metodologias ágeis.

# 4. Metodologias Ágeis

Este capítulo descreve o princípio de funcionamento de cada metodologia ágil, para que se permita a comparação e validação dos seus pontos fortes e fracos.

O presente capítulo encontra-se dividido em oito secções. As primeiras seis são dedicadas individualmente às metodologias identificadas no capítulo 2 como é o caso de Scrum, XP, Kanban, Scrum e XP e Scrumban, ou aquelas que tiveram alguma relevância na literatura estudada, como por exemplo Feature Driven Development. A sétima apresenta o resumo de algumas metodologias e por fim é apresentado o sumário do capítulo.

## 4.1 Scrum

Nos anos 80, Hirotaka Takeuchi e Nonaka Ikujiro desenvolveram uma estratégia flexível para o desenvolvimento de produtos, onde uma equipa trabalha como um só, para atingir um fim. A palavra Scrum teve origem no jogo "*rugby*"[5], relacionada com a regra de reinício do jogo, na qual os jogadores, em conjunto, empurram a equipa adversária para ganharem a posse da bola. Por analogia, quando se inicia um *sprint* no Scrum, a equipa de desenvolvimento, juntamente com a do cliente, têm de trabalhar em conjunto, para atingirem o objetivo (Hayata e Han, 2011), (Schwaber e Sutherland, 2013).

[5] *Rugby* – é um jogo de futebol colectivo com o objectivo de alcançar a linha de fundo do adversário.

O Scrum, representado na Figura 4-1, tal como atualmente é conhecido, foi escrito em 1995 por Ken Schwaver e Jeff Sutherland, sendo um método fácil de estudar (Hayata e Han, 2011). Sumariamente, Scrum é uma *framework* iterativa e incremental para a gestão do projeto. O seu desenvolvimento é realizado em ciclos chamados de *sprints*, uns após outros, sem pausas. A cada *sprint*, a equipa compromete-se com o cliente na implementação de determinados requisitos, os quais não são passiveis de mudanças durante o decorrer do ciclo. Diariamente, a equipa reúne para dar a conhecer o que foi realizado e o que irá fazer para alcançar os objetivos do *sprint*. No final, é realizada a entrega de um produto devidamente testado e demonstra-o ao cliente, obtendo-se o *feedback* que pode ser incorporado na próxima iteração (Lei *et al.*, 2017). O Scrum pode ser resumido na citação seguinte:

> *"A major theme in Scrum is "inspect and adapt." Since development inevitably involves learning, innovation, and surprises, Scrum emphasizes taking a short step of development, inspecting both the resulting product and the efficacy of current practices, and then adapting the product goals and process practices. Repeat forever." (Sutherland, 2012)*

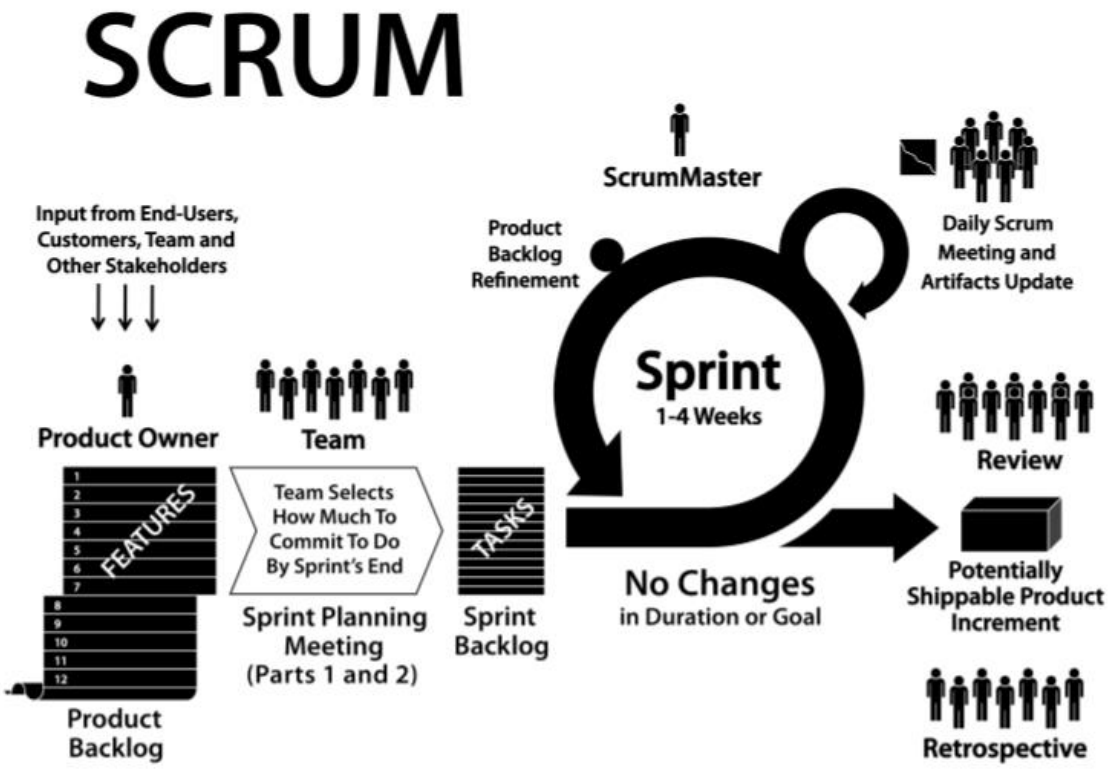


**Figura 4-1: Metodologia Scrum**
**Fonte: (Sutherland, 2012)**

### 4.1.1 Papéis

No Scrum existem três papéis: *Product Owner (*PO), *Team* e *Scrum Master*. As equipas têm todas as competências necessárias para desenvolverem o trabalho de forma autónoma. As equipas são pensadas para serem flexíveis, criativas e produtivas (Schwaber e Sutherland, 2013).

- Product Owner (PO) – Responsável por gerir o *Product Backlog (PO),* cada item deve estar claro para que toda a equipa o perceba. O PO representa os interesses do cliente, logo é ele que toma as decisões (Hayata e Han, 2011).

- *Team* – É formada pelas pessoas que executam o trabalho, para que seja possível uma entrega, a qual incluirá um incremento ao produto no final de cada *sprint.* Estas equipas são autónomas e gerem o seu próprio trabalho. O Scrum não atribui títulos ou funções às pessoas, quer estejam a desempenhar funções de programação, testes ou qualquer outra tarefa. A responsabilidade pertence sempre à equipa que funciona como um todo. As equipas devem ser pequenas o suficiente, mas grandes o bastante para poderem fazer o trabalho com que se comprometem a cada *sprint*. Neste contexto, as equipas devem situar-se entre os três e os nove elementos. O *Scrum Master* não é contabilizado.

- *Scrum Master* – A principal função/papel é garantir que o Scrum é colocado em prática, ou seja, são cumpridas todas as regras do Scrum. Presta auxílio ao PO na gestão do *Backlog,* para além de ser um facilitador dos eventos Scrum. Além disso, tem como função resolver constrangimentos que impeçam o progresso do trabalho.

### 4.1.2 Eventos

O Scrum estabelece alguns eventos que ocorrem com regularidade. Todos devem ocorrer dentro de uma janela temporal pré-definida e com uma duração máxima estipulada. Cada evento significa uma oportunidade para inspecionar e adaptar (Schwaber e Sutherland, 2013).

- *Sprint* - É um elemento vital no Scrum, com duração limitada, podendo prolongar-se até um mês. Durante este período é criado um incremento. Os *sprints* contêm diversos eventos: planeamento do *sprint, Daily Scrum* (reuniões diárias), revisão de *sprint* e a reunião de retrospetiva. Durante um *sprint* não podem ser colocados em segundo plano

os objetivos de qualidade nem serem feitas alterações que comprometam o objetivo. Contudo o PO e a equipa de desenvolvimento podem clarificar e renegociar itens.

- Planeamento do *sprint* - Neste evento é definido o volume de trabalho a realizar num *sprint*, onde todos os elementos da equipa colaboram. A sua duração é no máximo de oito horas, para um *sprint* de um mês. O *Scrum Master* tem de garantir que este evento ocorre e assegura que acontece dentro do limite temporal.

- Reuniões diárias - Ocorrem todos os dias e têm uma duração máxima de quinze minutos. Servem para a equipa de desenvolvimento, partilhar o que tem feito e definir os novos desenvolvimentos. Portanto, todos os elementos respondem a algumas perguntas. O que fiz ontem? O que vou fazer? Que impedimentos tenho?

- Revisão do *sprint* - Esta reunião ocorre no final do *sprint* e tem uma duração máxima de quatro horas para um *sprint* de um mês. Neste evento encontram-se presentes os clientes, o PO, a equipa e o *Scrum Master*, sendo que o PO tem a responsabilidade de apresentar os itens concluídos e aqueles que não foi possível concluir. Mas a apresentação do *sprint* fica a cargo da equipa de desenvolvimento, a qual deve explicar que problemas ocorreram e como foram ultrapassados. Todos os participantes devem colaborar para que sirva de *input* para o próximo *sprint*.

- Retrospetiva do *sprint* - Nesta reunião, a equipa avalia o seu próprio desempenho e identifica aspetos positivos e negativos para ser possível traçar um plano que permita melhorar os diversos aspetos do próximo *sprint*. Tem uma duração máxima de três horas para um *sprint* de um mês. A retrospetiva é a oportunidade formal para se identificarem possíveis melhorias, embora estas possam ser realizadas em qualquer momento.

### 4.1.3 Artefactos

Os artefactos permitem construir uma visão geral do projeto, fornecendo transparência e a possibilidade de inspecionar e adaptar (Schwaber e Sutherland, 2013).

- *Product Backlog* - É uma lista ordenada de todos os requisitos necessários para o projeto. Este artefacto nunca se encontra concluído e evolui de acordo com o produto e o meio envolvente. Pode-se considerar que "está vivo".

- *Sprint Backlog* - É um conjunto de itens do *Product Backlog*, contendo todo o detalhe necessário para que durante a *Daily Scrum* seja possível registar e examinar todo o progresso efetuado. Este deverá refletir o estado real do *sprint*, o que se encontra feito e o que se pretende atingir.

- Incremento - É o somatório de todos os itens do *Product Backlog* que foram concluídos durante todos os *sprints*.

## *4.2 Extreme Programming*

*O Extreme Programming* (XP) é um dos métodos mais populares das práticas ágeis, para desenvolver *software* com padrões de qualidade elevados (Anand e Dinakaran, 2016). Foi criado na década de 90 por Kent Beck e publicado em 1999, no livro "Extreme Programming Explainded" (Beck, 2000), (Beck, 1999).

O XP usa a regra dos 20-80 (80% do benefício vem de 20% de esforço) e tem como principal objetivo criar sistemas de melhor qualidade no menor intervalo de tempo, de forma a reduzir custos. Assim, as equipas são de reduzida dimensão e os seus papéis são bem especificados, visando alcançar a eficiência e efetividade (Beck, 2000). Posteriormente, seguem um conjunto de valores, princípios e práticas básicas adotadas em todo o processo de desenvolvimento, tal como é possível verificar na Figura 4-2 (Semedo e others, 2012).

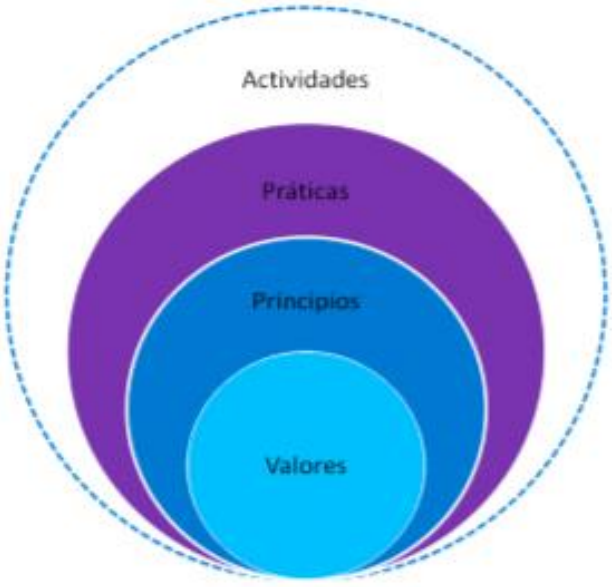


**Figura 4-2: Base XP**
**Fonte: (Semedo, 2012)**

### 4.2.1 Valores

Há cinco valores essenciais em que o XP assenta, tendo em vista a obtenção dos melhores resultados com o projeto de *software*. São eles: a comunicação, simplicidade, *feedback*, respeito e coragem (Anand e Dinakaran, 2016), patentes na citação seguinte.

> *"We will be successful when we have a style that celebrates a consistent set of values that serve both human and commercial needs: communication, simplicity, feedback, and courage"(Beck, 2000)*

- Comunicação - É um aspeto fundamental quando se pretende obter um produto de qualidade, que vá de encontro às necessidades do cliente. Todos os membros da equipa devem comunicar entre si e de forma direta, para que não existam mal entendidos, ou levantadas especulações, ou dúvidas relativamente ao projeto (Santos Soares, dos, 2004). Portanto, quanto melhor for a comunicação, melhor serão as soluções implementadas, pois as contribuições de cada membro serão aproveitadas, fazendo com que críticas se tornem mais-valias (Beck, 2000).

- Simplicidade - Devemos sempre colocar a seguinte questão. Qual é a forma mais simples para funcionar? Ou seja, qual é a coisa mais simples que pode funcionar? E muitas vezes não é fácil obter respostas, pois estamos habituados a olhar para o amanhã e ao raciocinar desta forma estamos a aumentar o custo da mudança. A simplicidade ajuda a aumentar a confiança do trabalho que estamos a realizar (Beck, 2000).

- *Feedback* - Saber a opinião real do cliente sobre um *software* é absolutamente fundamental, uma vez que estamos a colmatar um dos maiores riscos que é o otimismo. O *feedback* pode ser obtido quase instantaneamente ou após meses. Exemplo disso é o retorno dos utilizadores finais que só têm contacto com a aplicação depois das equipas de desenvolvimento e qualidade terem terminado (Beck, 2000).

- Coragem - É um dos valores mais importantes, pois é necessária coragem para implementar os outros valores, para pedir ajuda e/ou perder todo o trabalho executado durante um dia. É necessário simplificar e comunicar o atraso e ou o não cumprimento de um prazo estabelecido com o cliente. Coragem é ter a capacidade de poder esquecer métricas para se resolver um problema de raiz, mesmo que isso implique um esforço adicional na equipa (Beck, 2000).

- Respeito - Saber respeitar o outro é fundamental para que todos os valores do XP funcionem. Sem respeito não há comunicação e sem comunicação irá falhar o *feedback*. Caso não haja respeito, a coragem pode desaparecer. Faltar ao respeito é estarmos a destruir a nossa equipa e não comprometidos com ela. É necessário confiar que cada um fará o seu melhor. Este valor só apareceu na segunda edição do livro de Beck (Beck, 2000).

### 4.2.2 Princípios

Na segunda edição do livro de Beck foram adicionados alguns princípios na metodologia, ajudando os valores a serem transformados em práticas (Beck, 2000).

- Humanidade - Todo o desenvolvimento é feito por pessoas que necessitam de ser respeitadas e tidas em conta, balanceando com as necessidades da empresa e da equipa.

- Economia - Toda a equipa deve ser conhecedora das reais necessidades da área de negócio e conseguir, num menor intervalo de tempo possível entregar as áreas fundamentais do projeto. A equipa deve ter capacidade para reagir rapidamente a mudanças na área de negócio.

- Benefício mútuo - Tudo o que é feito deve beneficiar todos os interessados. Por exemplo, os testes trazem benefícios imediatos, pois ganha-se confiança no *software*. Um exemplo de uma prática que viola este princípio é a extensa documentação de *software*, a qual pode representar um incerto benefício futuro a alguém, mas é uma incógnita. No entanto, no imediato, irá retirar tempo ao programador.

- Semelhanças - As boas soluções aplicadas em projetos anteriores podem ser usadas novamente, mesmo que o contexto do projeto seja diferente. O objetivo é identificar estruturas e padrões entre projetos.

- Melhoria - Não existe nada perfeito no desenvolvimento de *software*, existindo sempre espaço para melhorar.

- Diversidade - A equipa deve conter elementos com diversas habilidades e opiniões, beneficiando todos desta diversidade. Contudo, é necessário existir consideração entre

todos, pois assim se garante que as diferenças sejam sinónimo de crescimento para os elementos e contribuam para a melhoria do projeto.

- Reflexão - É preciso ser autocrítico e aprender com o passado, identificar pontos de melhoria e partilhar experiências de sucesso, para que possam ser repetidas.

- Fluxo - O ritmo deve ser constante. A equipa deve realizadas entregas constantes e em curtos períodos de tempo, minimizando o risco de possíveis erros e do custo de corrigi-los.

- Oportunidade - A equipa deve estar sempre atenta às oportunidades que surgem. As mudanças inesperadas, surpresas e problemas devem ser encarados como uma oportunidade de melhoria.

- Redundância - Devemos reduzir a possibilidade de existirem erros e assegurar a qualidade através de testes automatizados e *pair programming*.

- Falha - A equipa deve ter coragem para tentar mudar, mesmo sabendo que por vezes irão falhar. Todavia, deve ser sempre tirada uma lição da experiência para investir numa melhor opção.

- Qualidade - A qualidade não se negoceia, é uma constante para a equipa. Uma maior qualidade traz motivação e satisfação para todos.

- Passos pequenos - As histórias devem ser repartidas em pequenos passos para que a equipa possa controlar melhor o que se encontra a desenvolver e seja mais fácil de testar. As entregas devem ser executadas em intervalos reduzidos, de modo a minorar o risco de aceitação.

- Aceitar responsabilidades - Todos os elementos devem estar comprometidos com as suas responsabilidades. Cada um deve responsabilizar-se pelas estimativas, pela implementação e pelos testes que assumem.

### 4.2.3 Práticas

A maioria das práticas do XP, tal como podemos observar na Figura 4-3, podem ser polémicas à primeira vista, como *pair programming*, que é visto muitas vezes como uma despesa. Além

disso, não podem ser usadas isoladamente, pois é a sinergia delas que sustenta a metodologia do XP (Beck, 2000). De seguida, serão apresentadas as práticas do XP, segundo Beck, Sommerville e Jeffries (Beck, 2000), (Sommerville, 2007), (Jeffries, 2011).

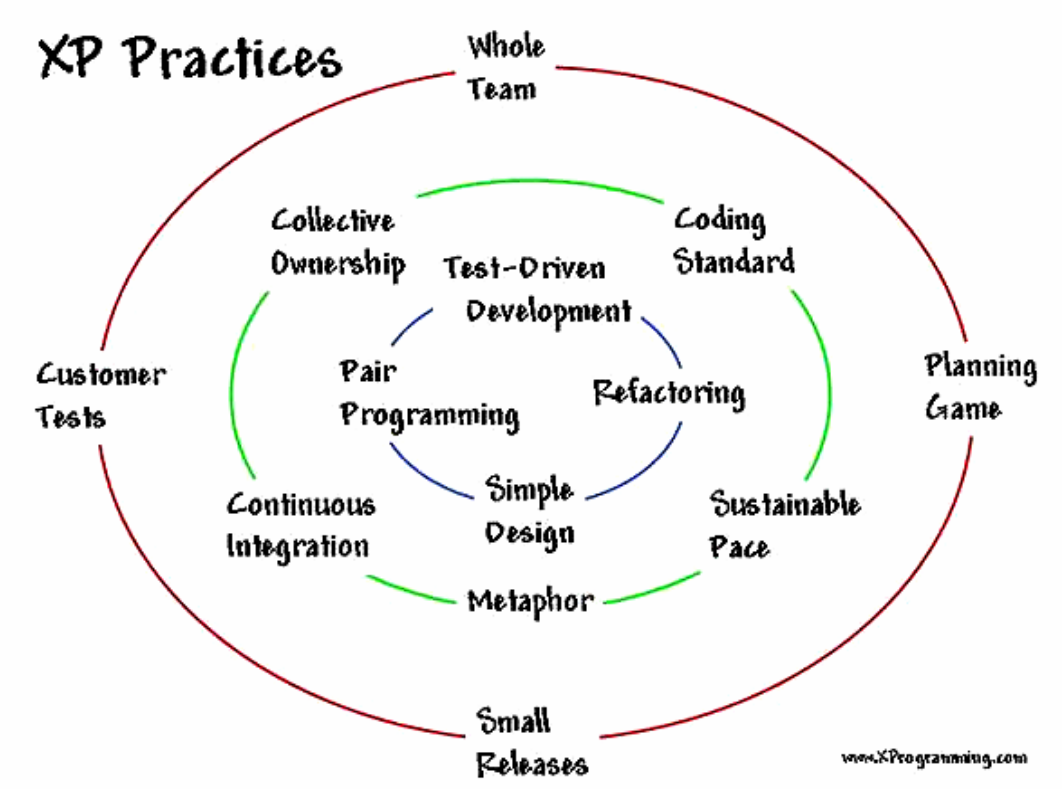


**Figura 4-3: Práticas do XP**
**Fonte: (Jeffries, 2011)**

- Jogo do planeamento – Esta prática implica uma grande interação entre cliente e equipa de programadores, uma vez que a equipa indica o tempo necessário para implementar cada história e o cliente prioriza cada item a ser realizado. Neste jogo são planeadas as iterações que duram entre uma a quatro semanas (deve ser entregue um *software* executável). As iterações incluem histórias que são apresentadas pelo cliente. Os programadores dividem-nas em tarefas mais pequenas, até alcançarem um nível de detalhe adequado. São também planeadas as entregas futuras, onde as histórias são planeadas num nível de detalhe mais geral;

- Entregas frequentes – Cada iteração conclui-se com uma entrega que contém as histórias acordadas no planeamento. Após a entrega, o cliente irá proceder à validação dos seus testes para que no final possa indicar o seu *feedback* à equipa de desenvolvimento. O

objetivo desta prática é o *software* ser monitorizado, tanto pela equipa como pelo cliente;

- Metáfora – A metáfora ajuda toda a equipa a estar em sintonia com o projeto. Para tal, deve conseguir transmitir ideias complexas de forma simples e clara. Beck (Beck, 2000) afirma, igualmente, que o vocabulário utilizado deve ser o que o cliente usa no seu dia-a-dia;

- Projeto simples – O projeto deve ser o mais simples possível, para permanecer flexível e ágil. Para tal, é necessário existirem revisões do que se encontra a ser feito, já que o projeto irá crescer;

- Testes – Os programadores programam e executam testes unitários[6], enquanto o cliente é a pessoa responsável pelos testes funcionais. Os testes unitários têm como função ajudar os programadores a validar o seu código. Para cada tarefa, é escrito um mínimo de testes de aceitação na qual se responde à questão "o que deve ser verificado para estar concluído?". O resultado dos testes deve ser público para toda a equipa;

- *Refactoring* – No processo de manter a simplicidade e a melhoria contínua do projeto, é utilizada a técnica de *refactoring*, cujo objetivo é tornar o código reutilizável e compreensível, sem que o comportamento da aplicação mude de comportamento;

- *Pair programming* – Em XP, todo o código deve ser produzido por dois programadores, assegurando que tudo é revisto pelo menos por um programador. As consequências desta prática é um aumento da velocidade de desenvolvimento, redução do número de *bugs* e otimização de código. Cada um dos elementos do par de programadores têm papéis distintos. Enquanto um se encontra a escrever o código da melhor forma possível o outro encontra-se a analisar a estratégia usada, de forma a detetarem se existe algum caso de teste não planeado e como simplificar o código;

- Propriedade coletiva – Qualquer pessoa, em qualquer momento, pode fazer uma alteração que traga valor. Todos são responsáveis por todo o processo. Este facto não

[6] Testes unitários – Estes testes são efetuados pelos programadores e consistem em testar isoladamente uma parte da aplicação.

implica que todos devem conhecer tudo a 100%, mas todos devem saber o mínimo do que faz cada parte.

- Integração contínua – As equipas de XP encontram-se em constante integração do código desenvolvido por todos. No pior cenário, o código é integrado com um dia de atraso. Para a implementação desta prática é fundamental usar-se um sistema de controlo de versões;

- Semana de quarenta horas – As horas extra não são recomendáveis. É essencial existir descanso para a produtividade continuar em alta. Um programador cansado verá diminuída a sua capacidade de raciocínio, e por consequência, aumentar o número de erros. Segundo Beck (Beck, 2000), a sobrecarga de trabalho é um sintoma de que o projeto se encontra com problemas;

- Cliente presente – O cliente é um elemento integrante da equipa. É primordial estar sempre disponível, para dar o seu *feedback* sobre dúvidas que possam existir ou alterações a entregar, de forma a agilizar todo o processo. Na maioria das vezes, não é possível às equipas de desenvolvimento estarem fisicamente juntas com o cliente e como tal é fundamental existir um canal de comunicação sempre aberto;

- Padrões de desenvolvimento – Todos os elementos seguem o mesmo padrão de codificação, para que seja exequível entender e alterar o código desenvolvido por outros membros do projeto. Não é importante a utilização de um padrão em particular. O relevante é que exista um e que a equipa o assuma voluntariamente como seu.

### 4.2.4 Papéis

Nesta metodologia existem diferentes papéis a desempenhar ao longo do projeto. Mas nem todos necessitam de coexistir ou serem atribuídos a pessoas diferentes. De acordo com Beck (Beck, 2000), os papéis são:

- Programador – É responsável por escrever o código do programa, estimar as histórias e desenhar os testes. Deve manter sempre o programa o mais simples possível;

- *Coach* – É o papel desempenhado por um programador sénior. É responsável por assegurar que todos os elementos da equipa de programação realizam as práticas

propostas pela metodologia. É a consciência da equipa (Beck, 2000). Este papel é facultativo e pode até rodar entre os membros da equipa;

- *Tracker* – É o programador responsável por informar qual o ponto de situação do projeto, recorrendo a gráficos colocados na sala de trabalho, por exemplo. Tem ainda como função selecionar o que melhor ajudará a equipa (Jeffries, Anderson e Hendrickson, 2001);

- Cliente – É responsável por escrever as histórias e os testes que irão permitir a aceitação da história. Além disso, o cliente define a prioridade da implementação e indica quando um requisito se encontra implementado. Ainda que não participe diretamente na equipa de programação, deve estar disponível para esclarecer dúvidas, e fornecer o *feedback* sobre os desenvolvimentos e testes a realizar pela equipa de programação ou outras ocorrências. Na ausência do cliente, o *Coach* pode representá-lo e assumir o seu papel (Wallace, Bailey e Ashworth, 2002);

- Qualidade – É o responsável pela execução dos testes a todo o sistema de forma regular, assim como transmitir o resultado dos testes à equipa de programação, recolhendo as suas evidências, guardando-os num formato a definir pela equipa. Pode aliás, ajudar o cliente a escrever os testes funcionais;

- Consultor – É um elemento externo ao projeto, mas com conhecimento específico numa área de negócio ou tecnológica, o qual pode ser chamado a ajudar a equipa de programação. Frequentemente, após indicar uma possível solução para o problema, a equipa de programação volta a analisar e refaz, mas tal não deve ser interpretado como ofensa ao consultor, uma vez que se trata de uma prática comum do XP;

- Chefe – É o responsável pela tomada de decisões. Encontra-se próximo da equipa de desenvolvimento e existe uma troca constante de informação. A equipa alerta para qualquer problema ou dificuldade, para que a solução seja encontrada o mais rapidamente possível.

### 4.2.5 Funcionamento

Os programadores de XP elegem normalmente, projetos com uma fase de desenvolvimento curta e, em seguida, um longo período de produção e otimização, uma vez que o método XP

tem uma preocupação com a interação entre o cliente e a equipa de desenvolvimento, como pode ser verificado na citação de Kent Beck, 2012, p. 100. Na Figura 4-4, podemos examinar as fases do ciclo de vida do XP (Beck, 2000).

> *"The ideal XP project goes through a short initial development phase, followed by years of simultaneous production support and refinement, and finally graceful retirement when the project no longer makes sense." (Beck, 2000)*

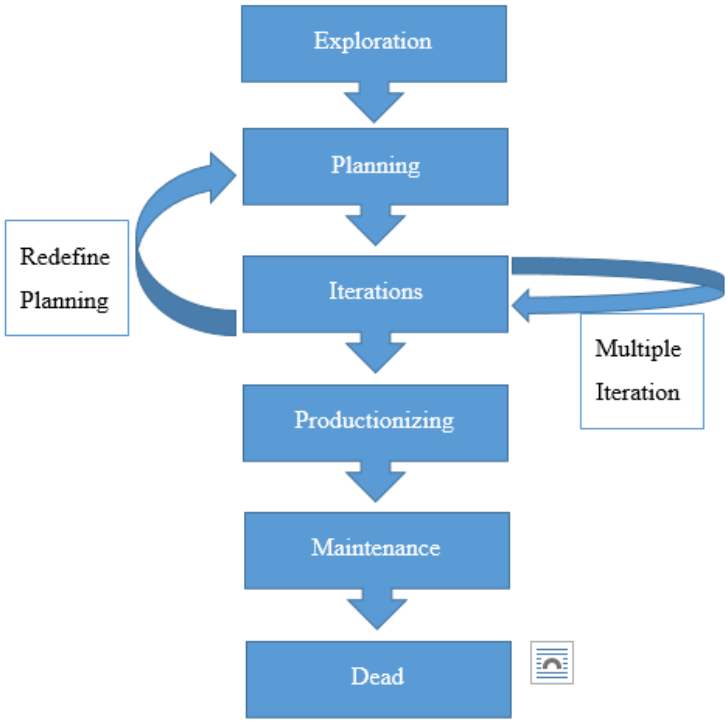


**Figura 4-4: Ciclo de vida XP**
**Fonte: Adaptado de Extreme Programming Explained (Beck e Andres, 2004)**

Observando a Figura 4-4, pode constatar-se que um projeto desenvolvido com a metodologia XP é iniciado com a fase de exploração, que tanto pode ser curta, com a duração de alguns dias, como pode durar algumas semanas. Neste momento, a equipa de programação reúne-se com o cliente para perceber as suas reais necessidades e intenções e, dessa forma, adquirir um conhecimento mais amplo de todo o produto a desenvolver. Nesta fase, a equipa realiza diversos esboços e pequenas aplicações, chamadas de "*spikes*", para poder avaliar qual o melhor caminho.

Quando se avança para o planeamento, o cliente escreve as histórias e existe interação com a equipa de desenvolvimento, para serem esclarecidas dúvidas e/ou melhorias. Como indicado

anteriormente, este ponto faz parte das práticas do XP e ocorre diversas vezes durante o ciclo de vida do projeto.

Durante a fase de implementação, ocorrem diversas iterações que terminam com entregas e dão início a um novo planeamento. No final de cada iteração, o cliente realiza os testes de aceitação para fechar as histórias ou então regista alterações que serão incluídas no novo planeamento. Nas iterações podem ser feitas reuniões diárias, nas quais todos partilham o trabalho presente e futuro, para que toda a equipa tenha noção de todo o projeto e o seu estado. Beck sugere que estas reuniões sejam feitas de pé, de forma agilizar o projeto (Beck, 2000).

## 4.3 Kanban

O método Kanban surge no Japão na década de 1950, mais propriamente na Toyota, que desenvolveu um sistema de controlo de processo de fabrico com recurso a cartões (Teixeira, 2013). O desenvolvimento de *software* surgiu por David J. Anderson (Anderson, 2010) que usou esta metodologia para o ajudar na resolução de problemas. Os resultados que obteve foram expostos nas conferências "Lean New Product Development" e "Agile 2007" (Anderson, 2010) e, desde então, o Kanban tem vindo a crescer (Raju e Krishnegowda, 2013).

Aquando do aparecimento do Kanban, surgiu a filosofia *"Just in time* - JIT" que o ajudou a suportar. JIT determina tudo o que deve ser produzido sem gerar stocks, de forma a atender os pedidos dos clientes, garantindo que só é produzido o que realmente é necessário (Anderson, 2010), (Marek, Elkins e Smith, 2001).

Segundo Reinersten (Reinertsen e Bellinson, 2014), este método tem como objetivo impulsionar as equipas a representar graficamente o fluxo de trabalho. A Figura 4-5 mostra o exemplo de um quadro com o trabalho de cada sujeito e contém ainda o conceito de "Work in Progress – WIP", que compreende delimitar o número máximo de itens em cada fase do quadro (Reinertsen e Bellinson, 2014).


**Figura 4-5: Quadro Kanban**
**Fonte: (10 Kanban Board Examples | LeanKit, 2017)**

Alguns autores têm especificado os princípios e práticas da metodologia Kanban, nomeadamente Anderson (Anderson, 2010), Gross (Gross e McInnis, 2003) e Boeg (Boeg, 2010), sendo que a deste último é a mais detalhada. Na Figura 4-6 podem examinar-se os princípios e práticas.

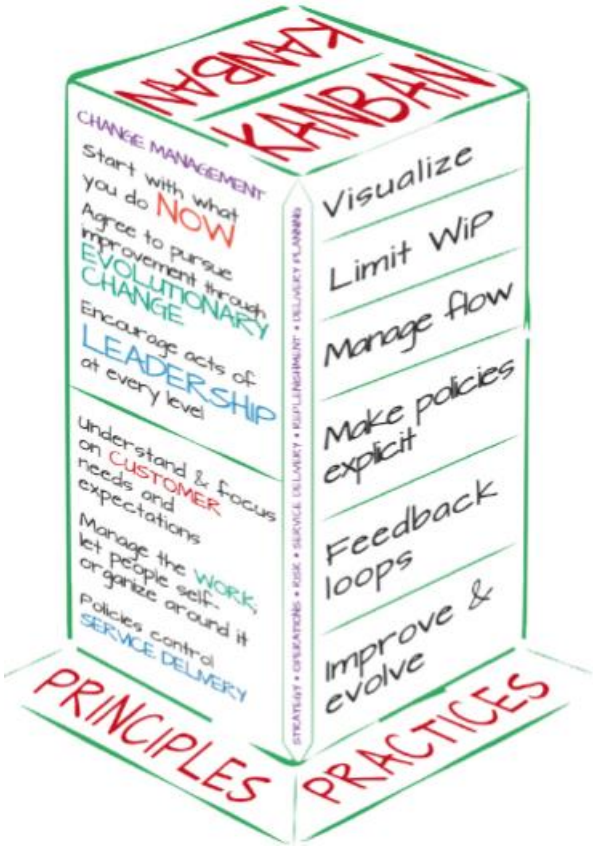


**Figura 4-6: Princípios e práticas Kanban**
**Fonte: (*What is the Kanban Method? | Lean Kanban*, 2017)**

Princípios e práticas do Kanban:

- Começar com o que existe - O método Kanban não obriga à implementação de um conjunto de processos, etapas ou funções. É iniciado com os processos já existentes na empresa e estimula a mudança contínua, incremental e evolutiva do sistema (Anand e Dinakaran, 2016);

- Concordar com a mudança contínua - A organização tem de concordar com a mudança contínua e incremental;

- Respeitar o processo atual - O que funciona bem deve-se manter. Todavia, devem-se procurar os pontos fracos para serem retirados do método de trabalho e assim facilitar a mudança. O respeitar o que se encontra a correr bem, como papéis e responsabilidades no local de trabalho, gera um maior consenso no apoio às mudanças necessárias;

- Liderança - São incentivados atos de liderança em todos os níveis da organização, desde os papéis mais simples ao papel de gestores, uma vez que as melhorias são esperadas em qualquer nível de uma organização;

- Ver o trabalho – Ao ser criado um modelo visual de todo o trabalho e seu fluxo[7] (divisão do quadro em colunas que representam as diferentes fases de um pedido), podemos observar o fluxo em movimento através do sistema Kanban. Tornar o trabalho visível junto com todos os constrangimentos, instantaneamente, contribui a uma maior compreensão e colaboração;

- *Work-in-Progress* (WiP) – Ao ser limitado o número máximo de pedidos por estado do fluxo, reduz-se o tempo que um item demora a percorrê-lo e auxilia também na priorização dos pedidos, evitando problemas de trocas;

- Gerir o fluxo – É necessário monitorizar como os pedidos percorrem o fluxo. Deve ser rápido e suave, ou seja, criar rapidamente valor para o negócio, minimizar o risco e custos de atraso;

[7] É a sequência de passos para automatizar processos de negócio, de acordo com um conjunto de normas.

- Regras bem definidas – Todas as normas de trabalho devem ser bem compreendidas no seio da equipa, caso contrário é impossível haver entendimentos ou processos de melhoria.

## 4.4 Scrumban

Scrumban é uma metodologia híbrida, que contém processos de Scrum e Kanban e foi desenhada para ajudar a lidar com as constantes mudanças de requisitos por parte dos clientes (Yilmaz e O'Connor, 2016). O termo Scrumban surge em 2008 numa publicação de Ladas "Scrumban-Essays on Kanban Systems for Lean Software Development", na qual se exprime a exigência de compromissos técnicos e metodológicos entre as metodologias de origem, como podemos observar na Figura 4-7 (Stoica *et al.*, 2016).

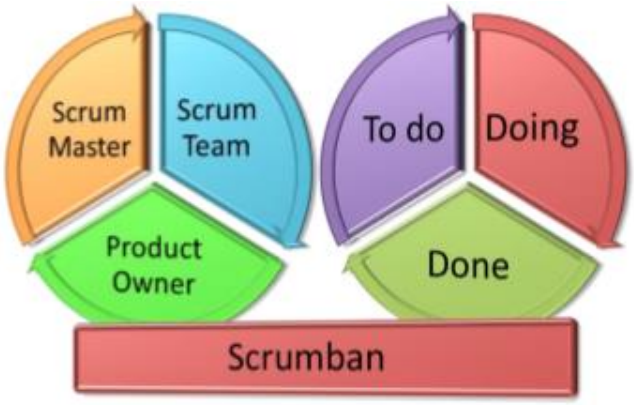


**Figura 4-7: Scrum+ Kanban**
**Fonte: (Stoica, 2016)**

Se o Scrum é mais indicado para projetos de desenvolvimento e o Kanban para projetos de manutenção. A combinação de ambos tornou-se popular nos dias de hoje, já que as empresas de serviços dispõem de projetos de desenvolvimento e manutenção (Pahuja, 2012).

Do Scrum, foram usadas as melhores práticas, tais como reuniões diárias, histórias de clientes e a organização da equipa. Não obstante, o quadro do Scrum não reflete toda a realidade do projeto e neste ponto passou a ser usado o quadro do Kanban com o mecanismo de WIP.

O Scrum obriga a que em cada iteração seja feito todo o planeamento das histórias que irão constar do *sprint*. Portanto, tendo todos os itens visíveis, é reduzido o desperdício de tempo na

discussão de planeamento, obrigando a que o quadro seja bom para ajudar as equipas a melhorar a sua produtividade.

O Scrumban tem como práticas:

- Visualizar o fluxo de trabalho - A equipa examina todo o trabalho, desde as histórias que ainda só foram pedidas pelo cliente até às entregues. Estas práticas ajudam a equipa a saber quem se encontra a trabalhar em cada tarefa e qual o seu estado de progresso. O quadro é dividido por diversas colunas, representando estados, os quais possibilitam à equipa efetuar o acompanhamento e mover as histórias em torno dessas colunas. Uma representação típica desse fluxo pode ser vista na Figura 4-8.


**Figura 4-8: Quadro Scrumban**
**Fonte: (Pahuja, 2012)**

- *Pull work* - O trabalho é executado conforme as necessidades do cliente e por ordem de prioridade. Quando terminam uma tarefa, os elementos da equipa retiram da primeira coluna um novo pedido, sem necessidade de pedirem ao líder da equipa a atribuição de trabalho, contribuindo para que o fluxo seja com transições suaves.

- *Limit WIP* - Cada elemento deve trabalhar numa tarefa de cada vez. Para garantir esta prática, o Scrumban coloca o número máximo de itens que pode estar em cada coluna. Geralmente, corresponde ao número de pessoas da equipa.

## 4.5 Scrum & XP

O XP com Scrum é mais um método híbrido que reúne práticas das duas metodologias ágeis e pretende unir as práticas de gestão de *software* do Scrum com as práticas de engenharia do método XP.

Na literatura pesquisada, entre Dezembro de 2016 e Março de 2017, não foi possível identificar muitas referências de Scrum & XP. Uma das justificações encontradas é que as equipas começam por usar Scrum e acabam por adotar a sua própria versão de XP "start with Scrum and then invent your own version of XP." (Cohn, 2005). Porém, alguns autores como Henrik Kniberg (Kniberg, 2015), Mushtaq e Qureshi (Mushtaq e Qureshi, 2012), Christ Vriens (Vriens, 2003), têm apontado diretrizes para a utilização das duas metodologias.

Neste contexto, analisa-se a proposta de Mushtaq e Qureshi em 2012, apresentada na Figura 4-9, onde se pretendem ligar as boas práticas de ambos os métodos, aumentando o desempenho das equipas e melhorando a qualidade do *software*, de acordo com os seus requisitos (Mushtaq e Qureshi, 2012).

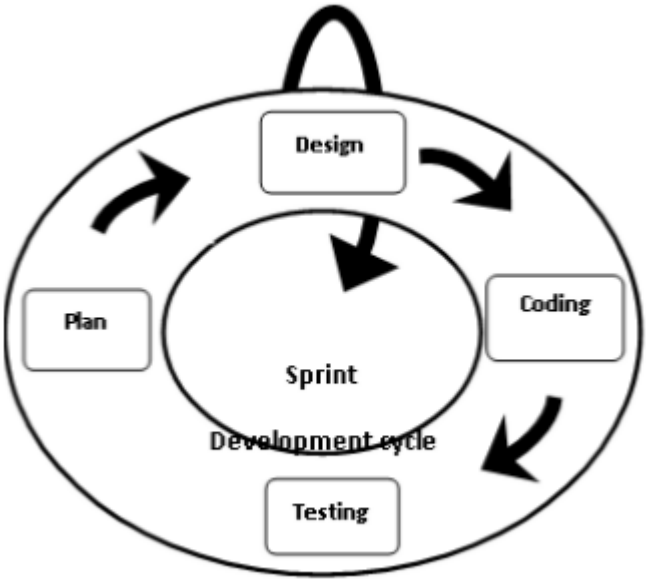


**Figura 4-9: Ciclo Scrum&XP**
**Fonte: (Mushtaq e Qureshi, 2012)**

Este modelo híbrido inicia com um *sprint* zero onde é analisado o produto, o qual é dividido em subprocessos que envolvem estimativas e priorizações. Os outros *sprints* são em ciclo, até terminar todas as entregas. Estes *sprints* contêm planeamento, desenvolvimento, testes, e por último, integração. Este modelo necessita que o cliente esteja disponível durante todo o ciclo, para poder acompanhar e dar o seu *feedback*.

Na fase de planeamento de cada ciclo, são definidas todas as tarefas que a equipa de desenvolvimento irá entregar. Esta fase é muito idêntica ao que se utiliza no Scrum, em cada planeamento de *sprint*.

Na fase de desenho, a equipa encontra-se focada nos itens do *sprint* e deve seguir o princípio do XP da simplicidade. Além disso, é nesta fase que são concebidos os casos de testes e diagramas de classes e objetos.

Na fase de desenvolvimento de código, a equipa deve seguir algumas práticas do XP como *pair programming*, testes e *refactoring*. Tal como no XP, todos são responsáveis pelo código e todos podem alterá-lo.

A fase de testes encontra-se sempre presente, uma vez que começa antes de se iniciar o desenvolvimento de qualquer item. Continua no desenvolvimento com testes unitários e, no final, tudo é testado.

## 4.6 Outsystems

Ainda que na literatura não se encontrem muitas referências relativamente à metodologia Outsystems, o autor opta por a referir, uma vez que já trabalhou com ela em diversos projetos. Apesar de Outsystems ser uma plataforma de desenvolvimento, possui a sua própria metodologia ágil que se baseia muito em Scrum, mas adaptada. Esta metodologia surge devido às metodologias tradicionais não se adaptarem às constantes mudanças de requisitos pedidas pelos clientes e do facto de 53% dos projetos no mundo ultrapassarem o seu custo e tempo (OutSystems, 2010), (Ferreira, 2013).

Tipicamente, Outsystems sugere o modo como os seus projetos devem ser divididos em *sprints*, sugerindo a duração de cada *sprint* entre duas a quatro semanas. Podemos dividir os *sprints* em três grupos, como se pode averiguar na Figura 4-10.

- Análise de requisitos, muito mais reduzida que nas metodologias tradicionais;
- Os *sprints* intermédios - em cada um é feita análise, desenvolvimento e testes. Termina com uma demonstração aos clientes, para obter *feedback*;
- Os últimos dois *sprints* correspondem à formação e estabilização.

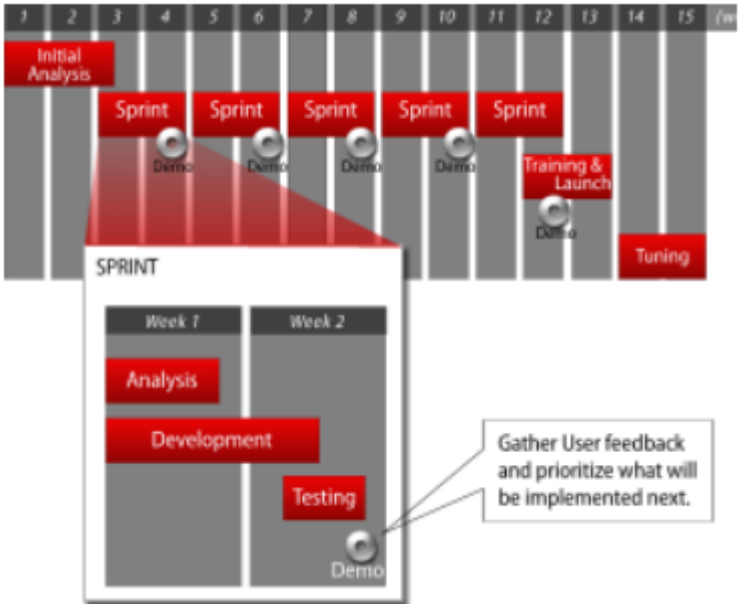


**Figura 4-10: Ciclo Outsystems**
**Fonte: (OutSystems, 2010)**

Deste método iterativo destacam-se algumas práticas que levam a que o resultado do projeto seja o mais próximo das necessidades do cliente. Entre elas, encontra-se a *Demo* e o *Feedback*, as quais se encontram muito interligadas, uma vez que é durante a reunião que o cliente indica se o que foi feito no *sprint* vai de encontro com as suas expetativas e encontra-se de acordo com as regras de negócio. Compete à equipa recolher no seguinte *sprint* este *feedback* para aperfeiçoar o que não estiver correto. No fim de cada *sprint*, é necessário estabelecer prioridades na execução das tarefas com base nas necessidades do cliente, assim como indicar o esforço necessário para tal (Ferreira, 2013).

Em Outsystems, todas as tarefas a fazer que derivam de histórias do cliente encontram-se contidas numa lista de tarefas chamada de *Project Backlog*. Esta lista é dinâmica e encontra-se ordenada por ordem de prioridade do cliente.

Nesta metodologia existem duas variáveis que nunca podem ser alteradas - o esforço previsto para o projeto, expresso em homem/dia, denominado de *Budget*, para que os valores se mantenham os mesmos de início ao fim do projeto. A outra variável é o tempo definido para a duração do projeto, o qual dependerá integralmente do *Budget* (OutSystems, 2010).

Existem ainda alguns papéis fundamentais para o sucesso dos projetos, podendo ser divididos em duas equipas: clientes e desenvolvimento (Ferreira, 2013):

- Cliente
    - *Business Sponsor* – Tipicamente, a pessoa que realiza a requisição do projeto e que deve delegar responsabilidades, representantes e a sua disponibilidade;
    - *Business Manager* – Pessoa responsável por esclarecer dúvidas relacionadas com os requisitos de negócio, garantir a participação dos elementos necessários nas diversas fases do projeto e por coordenar a comunicação entre todos os *stakeholders*;
    - *IT Manager* – Responsável pela infraestrutura necessária ao projeto. Deve garantir todo o suporte, tal como as infraestruturas que a equipa de desenvolvimento necessite;
    - *Business User* – Utilizador final, ou seja, o utilizador que irá usar a aplicação no seu quotidiano. Deve ajudar a esclarecer dúvidas relacionadas com os requisitos de negócio, dar o seu *feedback* sobre cada demonstração e fazer testes de aceitação a cada *sprint*;
    - *Maintenance* – Responsável pela operação, manutenção e após passagem a produção da solução. Deve recolher todo o conhecimento possível sobre a aplicação para apoio na manutenção. Para tal, deve trabalhar, sempre que possível, com a equipa de desenvolvimento durante a implementação.
- Desenvolvimento
    - *Engagement Manager* (EM) – Pessoa responsável pelo estabelecimento da ponte entre a equipa de desenvolvimento e o cliente. Deve gerir todo o planeamento do projeto, promover o relacionamento entre todas as partes, analisar a área de negócio e fazer relatórios de gestão, nomeadamente do progresso do projeto;
    - *Delivery Manager* (DM) - Responsável pelo desenvolvimento do projeto, arquitetura e gestão da equipa de desenvolvimento.
    - *Developer* – Elemento da equipa de desenvolvimento. Deve implementar o trabalho dado pelo DM e realizar testes unitários.

## 4.7 Outras Metodologias Ágeis

No decorrer desta dissertação foram estudadas outras metodologias ágeis, uma vez que atualmente ainda são usadas, como se pode constatar no capíulo 2. De seguida é apresentado um resumo de cada uma, podendo no anexo 2 serem consultadas com mais pormenor.

- Adpative Software Development (ASD) – Foi proposta por Jym Highsmith (Highsmith, 2013), e focou-se na colaboração humana e na auto-organização. Este método adapta-se a ambientes onde existem mudanças constantes e pouco planeamento. Por isso, o ciclo de desenvolvimento assenta em três fases, sendo elas especulação, colaboração e aprendizagem;

- Crystal – É uma família de metodologias de desenvolvimento, que foi criada por Alistair Cockburn e Jim Highsmith. Os elementos da família são Crystal Clear, Crystal Yellow, Crystal Orange, Crystal Red, Crystal Marron, e para escolher qual utilizar, usam-se dois parâmetros, sendo o primeiro o número de elementos envolvidos e o segundo o nível crítico do projeto (Highsmith e Cockburn, 2001);

- Dynamic Systems Development Method (DSDM) – Esta metodologia é gerida por um grupo de empresas mundiais que definem todo o seu ciclo, composto por três fases: Pré-projeto, Projeto e Pós-projeto. O DSDM segue o princípio que 80% da aplicação deve ser entregue em 20% do tempo (Pressman, 2005), (*AgileBusiness*, 2017);

- Feature Driven Development (FDD) – Foi criada inicialmente por Jeff De Luca e por Perter Coad (Coad, Luca e Lefebvre, 1999) para efetuarem um projeto de desenvolvimento de 15 meses e 50 pessoas. O FDD é um processo de iterações curtas e o seu ciclo é composto por cinco atividades: Desenvolver um modelo abrangente; Construir a lista de funcionalidades; Planear as funcionalidades; Detalhar as funcionalidades e Desenvolver as funcionalidades (Coad, Luca e Lefebvre, 1999), (Pressman, 2005), (Rico, Sayani e Sone, 2009).

## 4.8 Sumário

Neste capítulo foi efetuada a descrição das diversas metodologias. Na sua maioria, encontram-se mais direcionadas para a gestão de projetos como o Scrum, Kanban, Outsystems, enquanto

outras, como o XP, direciona-se para o funcionamento da equipa e práticas a implementar no processo de desenvolvimento de um projeto.

No próximo capítulo será realizada uma comparação entre as metodologias apresentadas, para se entender quais as suas vantagens e os seus pontos fracos. Além disso, serão também efetuadas comparações com o PMBOK.

# 5. Análise entre Metodologias de Gestão de Projetos de Desenvolvimento de *Software*

Este capítulo irá comparar as diferentes metodologias ágeis, apresentadas anteriormente, assim como, com o *standard* (PMBOK) escolhido no capítulo 3.

Inicialmente, atendendo à literatura, será realizada uma análise comparativa entre as metodologias. Seguidamente, será efetuada uma análise entre métodos tradicionais e ágeis. Após esta análise, serão comparadas as metodologias ágeis, nomeadamente entre XP e Scrum, pois além de serem as mais usadas, identicamente existe uma metodologia híbrida que envolve os dois – Scrum & XP. Por último, será realizada uma análise comparativa entre o Scrum com o PMBOK.

## 5.1 Como Comparar Metodologias

A comparação de diferentes metodologias baseia-se, geralmente, na opinião dos autores que as estudam, as quais resultam da influência obtida de experiências na aplicação de metodologias a projetos.

O PMI define projeto como único e ocorre em circunstâncias singulares, podendo-se concluir que não é fácil o processo de comparação, pois não existem dois cenários iguais (Almeida, 2008).

Para se realizar uma comparação dos métodos ágeis, é necessário recorrer de metodologias adequadas. O estudo de Pekka Abrahamsson (Abrahamsson *et al.*, 2003) apresenta cinco abordagens de análise:

- Ciclo de vida de *software* – analisa as diferentes fases;
- Gestão de projetos – verifica se o método apoia as atividades de gestão de projetos;
- Princípios abstratos vs diretrizes concretas - valida se o método se baseia em métodos abstratos ou se possui diretrizes;
- Universalmente pré-definido vs situação apropriada – verifica se o método se adapta a todas as situações ou se o método é ajustável, dependendo da situação;
- Evidência empírica – O método tem suporte empírico, ou seja baseia-se nas experiências vividas, não tendo em conta métodos científicos.

Os autores João Freitas (Freitas, 2015) e Mauro Almeida (Almeida, 2008) nas, suas respetivas dissertações de mestrado, usaram o método quase informal de Henk G. Sol (Sol, 1983), onde identificaram um conjunto de atributos significativos e utilizaram-nos para efetuarem a comparação, embora dependa sempre da subjetividade do autor.

Segundo Song e Osterweil (Song e Osterweil, 1994), para levarmos a efeito uma comparação devemos encontrar um conjunto de caraterísticas fundamentais, descrevendo-as num vocabulário que permita a comparação, ou seja criar um formalismo.

## 5.2 Abordagem Tradicional *Versus* Ágil

O relatório de 2015 do Chaos Report, lançado pelo Standish Group (Hastie e Wojewoda, 2015), realizou um estudo em mais de 50 mil projetos, de onde é possível concluir que existe uma maior taxa de sucesso em projetos que adotaram metodologias ágeis. Até para os projetos de grandes dimensões, as metodologias ágeis conseguem obter melhores resultados, contrariando

assim uma das desvantagens que lhes é atribuída. Na Tabela 5-1 podemos ver em detalhe os resultados (Hastie e Wojewoda, 2015).

**Tabela 5-1: Taxa de sucesso entre metodologias ágeis e tradicionais**
**Fonte: (Hastie e Wojewoda, 2015)**

| | Abordagem | Successful | Challenged | Fail |
|---|---|---|---|---|
| Todos | Agile | 39% | 52 % | 9% |
| | Traditional | 11% | 60% | 29% |
| | | | | |
| Projetos Grandes | Agile | 18% | 59% | 23% |
| | Traditional | 3% | 55% | 42% |
| Projetos Médios | Agile | 27% | 62% | 11% |
| | Traditional | 7% | 68% | 25% |
| Projetos Pequenos | Agile | 58% | 38% | 4% |
| | Traditional | 44% | 45% | 11% |

Na Tabela 5-2 é possível comparar, os valores e os princípios entre as abordagens tradicionais e as ágeis. Cockburn (Cockburn et al., 2001) afirma que as metodologias ágeis não possuem nada de novo, o que as diferencia das metodologias tradicionais é o enfoque e os valores.

**Tabela 5-2: Comparação entre metodologias ágeis e tradicionais**
**Fonte: Adaptado de Software development: Agile vs. traditional (Stoica, Mircea e Ghilic-Micu, 2013)**

| Abordagem Tradicional | Abordagem Ágil |
|---|---|
| Preditivo – Detalhar tudo, mesmo o que ainda não é bem conhecido | Adaptativo – Conhecer o problema e resolver primeiro as prioridades |
| Rígido – Seguir as especificações definidas | Flexível – Adaptar-se a requisitos atuais, que podem mudar face ao planeado |
| Orientado a processos – Seguir os processos possibilita garantir a qualidade | Orientado a pessoas – Motivar as pessoas para que se comprometam e sejam produtivas |
| Documentação - Gera confiança | Comunicação – Gera confiança |
| Sucesso – Entregar o planeado | Sucesso – Entregar o desejado |
| Cliente – Pouco envolvido | Cliente – Comprometido, experiente, cooperativo |
| Requisitos – Estáveis | Requisitos – Emergentes |
| Reestruturação – Cara | Restruturação – Barata |
| Planeamento direciona aos resultados | Os resultados direcionam ao planeamento |
| Objetivo – Controlo para alcançar o objetivo planeado | Objetivo – Simplificar o processo de desenvolvimento |
| Arquitetura – Desenho para todos os requisitos | Arquitetura – Desenho para os requisitos atuais |
| Testes – No final da aplicação | Testes – Durante todos os *sprints* com envolvimento do cliente |

## 5.3 Comparação das Metodologias Ágeis

As metodologias ágeis convergem em muitas das suas práticas e princípios, uma vez que todas foram construídas com base na mesma perspetiva, ou seja, nos ideais do Manifesto Ágil (Begel e Nagappan, 2007). Além disso, todas elas pretendem tratar, diferentemente, a forma como lidamos com um projeto. Com base na análise realizada por Abrahamsson (Abrahamsson *et al.*, 2002), apresentam-se, de forma breve, na Tabela 5-3, os pontos-chave, principais recursos e pontos negativos de cada metodologia.

**Tabela 5-3: Comparação entre metodologias ágeis**
**Adaptado de Qumer e Henderson-Sellers, 2008 e harma, e Bawa, 2017**

| Metodologia | Pontos-Chave | Principais recursos | Pontos Negativos |
|---|---|---|---|
| Scrum | - Processos simples;<br>- Equipas de desenvolvimento organizadas;<br>- Ciclos de entrega curtos (15 a 30 dias). | - Paradigma de orientação ao cliente. | - Requer outras abordagens para complementar o ciclo de vida;<br>- Focado na gestão do projeto;<br>- Ausência de práticas e técnicas de desenvolvimento. |
| Extreme Programming | - Desenvolvimento orientado ao cliente;<br>- Equipas pequenas;<br>- Ciclos de entrega curtos. | - *Refactoring*;<br>- Desenvolvimento orientado a testes;<br>- *Pair programming* | - Poucas práticas de gestão;<br>- Ausência de documentação;<br>- Focado essencialmente no desenvolvimento;<br>- Indicado para equipas pequenas. |
| Kanban | - WIP – Work in Progress;<br>- Começar com o que existe;<br>- Ver o trabalho. | - Visualização do fluxo de trabalho num quadro. | - Focado na gestão do projeto;<br>- Desenhado para manutenção;<br>- Ausência de práticas e técnicas de desenvolvimento. |
| Scrumban | - Visualização do fluxo de trabalho num quadro;<br>- WIP – Work in progress;<br>- Ver o trabalho. | - Desenhado para as constantes mudanças de trabalho. | - Focado na gestão do projeto;<br>- Ausência de práticas e técnicas de desenvolvimento. |

| Metodologia | Pontos-Chave | Principais recursos | Pontos Negativos |
|---|---|---|---|
| Scrum & XP | - Equipas de desenvolvimento organizadas;<br>- Ciclos de entrega curtos (15 a 30 dias). | - *Refactoring*;<br>- Desenvolvimento orientado a testes;<br>- *Pair programming*. | - Ausência de documentação;<br>- Indicado para equipas pequenas. |
| Outsystems | -*Demos* constantes ao cliente;<br>- Ciclos de entrega curtos (2 a 4 semanas). | - Orientada para o cliente. | - Ausência de documentação;<br>- Ausência de práticas e técnicas de desenvolvimento. |
| Adpative Software Development | - Cultura Adaptativa;<br>- Desenvolvimento baseado em componentes iterativos. | - As organizações são vistas como sistemas adaptativos. | - Encontra-se fortemente ligado a cultura e conceitos. |
| Crystal | - Família de métodos;<br>- Todos os métodos possuem os mesmos valores e princípios. | - O método estabelece princípios;<br>- Métodos mais indicados com base no tamanho e criticidade do projeto. | - Caraterísticas de processos preditivos;<br>- Amplitude de recursos pode tornar o processo complexo. |
| Dynamic Systems Development Method | - Controlo para desenvolvimento rápido de aplicações (RAD);<br>- Tempo estabelecido por iteração;<br>- Equipas autónomas;<br>- Consórcio ativo para manter a metodologia. | -Utilização de prototipagem;<br>- Diversos papéis do utilizador. | - Sugere que os requisitos devem estar estáveis;<br>- Rigidez nos princípios. |
| Feature-Driven Development | - Processo dividido em 5 passos;<br>- Desenvolvimento baseado em componentes orientadas a objetos;<br>- Pequenas iterações. | - Método simplista;<br>- *Design* e implementação por funcionalidades;<br>- Modelagem de objetos. | - Foco em *Design* e implementação;<br>- Requer outras abordagens para complementar o ciclo de vida. |

Com base nas perspetivas apresentadas na secção 5.1, será exibida uma comparação usando três perspetivas: pelo ciclo de vida, apoio à gestão de projetos e princípios abstratos vs diretrizes concretas. A Figura 5-1 apresenta a comparação realizada por Abrahamsson (Abrahamsson *et al.*, 2002), onde a barra superior representa se um método fornece apoio à gestão de projetos, a do meio compara-a com o ciclo de vida de um projeto e a última baseia-se em princípios

abstratos (a branco) e princípios concretos (a cinza). Nos restantes blocos, o cinza indica que o método compreende a perspetiva mostrada ou branco para a sua ausência.

- Ciclo de vida – Com base na Figura 5-1, pode-se constatar que a maioria das metodologias de desenvolvimento encontram-se focadas em partes do ciclo de vida. Contudo, o DSDM procura dar apoio e abranger todo o projeto. Métodos como XP, Scrum e FDD encontram-se focados em especificação de requisitos, desenvolvimento e implementação do projeto e por fim, também testes de todo o sistema. As metodologias híbridas tentam ocultar todo o ciclo de vida, mas não se tornam tão completas como o DSDM.

- Gestão de projetos – Conforme pode observar-se na Figura 5-1, as barras correspondentes ao XP, não se encontram todas preenchidas. Isto resulta da existência de escassas práticas de gestão, não oferecendo assim uma perspetiva global. Por outro lado, o Scrum é explicitamente destinado para a gestão de projetos ágeis. Neste âmbito, Schwaber e Beedle (Schwaber e Beedle, 2002) sugerem a utilização de métodos híbridos baseados em Scrum, nomeadamente o Scrum com XP. O foco do ASD é a alteração da cultura do desenvolvimento de *software*, afirmando que a gestão também deverá melhorar em resposta às mudanças dos projetos, sendo uma filosofia contraditória em comparação ao Kanban, a qual refere que devemos começar com o existente. O DSDM é um facilitador na gestão dos projetos. O processo de acompanhamento diário procura aumentar a capacidade de reação às mudanças. No Crystal, a gestão de projetos centra-se no aumento da capacidade de escolher o método correto para o objetivo do projeto (Cockburn, 2004).

- Princípios abstratos vs diretrizes concretas – Conforme pode observar-se na Figura 5-1, são poucas as metodologias que contêm esta barra preenchida. Para Nandhakumar e Avision (Avison e Fitzgerald, 2003), um método só é útil e eficaz se oferecer diretrizes concretas e não deve referir-se apenas a princípios abstratos. No âmbito das metodologias ágeis refere-se a orientações sobre como uma tarefa específica deve ser executada. A metodologia mais forte nesta área é o XP, já que foi derivada de cenários práticos.

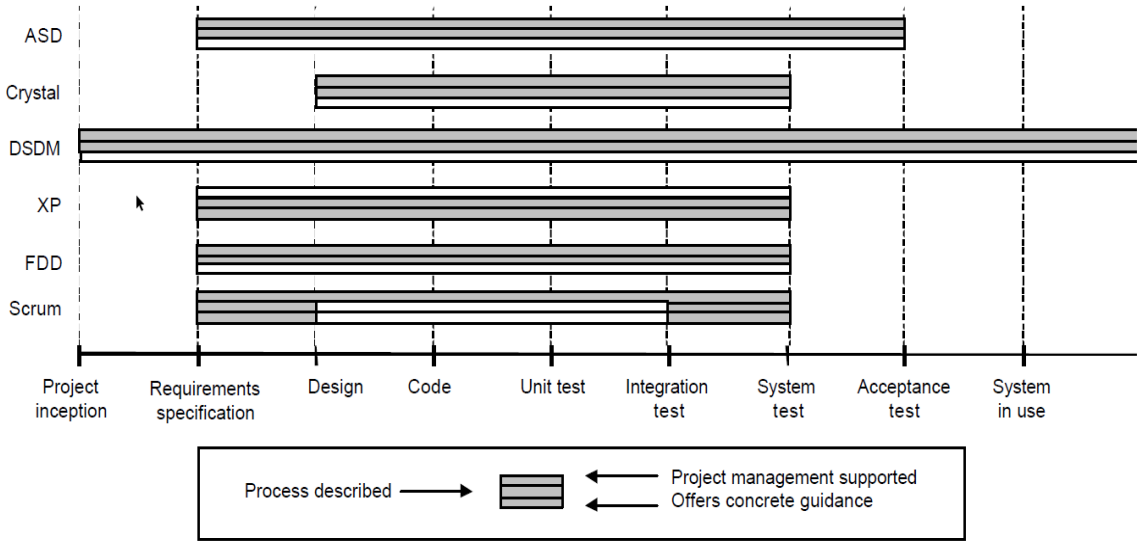


**Figura 5-1: Ciclo de vida FDD**
**Fonte: (Abrahamsson et al., 2002)**

Uma vez que as metodologias ágeis XP e Scrum, são as mais utilizadas, sobretudo no contexto de metodologias híbridas, procedeu-se a uma comparação entre ambas, tendo em conta os seguintes aspetos: documentação; interação com o cliente; tamanho e complexidade do projeto; requisitos e construção do *software*. Estes aspetos são utilizados por autores como Kiran Hiwarkar (Hiwarkar, Doshi e Chinta, 2016) , Sriram Rajagopalan (Rajagopalan e Mathew, 2016), Fahad (Fahad *et al.*, 2017).

- Documentação – Scrum e XP apenas produzem a documentação estritamente necessária;

- Interação com o cliente – Tanto Scrum como XP, necessitam de grande envolvimento e compromisso por parte do cliente durante o processo de desenvolvimento;

- Tamanho e complexidade do projeto – O XP adapta-se melhor a projetos de dimensão mais reduzida e é um dos métodos mais adequados para projetos que exigem mudanças por considerar a comunicação permanente com o cliente. O Scrum adapta-se a qualquer tipo de projeto;

- Requisitos de *software* – No XP existe um conjunto de práticas que ajudam no levantamento de requisitos como o jogo do planeamento, a criação das *user stories* e a identificação dos intervenientes. Todavia, não é realizada nenhuma distinção entre requisitos funcionais e não funcionais. Um dos pontos mais fortes relaciona-se com a possibilidade de ser elaborada a descrição dos testes funcionais com a colaboração do

cliente. No Scrum, todos os requisitos são colocados na lista de *product Backlog*, priorizados e classificados por todas as partes envolventes, complementando o XP que não faz a distinção;

- Construção do *software* – Neste ponto, o Scrum é omisso, uma vez que o seu foco é a gestão de *software* (Schwaber e Sutherland, 2013). No entanto, o XP encontra-se muito orientado para a implementação do *software*, complementando assim o Scrum. Podem-se salientar algumas caraterísticas do XP como a conceção simples, o código é propriedade de todos os programadores, *pair programming*, *refctoring* e integração contínua. Uma das mais-valias do XP são os testes presentes em todas as fases do processo de desenvolvimento.

Estes dois métodos de desenvolvimento têm, na maioria das vezes, aplicação em projetos ou são utilizados para melhorar a produtividade de uma equipa, já que se completam.

## 5.4 PMBOK *Versus* Scrum

O PMBOK é um corpo de conhecimento sobre gestão de projetos, reconhecida internacionalmente como *standard*, como foi descrito em capítulos anteriores, utilizado em métodos híbridos com metodologias ágeis, nomeadamente Scrum. Como tal, o PMBOK é utilizado como referência para comparação com o SCRUM.

A análise destas duas metodologias teve como base a quinta edição do PMBOK (Institute, 2013), uma vez que este possui um maior número de processos e diretrizes que o Scrum. Todos os processos do PMBOK foram comparados com o *Scrum* e foi atribuída uma nota entre 0 se o Scrum não cumpre com esse processo ou 4, se cumprido na íntegra. A tabela comparativa pode ser consultada no Anexo 3.

A quinta edição o PMBOK identifica 47 processos e uma vez que é a base da comparação é atribuído o máximo a cada processo, perfazendo 188 pontos. Por sua vez, o Scrum reuniu um total de 102 pontos, cobrindo cerca de 54% da totalidade de processos do PMBOK. Tal como se pode verificar na Figura 5-2, o Scrum corresponde em quase todos os grupos a metade do PMBOK.

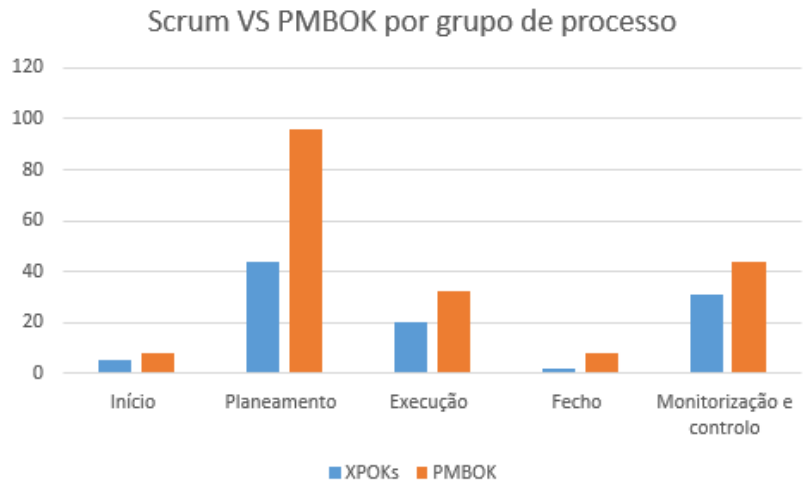


**Figura 5-2: Taxa de cobertura de processos Scrum vs PMBOK por grupo de processo**

Para se compreender as diferenças entre cada grupo de processos, é possível constatar na Figura 5-3, na qual conseguimos perceber que o Scrum não contempla as áreas de recursos e custos, enquanto, que a gestão de riscos dispõe de uma taxa de cobertura muito reduzida.

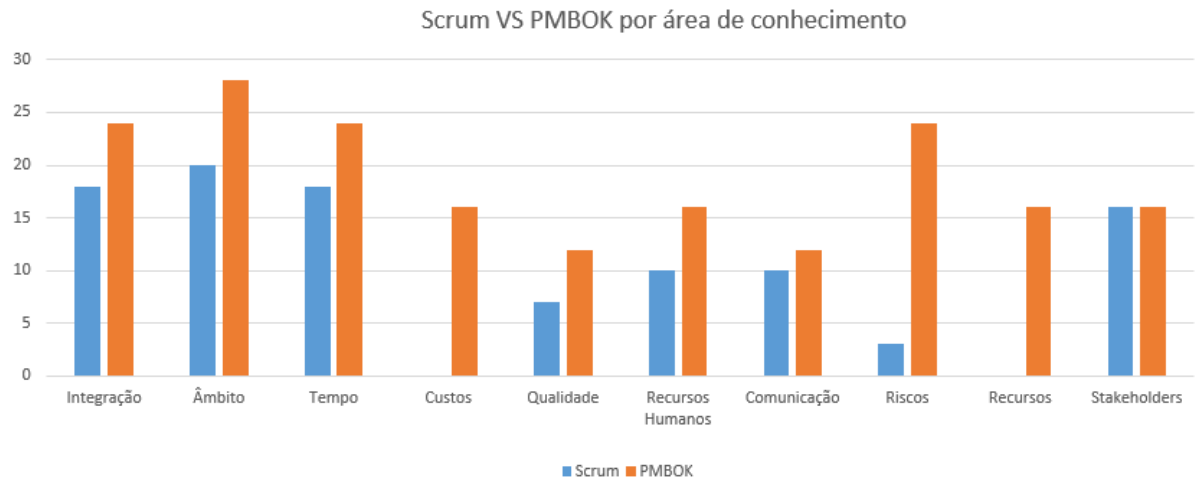


**Figura 5-3: Scrum vs PMBOK por área de conhecimento**

De seguida, são apresentadas as justificações por cada área de conhecimento que fundamentam as diferenças apresentadas na Figura 5-3.

- Integração – No PMBOK, a gestão de integração inclui os processos e as atividades necessárias para identificar, definir, combinar, unificar e coordenar os vários processos, nomeadamente dá a autorização necessária ao gestor de projeto. O planeamento é realizado de forma formal e detalhado no início do projeto. Por seu turno, no Scrum, o plano do projeto corresponde ao *Backlog* que é atualizado durante o decorrer do projeto. A diferença que se observa na Figura 5-3, pode ser justificada com o facto de o Scrum

não possuir algo semelhante ao termo de abertura, nem determinar como encerrar o projeto.

- Âmbito – No PMBOK, o âmbito tem como finalidade garantir que o projeto seja concluído com o esforço necessário, sendo detalhado no início e documentado na especificação de requisitos. Além disso, é útil na gestão de alterações que possam ocorrer com o decorrer do projeto. Já o Scrum define o âmbito sem detalhe ou resumidamente, pois a cada *sprint* os requisitos são detalhados e priorizados, o que obriga à envolvência de toda a equipa de projeto, inclusive o cliente.

- Tempo – No PMBOK é estimado o esforço e a duração de cada atividade, realizando um cronograma detalhado que contém todas as atividades para o sucesso do projeto. No Scrum também é elaborado um cronograma orientado ao produto que será produzido a cada *sprint*, com a colaboração do cliente.

- Custos – No PMBOK, o objetivo da gestão dos custos é garantir que o projeto termina dentro do orçamento estabelecido. Quando existe uma alteração, é o gestor de projeto que determina quais as melhores práticas e decide o modo como as implementar. No Scrum, as alterações podem ser levadas a efeito em qualquer momento, desde que exista consentimento do cliente, não existindo qualquer tipo de controlo.

- Qualidade – As metodologias reconhecem a importância da qualidade, mas garantem-na de forma desigual. No PMBOK, é efetuado o plano dos testes a partir das especificações iniciais. No Scrum, durante cada *sprint* a equipa valida se o pedido cumpre todos os requisitos pedidos pelo cliente e são realizados *Review* ao termo de cada *sprint*.

- Recursos Humanos – No PMBOK são atribuídos papéis e responsabilidades a todos os elementos da equipa, sendo primordial organizar e identificar toda a equipa e documentar todas as funções e relações hierárquicas. Por sua vez, o Scrum indica que a equipa é auto-organizada e a confiança e a colaboração entre os elementos é a chave do sucesso.

- Comunicação – No PMBOK, a comunicação é realizada de forma formal e documentada, sendo o principal objetivo documentar todos os acontecimentos a fim de

evitar conflitos entre o cliente e o gestor de projeto. No Scrum, devido ao facto de ser ágil promove um constante *feedback* entre todos os *stakeholders*, sendo o processo de comunicação feito de forma colaborativa entre todos os envolvidos no projeto, obrigando todas as partes a terem maturidade suficiente para evitar conflitos.

- Riscos – As metodologias preocupam-se em evitar riscos, embora o façam de forma bastante diferente, enquanto no PMBOK é realizado um plano formal para a gestão de riscos, garantindo a identificação, avaliação, quantificação, planeamento, monitorização e controlo. O Scrum reduz os riscos com a colaboração constante do cliente com a equipa de desenvolvimento, participando diariamente nas reuniões. A cada *sprint* é realizada uma retrospetiva, onde é discutido o que correu bem e mal durante o *sprint* para nas próximas iterações melhorar os pontos fracos.

- Recursos – No PMBOK, todas as aquisições são efetuadas a partir do âmbito, garantindo todo o acompanhamento necessário. Apesar disto, todo o processo é formalizado com base num contrato. No Scrum não existe esta formalização, nem a preocupação de definir detalhadamente o processo de aquisições;

- *Stakeholders* – As metodologias consideram processos relacionados com a identificação de *stakeholders* para assegurar que as expetativas serão satisfeitas. Contudo, no PMBOK, o gestor de projeto preocupa-se na fase inicial em escutar todos os envolvidos para que existam menos mudanças no decorrer do projeto. O Scrum prefere ter os *stakeholders* envolvidos no momento em que são necessários, como no planeamento do *sprint*.

## 5.5 Sumário

Neste capítulo, constatou-se que a maioria das metodologias ágeis carecem de documentação, pois encontram-se essencialmente focadas na gestão do projeto, mesmo quando combinadas entre si. A metodologia mais completa e estudada foi o Scrum & XP, uma vez que os aspetos negativos de uma abordagem são complementados pela outra.

No próximo capítulo serão combinadas diferentes metodologias, para se criar uma metodologia ágil híbrida. Além disso será efetuado um teste piloto para se proceder com a sua validação.

# 6. Criação e Validação da Metodologia Ágil

Este capítulo pretende responder à necessidade de ser criada uma metodologia ágil híbrida que permita obter mais documentação, práticas de desenvolvimento e que possua um âmbito fechado no início do projeto. Assim, é proposto o modelo XPOKs, construído a partir das metodologias Kanban, Outsystems, Scrum, XP e PMBOK.

Inicialmente, será apresentada a criação do XPOKs, procedendo-se à sua validação, recorrendo a um questionário e a uma análise comparativa com o PMBOK. Por fim, será efetuada uma análise autocrítica a este capítulo.

## 6.1 Criação da Metodologia Ágil XPOKs

O nome da metodologia XPOKs é um acrónimo de "XP, PMBOK, Outsystems, Kanban e Scrum". Esta metodologia pretende contribuir para a melhoria da gestão de projetos de *software*, passível de ser aplicada a pequenas e médias empresas de desenvolvimento de *software* que utilizem como metodologia Scrum ou Kanban. O cenário empresarial que originou este desafio utiliza duas metodologias, atendendo ao tipo de projeto. Ou seja, se for possível, ter um planeamento a longo prazo, permitindo conhecer os itens do *backlog*, opta-se por utilizar Scrum. Por sua vez, quando o projeto é de manutenção evolutiva ou corretiva e o trabalho resulta de pedidos ocasionais por parte do cliente é habitual a utilização da metodologia Kanban, por não ser necessário implementar tantas práticas e eventos como o Scrum exige.

A utilização de Scrum ou Kanban não apresenta recomendações relativamente às práticas a serem implementadas pelas equipas, o que as leva a recorrer a práticas externas, não sendo uniforme entre todas as equipas da empresa.

Uma vez que as metodologias ágeis optam por produzir a documentação estritamente necessária, leva as equipas, em diversas situações, a optarem por pequenos comentários no código-fonte. Quando entram elementos novos num projeto têm muitas vezes de ler código para entenderem o que faz o projeto, tornando assim a sua integração mais lenta. O mesmo se passa quando existe a necessidade de introduzir alterações, o que dificulta na identificação de possíveis impactos. A ausência de documentação faz com que, frequentemente, os projetos iniciem sem que possuam uma definição do âmbito, e quando chega o momento de fechar um projeto, o cliente consegue argumentar que ainda faltam desenvolver mais funcionalidades para considerar o projeto concluído.

Neste seguimento, XPOKs é uma abordagem ágil, que resulta da intersecção entre modelos ágeis com as boas práticas recomendadas pelo PMBOK. Continua a ter as caraterísticas dos modelos ágeis, apresentando-se como uma alternativa em projetos que requerem rigor na sua execução, mas que pretendem que o cliente se encontre presente e envolvendo entregas frequentes.

Na Figura 6-1 pode-se ver que o XPOKs teve como base o XP, mas incorporou partes de outras metodologias, tal como Outsystems, Scrum e algumas recomendações do PMBOK. Deste modo, nas próximas subsecções serão descritos os valores e princípios, eventos, práticas e artefactos, papéis e responsabilidades da metodologia XPOKs.

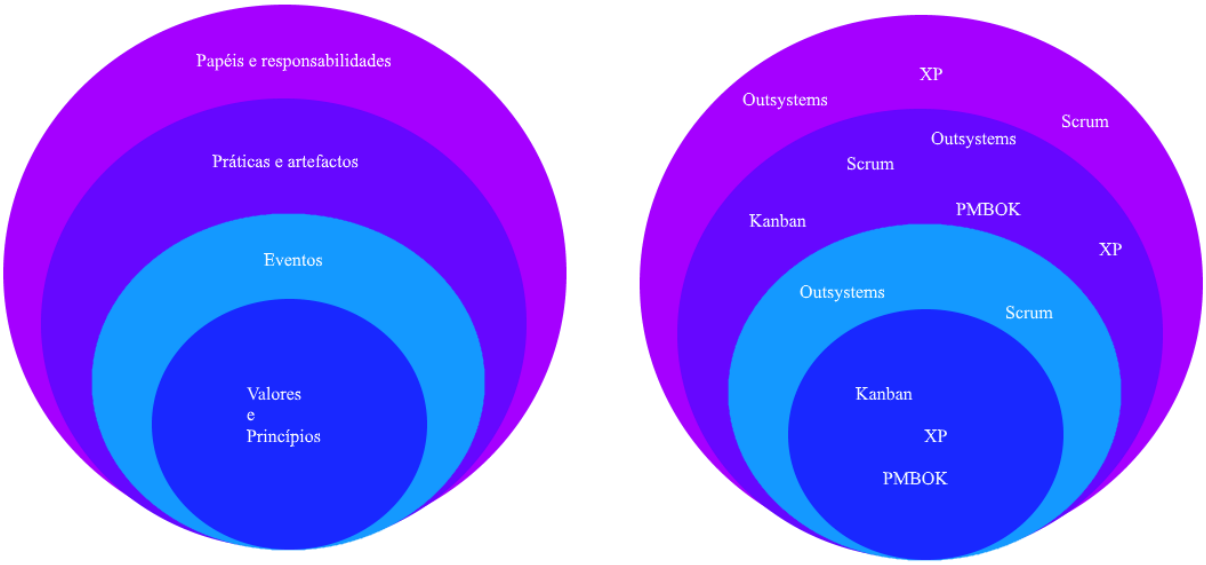


**Figura 6-1: Base XPOKs**

### 6.1.1 Valores e Princípios

De seguida, são expostos os valores e princípios do XPOKs, baseados nas metodologias estudadas anteriormente, tais como XP e Kanban, ou aqueles que foram adquiridos através de experiências profissionais.

- Começar com o que existe – A adoção da metodologia deve ser progressiva, partindo da análise ao processo atual, realizando a verificação dos pontos fracos do processo atual. Não é obrigatório implementar todo o método;

- Mudança contínua e incremental – Toda a equipa deve aderir à mudança contínua e incremental. Nada é perfeito e pode ser sempre melhorado;

- Regras bem definidas – Sempre que inicia um projeto a equipa deve definir as regras a serem implementadas. Contudo, estas devem reunir consenso entre todos e, em caso de conflito, o *team leader* desempata, podendo as regras ser melhoradas, a cada iteração;

- Reflexão – É necessário ser autocrítico e analisar os problemas para assim se melhorar e aprender com o passado;

- Passos pequenos – As histórias dos clientes devem ser "partidas em pequenas histórias" para que possam ser implementadas num dia de trabalho, reduzindo o risco de integração do código fonte e facilitando os testes;

- Máximo de 15 minutos – Um elemento da equipa não deve permanecer mais do que 15 minutos na resolução de uma dificuldade. Após esse período, deve requerer auxílio dos restantes elementos;

- Resumir em 120 caracteres – Todos devem ter a capacidade de resumir, em poucas palavras, a sua atividade ou resumir o problema que têm para resolver, para que os outros indivíduos o consigam perceber (podem pensar em voz alta ou falarem com um amigo imaginário) e, sempre que possível, devem incluir imagens;

- Humanidade – Os elementos das equipas são pessoas com as suas próprias necessidades, vontades, ideologias e ideias, pelo que todos devem ser respeitados. A empresa deve ter a capacidade de contribuir para o equilíbrio entre o lado pessoal com o profissional;

- *Feedback* – Como metodologia ágil que é, o XPOKs requer um permanente retorno de opinião do cliente e da equipa de desenvolvimento, sendo esta uma forma de reduzir o risco de *bugs* e de aproximar as expetativas à realidade;

- Documentação é um ativo de conhecimento – A documentação atualizada é a fonte ideal para a equipa consultar informações relacionadas com o modo de funcionamento do produto. Além de viabilizar as decisões tomadas, a documentação ajuda na integração de novos elementos, ou até mesmo de membros da equipa que não conheçam uma funcionalidade. Deve ser viável rastrear todos os pedidos, incluindo as suas especificações e testes efetuados. Deve ainda ser possível a qualquer elemento consultar o termo de abertura assim como a todos os planeamentos dos *sprints*.

### 6.1.2 Eventos

Os eventos adotados são essencialmente baseados no Scrum, sendo eles:

- *Kick-Off* - Momento formal para dar início ao projeto, no qual o cliente fica a conhecer as pessoas com quem irá trabalhar e a equipa o projeto a um nível de elevada granularidade. Daqui deve resultar o documento inicial, designado por termo de abertura;

- *Sprint* – Deve ter uma duração de duas a quatro semanas e no fim de cada *sprint* deve ser entregue ao cliente uma nova versão do seu produto. A primeira fase do *sprint*

precisa de incluir o planeamento, envolvendo todos os elementos do projeto. No decurso do *sprint* deve existir uma reunião diária e, no final do *sprint* deve ser apresentada uma *demo* ao cliente, seguida de uma retrospetiva;

- Planeamento do *sprint* – Deve ser definido todo o trabalho a produzir nessa iteração e revisado/ordenado o *Backlog*, produzindo assim a visão do *sprint*;

- Reuniões diárias – É um evento de frequência diária, no qual a equipa realiza um ponto de situação relativamente às atividades do dia anterior, assim como um planeamento desse dia. É importante o cliente encontrar-se presente nestas reuniões diárias. Em caso de ausência, os seus interesses são defendidos pelo *Engagement Manager*;

- *Demo* – Momento em que a equipa apresenta ao cliente o resultado final do seu trabalho, obtendo *feedback* do cliente para possíveis melhorias. O *Engagement Manager* é responsável pela realização do ponto de situação, ou seja, apresenta em que estado se encontra o projeto, enquanto a apresentação da *demo* é da responsabilidade do *Team Leader*;

- Retrospetiva do *sprint* – Momento em que a equipa realiza a autoavaliação e indica que aspetos benéficos devem continuar e aqueles que devem ser melhorados, revelando-se, a oportunidade formal para a implementação de melhorias no processo.

### 6.1.3 Práticas e Artefactos

Os artefactos são dos aspetos mais importantes da metodologia, pois permitem uma visão geral do produto a ser desenvolvido, como enunciaram Schwaber e Sutherland (Schwaber e Beedle, 2002). No XPOKs são propostos os seguintes:

- *Product Backlog* – É criado no princípio e pode ser alterado a cada *sprint*, contendo a lista de todas as tarefas a criar, devendo estar ordenada por ordem de prioridades;

- Visão do produto – Como indica a metodologia Kanban, deve ser possível visualizar o trabalho em curso. Deste modo, é proposta a existência de um quadro que resume o estado atual do projeto, devendo ser atualizado em cada iteração. Este quadro, apresentado na Figura 6-2, deve conter o objetivo do projeto, os intervenientes do cliente, as áreas de negócio envolvida, as entregas ao cliente, as mudanças e informações

significativas. É baseado no modelo Canvas, proposto por Osterwalder e Pigneur (Osterwalder e Pigneur, 2010);

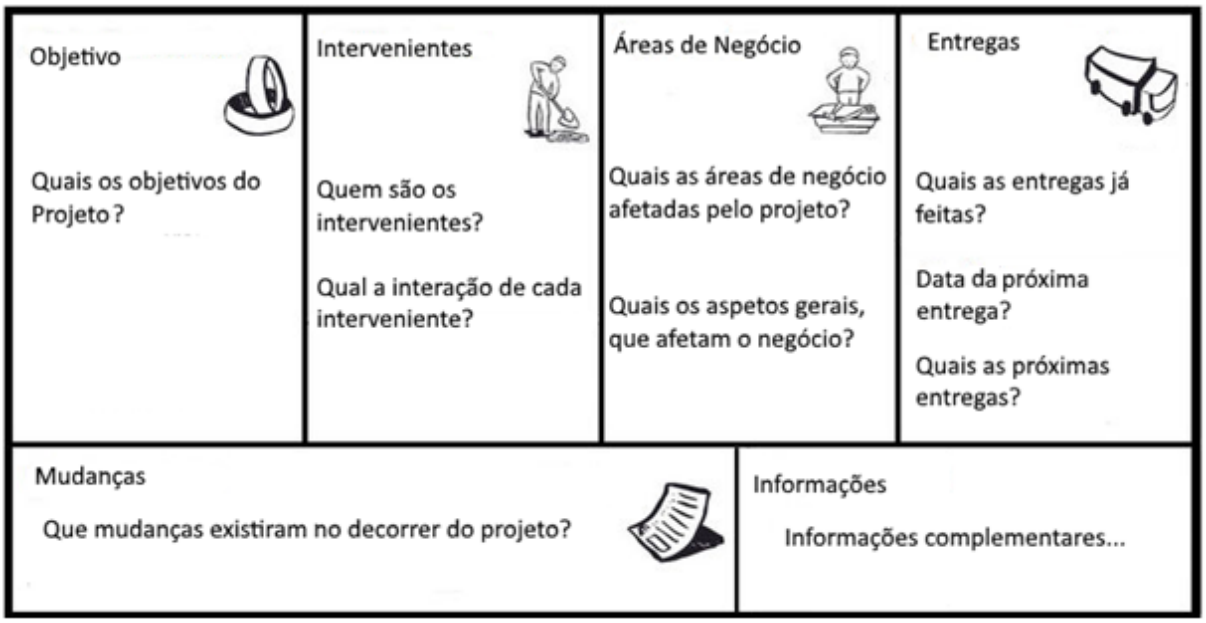


**Figura 6-2: Visão do produto**

- Termo de abertura – O objetivo deste documento é marcar o começo do projeto e alinhar o seu âmbito. O ideal é ser curto para manter o princípio ágil da simplicidade. É sugerido utilizar o documento da visão do produto, uma vez que reúne a informação necessária sobre o projeto, devendo ser preenchido da seguinte forma:
    - Objetivo – Um resumo do projeto e dos seus principais riscos;
    - Intervenientes – Identificar toda a equipa de modo a que todos os elementos do projeto passem a conhecer o *Engagement Manager*;
    - Áreas de negócio – Identifica as áreas de negócio envolventes e as mudanças que o projeto traz, assim como os seus riscos;
    - Entregas – Calendarização das próximas entregas, nomeadamente a data de fecho.
- Histórias (*User Story*) – Descrição da funcionalidade na perspetiva do cliente, incluído o conceito de *definition of done*. Deve conter o que é essencial implementar e testar para

se considerar a história fechada. Deve incluir também a especificação técnica para que possa ser analisada pela equipa de desenvolvimento;

- Modelo de arquitetura – O modelo de arquitetura deve ser definido no início do projeto, não sendo obrigatório seguir um modelo particular. A metodologia sugere o recomendado pela Outsystems, como tal podemos observar na Figura 6-3. Esta é uma arquitetura de quatro camadas, orientada a serviços. A primeira camada deve conter os sistemas externos e padrões de desenho. A segunda inclui as normas e regras de negócio e entidades partilhadas. A terceira camada usa os serviços da segunda e implementa as histórias dos utilizadores. Finalmente, a quarta camada engloba a junção das anteriores, dando uma experiência unificada ao utilizador;

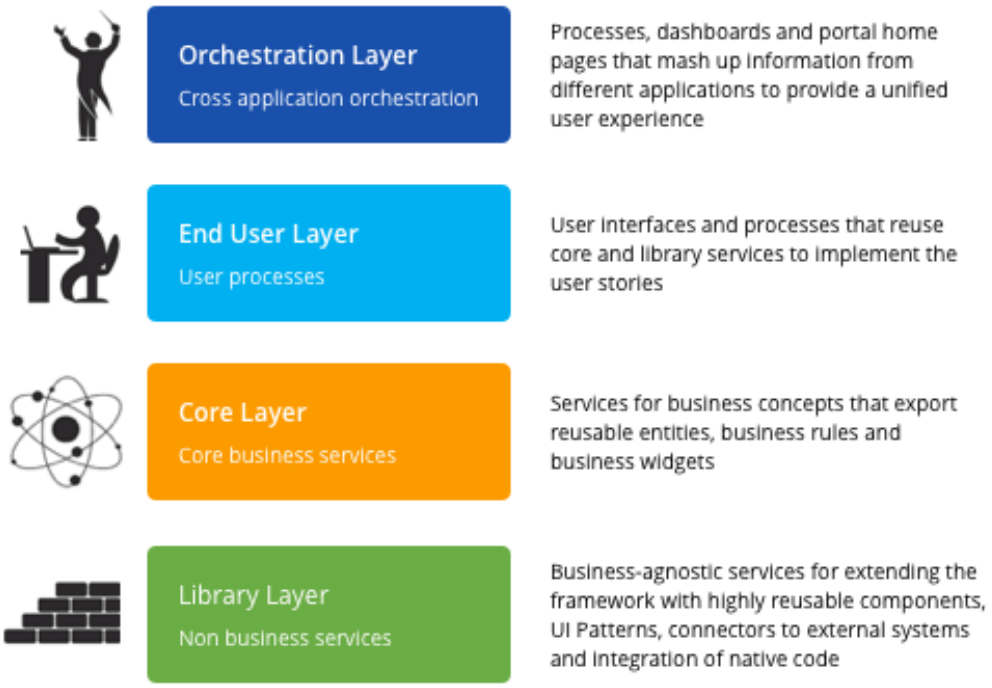


**Figura 6-3: Arquitetura do projeto**
**Fonte: (Outsystems, 2010)**

- *Sprint Backlog* – São as tarefas acordadas para entregar numa iteração, contendo todo o detalhe necessário para a equipa poder implementar;

- Visão do *sprint* – É construído com o planeamento do *sprint* e cria visualmente um quadro contendo as atividades no fluxo, semelhante ao recomendado pela metodologia Kanban. Deve ser atualizado diariamente pela equipa de desenvolvimento para facilitar

a identificação do que se encontra "por fazer", "em desenvolvimento" e "feito". Quando uma tarefa se encontra em desenvolvimento pode ainda encontrar-se tanto em estado de desenvolvimento (DEV), como em testes de qualidade (QA). Inicialmente, todas as tarefas encontram-se no estado "por fazer" e no final, o ideal é estarem todas como feitas. A Figura 6-4 ilustra a visão do *sprint*;

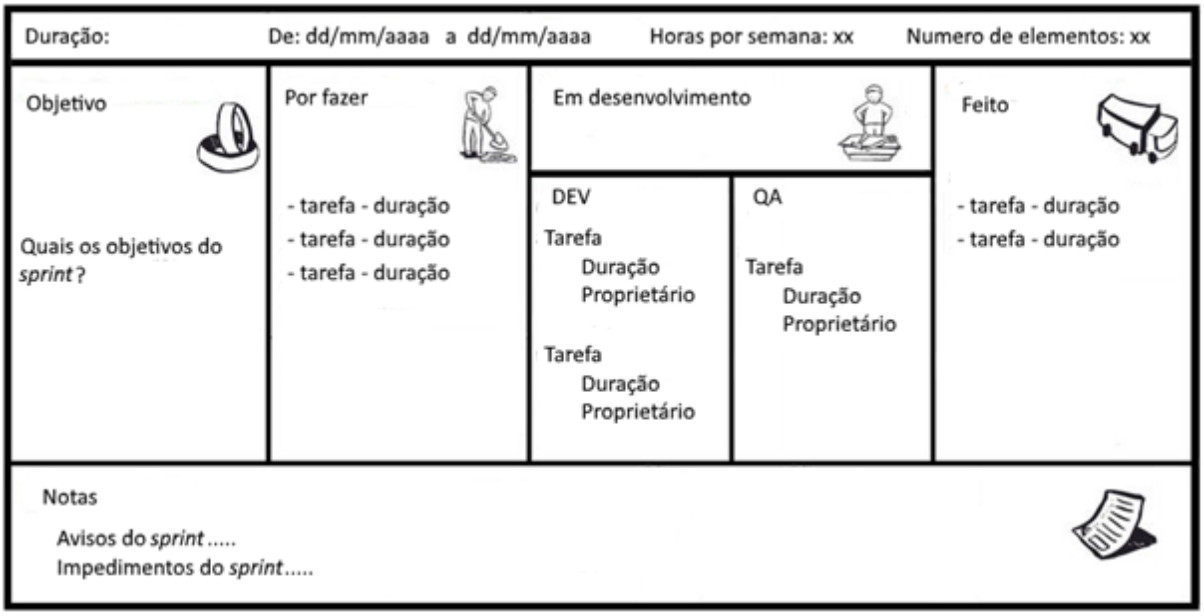


**Figura 6-4: Visão do Sprint**

- Documentação – É fundamental existirem documentos com o registo da alteração de histórias, ficando a alteração associada à história, para compreensão da perspetiva histórica sobre o âmbito. As interações com o cliente devem ficar registadas com o propósito de reter o conhecimento dele sobre as áreas de negócio, devendo sempre, que possível, ligar esta informação com os itens do *Product Backlog*;

- Incremento - É resultado do somatório de todos os itens do *Product Backlog* que foram concluídos durante todos os *sprints*;

- Projeto simples – O projeto deve nascer simples, implementando o que é estritamente necessário e introduzir progressivamente as melhorias necessárias até ao fim, para ser mais fácil a manutenção;

- Testes – Para se concluir o desenvolvimento, os programadores devem realizar os testes, para posteriormente ser revisto pela equipa de QA. O cliente será responsável pela execução de testes de aceitação;

- *Refactoring* – É uma das técnicas usadas para se manter o projeto simples. Este processo permite a melhoria contínua, mantendo a compatibilidade do código-fonte já existente, evitando ainda duplicações desnecessárias;

- *Pair programming* – Deve recorrer-se ao *pair programming* sempre que seja necessário, nomeadamente quando a complexidade da funcionalidade seja considerada alta pela equipa, evitando assim a probabilidade de erros. Além disso, ajuda na evolução dos elementos da equipa;

- Propriedade coletiva – O trabalho desenvolvido é da equipa e qualquer elemento pode introduzir alterações, o que permite que exista um conhecimento mínimo de tudo o que está feito;

- Integração contínua – Antes das reuniões diárias deve existir a integração dos itens produzidos no projeto, o que implica que cada tarefa pode ter a duração máxima de um dia de trabalho, caso contrário não é possível efetuar a integração diária. Para ser realizada a integração contínua, é indispensável existir um sistema de versões;

- Padrões de desenvolvimento – O código-fonte deve obedecer a padrões de desenvolvimento, previamente estabelecidos, e utilizados por todos os elementos da equipa. Para isso, devem ser instituídas regras e todos as devem seguir, sendo da responsabilidade do *Team Leader* assegurar o bom funcionamento.

### 6.1.4 Papéis e Responsabilidades

Os papéis foram herdados e adaptados das metodologias Scrum, Outsystems e XP, e podem ser divididos em dois grupos: cliente e equipa de desenvolvimento.

- Cliente

    - Patrocinador – Tipicamente, é a pessoa que solicita o projeto. É responsável por indicar, por cada área de negócio, as pessoas que estarão envolvidas, para que estas possam ajudar nas especificações das histórias, especificar o comportamento do *software* e proceder com testes de aceitação. Mas em caso de conflito, o patrocinador continua a ser o principal decisor quanto ao âmbito;

- Utilizadores chave – São as pessoas que irão utilizar o resultado do projeto e são a principal fonte de informação relativamente a processos e requisitos. Por isso, é fundamental serem conhecedores profundos da área subjacente ao projeto. Necessitam dispor de alta disponibilidade para comunicarem com a equipa de desenvolvimento. Uma das suas principais responsabilidades relaciona-se com a validação de cada entrega, na qual avaliam se o resultado da entrega se encontra de acordo com as especificações;

- Responsável IT – Elemento nomeado pelo cliente que deve garantir os requisitos técnicos da arquitetura estabelecidos pela equipa de desenvolvimento ou alertar para possíveis impedimentos.

- Equipa de desenvolvimento

  - *Engagement Manager* (EM) – É o elemento que representa o cliente perante a equipa de desenvolvimento. Tem como responsabilidade garantir que os processos e a estrutura do projeto são cumpridos e que o projeto obedece aos objetivos traçados. Deve efetuar a primeira análise dos pedidos do cliente e manter a equipa motivada, de forma a evitar conflitos internos;

  - *Team Leader* (TL) – É um elemento sénior responsável pela arquitetura do sistema. As boas práticas são implementadas pela equipa de desenvolvimento;

  - *Developer* – É o papel responsável por desenhar, codificar, testar e entregar o produto. Deve ter a capacidade de análise para poder verificar impactos e reduzir riscos;

  - Qualidade – Os elementos da qualidade fazem parte da equipa de desenvolvimento, não devendo ser *developers*. São responsáveis por realizar os testes antes de o trabalho ser entregue ao cliente de forma a garantir a implementação dos requisitos e garantir que não possui *bugs*.

### 6.1.5 Princípio de Funcionamento

A proposta de funcionamento do XPOKs compreende três fases: início, ciclo de desenvolvimento e fecho. Esta realidade é muito idêntica ao que acontece com Outsystems, o

qual reconhece uma maior importância ao início e ao fim do projeto. É possível, ainda comparar, as fases do ciclo de desenvolvimento do XPOKs com o grupo de processo do PMBOK, tal como podemos examinar na Figura 6-5.

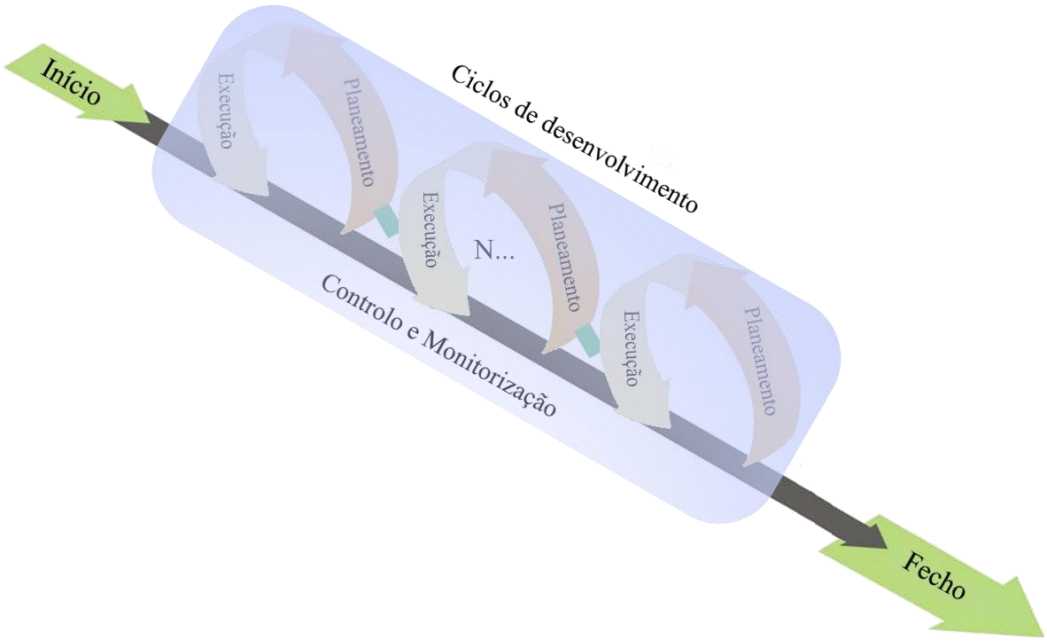


**Figura 6-5: Fases do ciclo de desenvolvimento do XPOKs**

O início do projeto é uma fase crucial para o seu sucesso, na medida que é neste momento que se define o seu âmbito. Nesta fase, o cliente deve definir as histórias do projeto em conjunto com o EM e TL, de forma a transformarem a linguagem do cliente em tarefas do *product backlog* e todos os elementos da equipa de desenvolvimento perceberem o projeto, como podemos observar na Figura 6-6. Para ajudar este momento, devem ser obtidas respostas a algumas perguntas:

1. Qual é a visão do projeto?
2. Qual é o objetivo do projeto?
3. Que restrições tem o projeto?
4. Quem são os envolvidos no projeto?

Após a construção do *product backlog*, existe uma tarefa elementar que é o planeamento das entregas que irão existir no decorrer do projeto, particularmente o momento final, conhecido por *Go Live*[8].

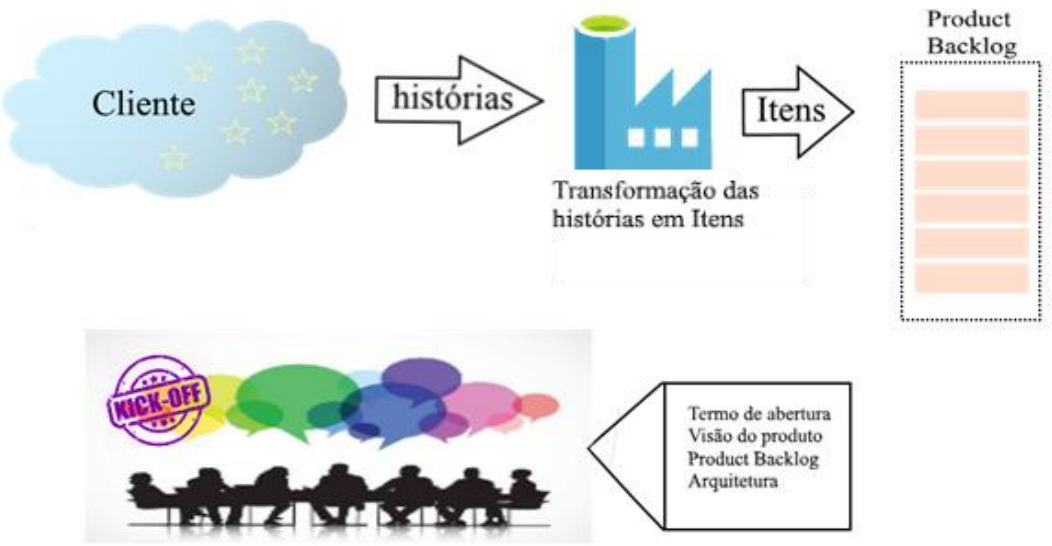


**Figura 6-6: Início do XPOKs**

O papel desempenhado pela equipa de desenvolvimento, nesta fase consiste, na definição da arquitetura e na definição das boas práticas, as quais todos os elementos devem usar de forma a garantir que a equipa trabalha como um todo. No tempo disponível da equipa, esta deve iniciar o desenho das funcionalidades mais críticas para evitar riscos.

O termo de abertura de um projeto é um dos artefactos mais importantes nesta etapa, que reforça o compromisso entre o cliente e a equipa no entendimento dos requisitos do projeto e na definição do âmbito, não perdendo a agilidade, pois durante o desenvolvimento pode ser adaptado.

Os ciclos de desenvolvimento são compostos por vários *sprints*, podendo-se observar na Figura 6-7 o modo como funciona um *sprint*. O círculo amarelo mostra a gestão do projeto, o qual se baseia no modelo Scrum e o círculo azul engloba as práticas e ferramentas usadas para o trabalho diário, baseado na metodologia XP.

O *sprint* inicia sempre com uma reunião de planeamento, onde se encontram o cliente e a equipa de desenvolvimento, sendo o principal objetivo definir quais os próximos itens do *backlog* do

[8] Go live – Momento em que um projeto entra em produção, podendo ser mais do que um.

produto a serem executados e definidos com mais rigor e detalhe. Além disso, deve ser verificado todo o *backlog* e reordenadas as prioridades, caso necessário. Deste evento, resulta um artefacto bastante importante, que permite verificar e controlar o progresso do *sprint*: a visão do *sprint*.

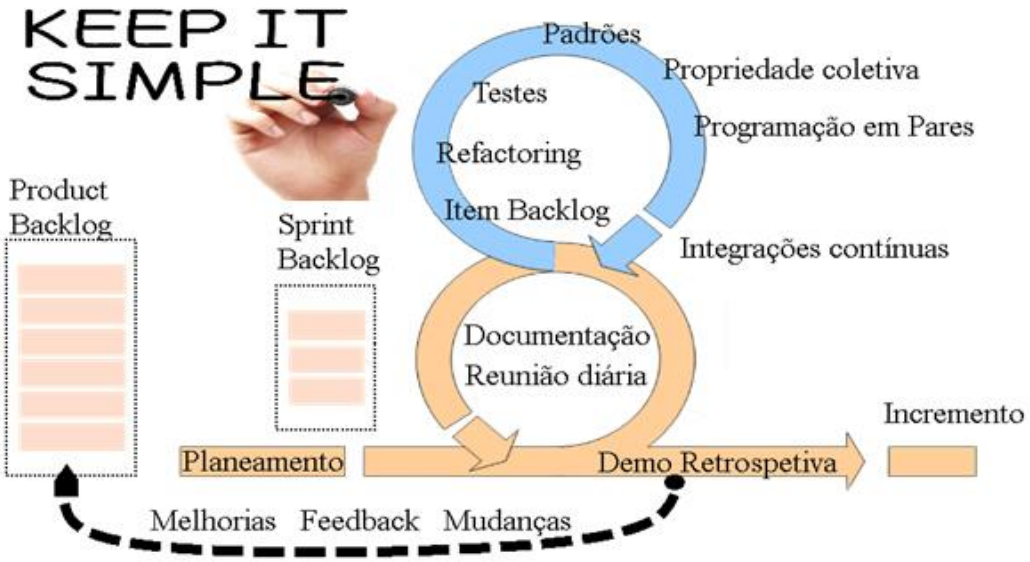


**Figura 6-7: Detalhe do ciclo de desenvolvimento do XPOKs**

Durante o desenvolvimento do produto, a equipa deve realizar, diariamente, um ponto de situação do progresso do projeto para o cliente. Aí, é monitorizado o avanço do *sprint*, verificando-se o trabalho realizado e o que se encontra por iniciar. São analisadas as dificuldades, constrangimentos e verificado o planeamento do trabalho a realizar até à próxima reunião, possibilitando a redução do risco.

Por cada item concluído, os elementos da equipa devem sempre analisar, codificar e testar, efetuando, no final a integração do seu trabalho com o projeto completo. A equipa de qualidade deve sempre validar todos os itens, verificando se cumprem com o *definition of done* e no caso de ser encontrada alguma inconformidade, deve apontá-la e requerer à equipa de desenvolvimento a sua correção. Devem ser registadas, no item, as evidências dos testes.

Embora o XPOKs recorra a técnicas como *pair programming* e *refactoring*, não implica obrigatoriamente a sua aplicação a todos os itens. Deve recorrer a elas, sempre que necessário e especialmente quando se encontram em causa itens considerados importantes.

O final do *sprint* coincide com dois eventos de extrema importância:

- A *demo*, onde é mostrado ao cliente o resultado do *sprint* e recolhido *feedback* que realimenta o *backlog*. É de salientar que se o cliente pretende uma alteração à lista de requisitos irá ter impacto sobre o trabalho realizado, que implica uma mudança ao que foi feito. Esta alteração deve ficar registada na visão do produto, para que o cliente tenha conhecimento da alteração da lista de requisitos.

- A retrospetiva é um evento no qual a equipa analisa o seu desempenho e faz uma autocrítica e aponta possíveis melhorias.

O momento final de um *sprint* termina com um incremento, ou seja, a entrega ao cliente de todo o trabalho desenvolvido durante o *sprint*, para que ele possa realizar os seus testes. Pretende-se que se encontre dentro dos parâmetros de qualidade estipulada e acrescente valor para o negócio.

A fase do fecho do projeto é normalmente realizada em dois *sprints*. O primeiro corresponde à estabilização, muito idêntico ao que acontece nos *sprints* durante o desenvolvimento, mas mais simplificado. Por não existir planeamento de *sprint*, o *backlog* do produto é a fonte direta do trabalho, podendo ser entregue ao cliente com a periodicidade que ele desejar. O segundo corresponde à formação acompanhada pela equipa de desenvolvimento, em especial à equipa de IT do cliente. Na Figura 6-8 podemos analisar o modelo de trabalho do fim do XPOKs.

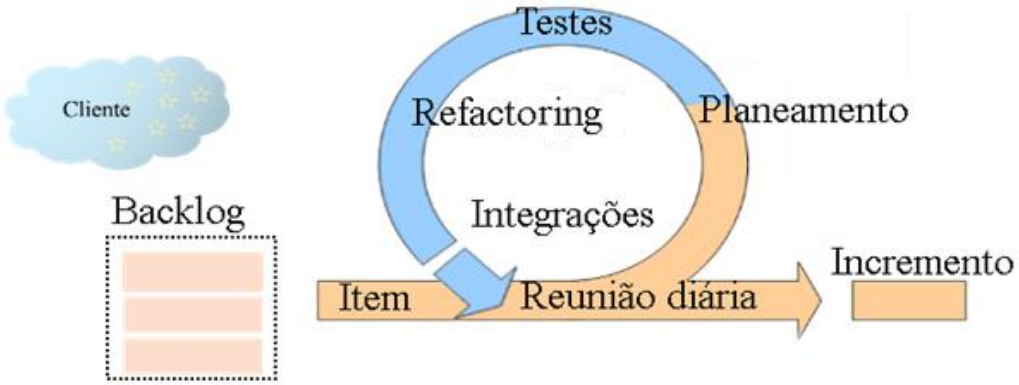


**Figura 6-8: Fim do XPOKs**

O XPOKs pretende seguir a filosofia proposta pela metodologia Kanban. Partindo do que existe, com a adaptação gradual dos processos e de acordo com as necessidades. Portanto, a adoção do XPOKs não implica que seja implementado na íntegra.

A fase do fecho na metodologia do XPOKs pode ser usada para projetos onde não é possível estimar a quantidade de trabalho necessária. Tipicamente, são projetos de manutenção corretiva ou evolutiva. Assim, com uma só metodologia, uma empresa com um cenário como o descrito nesta dissertação pode usar o XPOKs.

## 6.2 Validação

Esta secção descreve a simulação da aplicação da metodologia XPOKs a um cenário de desenvolvimento de um projeto, partindo de um pedido de um cliente até à entrega final da solução. A equipa do projeto escolhida deve ajustar-se à realidade e considerar as diversas caraterísticas e valências dos elementos.

### 6.2.1 Implementação do XPOKs em Projeto Piloto

Para proceder à implementação da metodologia, foi constituída uma equipa homogénea composta por um elemento de qualidade QA, que procede com testes há cerca de 7 anos e que sempre utilizou Scrum; um *team leader* TL, que trabalha com Scrum há 8 anos; um TL há cerca de 3 anos; um *developer* sénior com 5 anos de experiência que sempre trabalhou com Outsystems e um *developer* júnior, com 1 ano de experiência e que utiliza Scrum.

Para criar o desafio, foi pedido ao elemento da equipa, o analista de sistemas, para assumir o papel de cliente final e criar uma especificação a entregar à equipa de desenvolvimento (que pode ser consultada no Anexo 4). Descartam-se todos os processos de negociação de contratos entre empresas, uma vez estarem fora do âmbito desta dissertação.

A disponibilidade conseguida, para a implementação do XPOKs, foi de oito dias, com oito horas de trabalho em cada um. Desta forma, cada *sprint* passou a ser de dois dias e um dia de trabalho passou a ser representado por quatro horas, sendo implementados todos os eventos que a metodologia indica.

Após a chegada do pedido do projeto, o elemento da equipa, que assume os papéis de TL e *Engagement Manager* EM, tratou de marcar o *kick-off* do projeto, para iniciar os trabalhos, de onde resultou o *backlog* e a visão do produto, tal como é possível verificar na Figura 6-9.

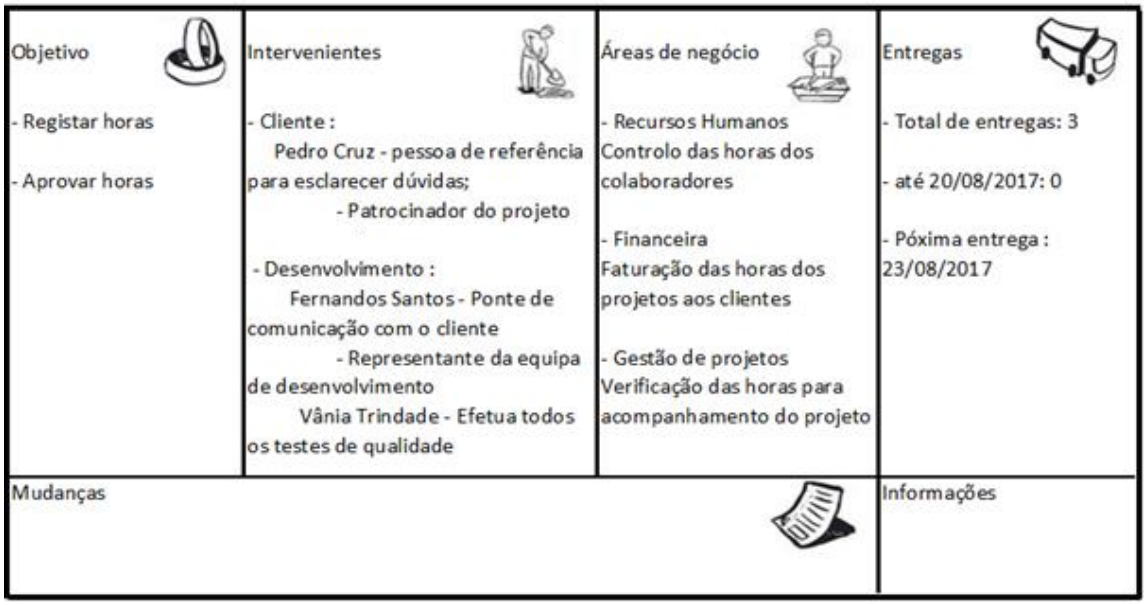


**Figura 6-9: Visão do projeto - Gestão de Horas**

Neste primeiro *sprint*, a equipa de desenvolvimento realizou a arquitetura do projeto, de acordo com a Figura 6-10. Todos os elementos participaram e definiram o que deve ser feito em cada camada, indo ao encontro dos padrões e propriedade coletiva defendida pelo XPOKs.

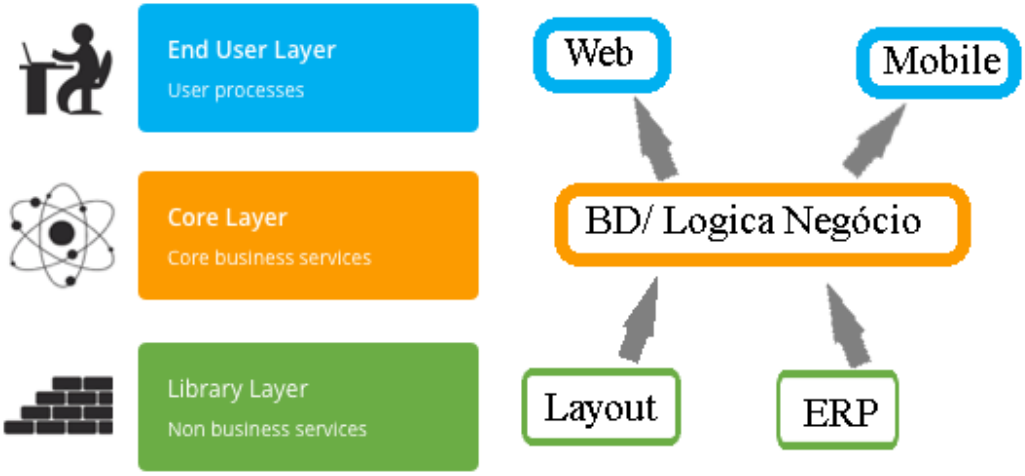


**Figura 6-10: Modelo de arquitetura**

As reuniões de início de *sprint* seguem sempre o mesmo padrão. A equipa de desenvolvimento, em conjunto com o cliente, determinam quais os itens do *backlog* fazem parte do *sprint*, criando o *backlog* do *sprint*, atendendo a constrangimentos relacionados com a capacidade e disponibilidade da equipa. Além disso, todos os *itens* são discutidos com a equipa, de forma a detalhar todos os aspetos relacionados com a execução, conseguindo ter, em cada um, o conceito de *definition of done*, ou seja, o que é necessário estar implementado para considerar um item como fechado.

No decorrer do *sprint*, no início de cada dia, é realizada a reunião diária, onde a equipa efetua o ponto de situação, permitindo assim ao EM proceder ao controlo do progresso do *sprint* para constatar eventuais atrasos. Nesta etapa, a presença do cliente permite esclarecer pequenas dúvidas que possam existir nos itens a decorrer.

No decorrer do *sprint*, a equipa de desenvolvimento optou por implementar um quadro simplificado da visão do *sprint* para conter itens comuns, ou seja, existiam itens que eram partilhados por outros itens. Portanto, quando alguém sentiu a necessidade de implementar um item, toda a equipa olhava para o quadro e percebia o que já tinha sido feito. Na Figura 6-11 é possível identificar o quadro com três itens, dos quais um já se encontra no estado "em curso".

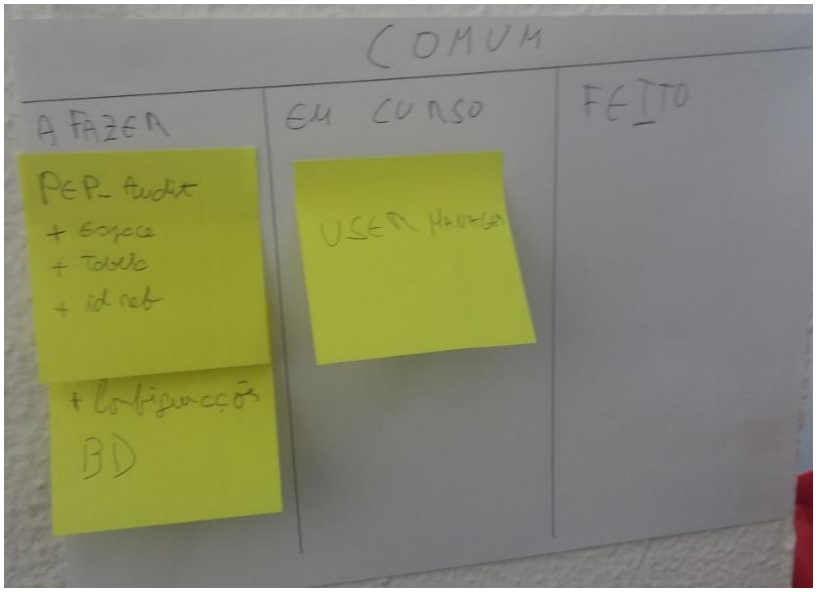


**Figura 6-11: Quadro comum de itens**

O fecho de cada *sprint* inclui a *demo* ao cliente e a sua retrospetiva. A *demo* permite demonstrar todo o trabalho realizado e receber o *feedback* do cliente relativamente ao trabalho executado. Neste momento, o cliente tem ainda a oportunidade de efetuar novos pedidos de melhoria, os quais são incluídos no *backlog*. A análise retrospetiva ao projeto possibilita à equipa adaptar-se a uma nova metodologia, para aprender com os erros anteriores.

O fecho do projeto, segundo o XPOKs, prevê um *sprint* de estabilização e um de formação, unificados num único, por força da limitação do tempo existente. Assim, enquanto foi fornecida formação sobre a aplicação ao cliente, também foram realizados os últimos pedidos de melhorias.

### 6.2.2 Análise da Implementação do Modelo

Para a análise e avaliação da metodologia proposta recorreu-se a um questionário, que pode ser consultado no Anexo 5, respondido pelos elementos da equipa do projeto, envolvidos no teste piloto. O objetivo foi validar se a metodologia XPOKs é útil na gestão de projetos de *software*, relacionado com a implementação de práticas de trabalho, assim como no reforço da relação entre o cliente e a equipa de desenvolvimento.

A construção do questionário baseou-se no trabalho de (Al-Zewairi *et al.*, 2017), (Callegari *et al.*, 2008) e (Azizyan, Magarian e Kajko-Matsson, 2011) que também necessitaram de avaliar metodologias. Porém, as perguntas foram adaptadas à realidade do XPOKs. É utilizada uma escala de 0 a 5 para quantificar a resposta a cada questão, na qual 0 é pouco e 5 é excelente.

O primeiro grupo de perguntas, cujas respostas se apresentam na Tabela 6-1, tem como objetivo validar, de forma genérica, se a metodologia ajuda na produtividade da equipa e, consequentemente, na melhoria da qualidade, para uma maior satisfação do cliente. Pode-se concluir que o melhor resultado se obteve na resposta à pergunta: "O desenvolvimento com XPOKs torna o trabalho mais organizado/planeado?", que alcançou uma média de 4,5, mostrando que é fundamental que todos os elementos da equipa devam ser conhecedores do que está a acontecer no projeto. Já nas outras questões, observou-se uma média igual ou superior a 4, demonstrando que com o XPOKs foi possível diminuir o número de erros e aumentar a satisfação do cliente, uma vez que o elemento de QA atribui a nota de 5.

**Tabela 6-1: Questionário – Avaliação genérica**

| Questões | QA | TL | *Developer* Júnior | *Developer* Sénior | Média |
|---|---|---|---|---|---|
| O desenvolvimento com XPOKs aumentou a eficácia do desenvolvimento? | 3 | 4 | 5 | 4 | 4 |
| O desenvolvimento com XPOKs aumentou a qualidade do produto? | 4 | 4 | 5 | 4 | 4,25 |
| O desenvolvimento com XPOKs aumenta a colaboração? | 3 | 4 | 5 | 4 | 4 |
| O desenvolvimento com XPOKs torna o trabalho mais organizado/ planeado? | 4 | 5 | 4 | 5 | 4,5 |
| O desenvolvimento com XPOKs permite a deteção precoce de erros / defeitos? | 5 | 4 | 4 | 3 | 4 |
| Média | 3,8 | 4,2 | 4,6 | 4 | 4,15 |

O segundo grupo de perguntas pretende compreender se a divisão da metodologia em início, ciclos de desenvolvimento e fecho foi bem aceite, assim como os seus artefactos e as suas práticas. Da análise da Tabela 6-2, pode concluir-se que a divisão de um projeto em três fases foi bem aceite, uma vez que contou com nota máxima por parte de todos os elementos. Já as restantes questões obtiveram todas médias superiores a 4, indicando que as práticas sugeridas pelo XPOKs contribuíram para o sucesso do projeto.

Relativamente à questão "Práticas da primeira fase, início do ciclo?", podemos constatar, na Tabela 6-2, que a utilização do termo de abertura, que obteve 5 de média, sendo aquela que reuniu maior consenso. Tal, pode dever-se a que o termo de abertura seja um documento mais simplificado que o sugerido pelo PMBOK, e ajuda toda a equipa a entender o objetivo do projeto. A sugestão do modelo de arquitetura reuniu somente a nota máxima pelos elementos que efetuam a codificação, enquanto o elemento encarregado dos testes não lhe atribuiu tanta importância, apenas lhe concedeu a classificação de 3. Uma das possíveis razões é a equipa de desenvolvimento não considerar este artefacto nas metodologias que utiliza.

Dos resultados apresentados na Tabela 6-2, também se pode constatar que, durante os ciclos de desenvolvimento, a reunião diária com o cliente e o *pair programming* foram as que reuniram melhores avaliações, com uma média de 4,5, contribuindo para esse resultado uma boa avaliação por parte do elemento de qualidade. O pior resultado foi alcançado com a documentação, que obteve uma média de 3,75, não sendo pior, pois o elemento responsável pela qualidade atribuiu nota máxima. Uma das justificações possíveis é a documentação representar um esforço adicional em comparação a outras metodologias ágeis como Scrum.

A fase final do projeto teve uma média de 4,25, como podemos constatar na Tabela 6-2, contribuindo para este resultado a classificação de 4,5 da estabilização e da ausência de planeamento, embora o elemento de qualidade atribua a sua pior avaliação ao facto de não existir planeamento, o que de certa forma afeta o seu trabalho.

**Tabela 6-2: Questionário – Fases do XPOKs**

| Questões | QA | TL | *Developer* Júnior | *Developer* Sénior | Média |
|---|---|---|---|---|---|
| Considera fundamental a divisão do projeto em 3 fases? | 5 | 5 | 5 | 5 | 5 |
| Práticas da primeira fase, início do ciclo? | 4 | 4,75 | 4,5 | 4,5 | 4,43 |
| Termo de abertura | 5 | 5 | 5 | 5 | 5 |

| Questões | | QA | TL | *Developer* Júnior | *Developer* Sénior | Média |
|---|---|---|---|---|---|---|
| | Visão do produto | 4 | 4 | 5 | 5 | 4,5 |
| | *Product Backlog* | 4 | 5 | 3 | 3 | 3,75 |
| | Modelo de arquitetura | 3 | 5 | 5 | 5 | 4,5 |
| Práticas dos ciclos de desenvolvimento? | | 4,33 | 4,16 | 4,33 | 4,16 | 4,25 |
| | Documentação | 5 | 4 | 3 | 3 | 3,75 |
| | Reuniões diárias com o cliente | 5 | 4 | 5 | 4 | 4,5 |
| | Visão do *sprint* | 4 | 4 | 5 | 4 | 4,25 |
| | *Pair programming* | 4 | 4 | 5 | 5 | 4,5 |
| | *Refactoring* | 3 | 5 | 4 | 5 | 4,25 |
| | Integrações contínuas | 5 | 4 | 4 | 4 | 4,25 |
| Práticas do fecho do ciclo? | | 4,33 | 4,66 | 4 | 4 | 4,25 |
| | Formação | 5 | 4 | 3 | 3 | 3,75 |
| | Estabilização | 5 | 5 | 4 | 4 | 4,5 |
| | Ausência de planeamento | 3 | 5 | 5 | 5 | 4,5 |
| | Média | 4,28 | 4,28 | 4,35 | 4,28 | 4,35 |

A fase final do questionário teve como objetivo perceber se os elementos do projeto voltariam a usar esta metodologia nos seus projetos. Foi possível averiguar que se tivessem a responsabilidade de escolher uma metodologia, provavelmente optariam por esta metodologia. Já para apresentarem a outros colegas das suas empresas, notou-se que não se sentiam tão confortáveis, já que a média foi de 3,5, uma vez que os elementos *developers* deram apenas nota 3, como podemos constatar na Tabela 6-3.

**Tabela 6-3: Questionário – Ciclos do XPOKs**

| Questões | QA | TL | *Developer* Júnior | *Developer* Sénior | Média |
|---|---|---|---|---|---|
| Se pudesse escolher uma metodologia para o seu projeto, optava pelo XPOKs? | 4 | 4 | 4 | 4 | 4 |
| Uma vez que o XPOKs ainda não foi aplicado a um número significativo de projetos, apresentaria hoje aos seus superiores para aplicarem em trabalhos futuros? | 4 | 4 | 3 | 3 | 3,5 |

Verificando todos os resultados obtidos no questionário e apresentados anteriormente, pode-se verificar que a média é superior a 4 numa escala de 0 a 5, comprovando que a metodologia foi bem recebida por todos os elementos, ainda que no questionário se tenha observado uma tendência para avaliar melhor as áreas que favorecem cada um dos elementos, como podemos verificar por exemplo nos valores atribuídos ao *Refactoring*, onde o elemento de qualidade atribui 3 e os elementos de desenvolvimentos 4 e 5.

### 6.2.3 Comparação XPOKs *Versus* PMBOK

Nesta secção realiza-se uma comparação entre o PMBOK e o XPOKs, recorrendo aos critérios utilizados na comparação efetuada na secção 5.4 (a tabela comparativa pode ser consultada no Anexo 6).

Tal como se pode observar na Figura 6-12, o XPOKs cobre cerca de 50% dos processos do PMBOK, sendo o grupo de processos de planeamento aquele onde se constata uma menor taxa de cobertura, pois as metodologias ágeis são direcionadas para as mudanças constantes e não atendem a um planeamento total do projeto.

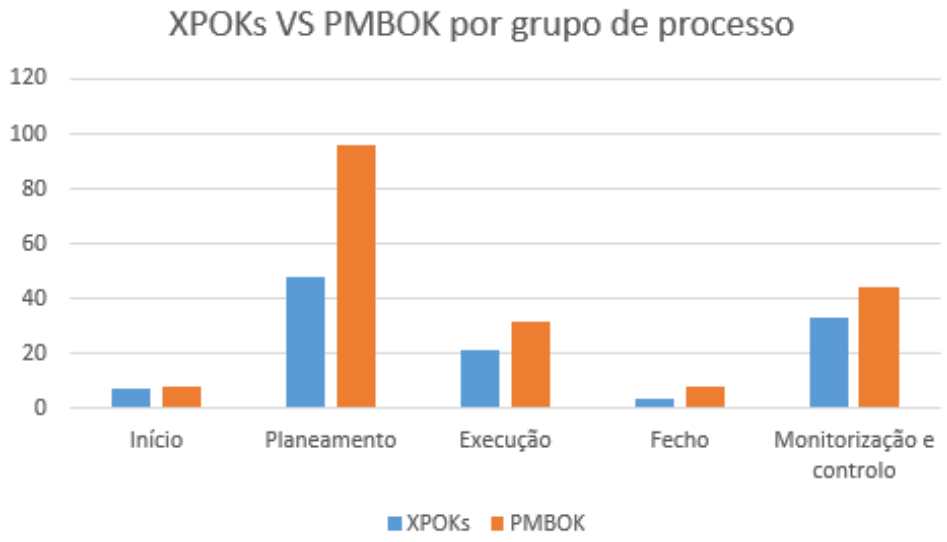


**Figura 6-12: XPOKs vs PMBOK por grupo de processo**

Foi realizada a análise por área de conhecimento, sendo os resultados exibidos na Figura 6-13. À semelhança dos resultados obtidos anteriormente na comparação com o Scrum, as áreas de custos e recursos não têm nenhuma equivalência na metodologia XPOKs. Por sua vez, os *stakeholders* são muito idênticos, uma vez que o XPOKs à semelhança das metodologias ágeis depende da interação entre a equipa de desenvolvimento e o cliente.

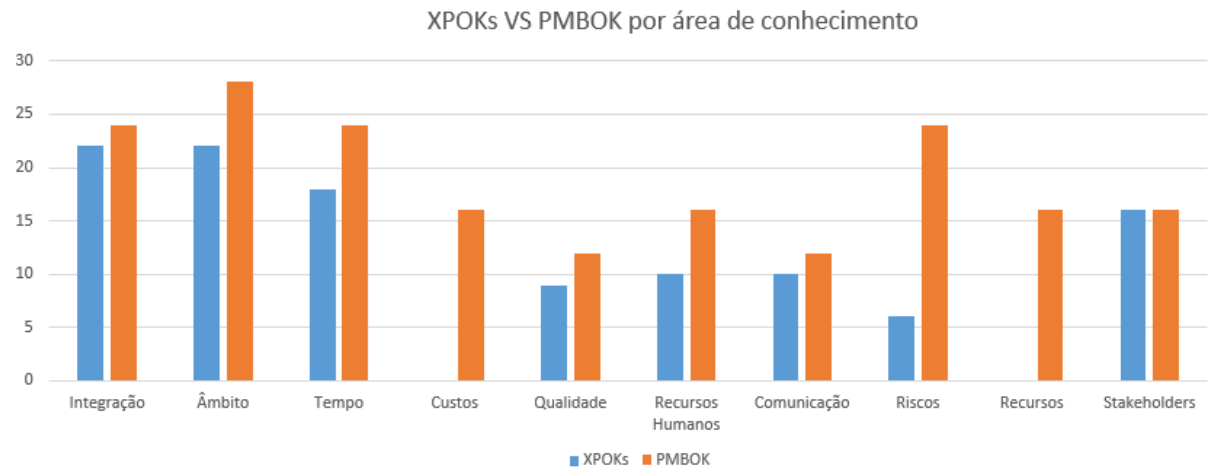


**Figura 6-13: XPOKs vs PMBOK por área de conhecimento**

Da comparação da taxa de cobertura de processos da área de conhecimento da gestão dos riscos, do XPOKs e PMBOK, resulta que esta permanece baixa. Daqui depreende-se que o XPOKs dispõe de poucas ferramentas na gestão de riscos. No entanto, à semelhança das metodologias ágeis descritas no capítulo 4, opta-se por uma abordagem que inclui um envolvimento constante do cliente em todo o projeto, mesmo nas reuniões diárias.

Uma vez que as áreas de conhecimento de custos e recursos não foram consideradas nesta metodologia ágil, resulta um número menor de processos de planeamento, quando comparado com o PMBOK. Neste âmbito, procedeu-se à análise sem estes processos, para compreender que diferenças existem (a tabela comparativa pode ser consultada no Anexo 6).

Na Figura 6-14 podemos apurar que o XPOKs cobre cerca de 85% dos processos do PMBOK, com a configuração referida anteriormente. Os valores poderiam ser mais expressivos se a gestão dos recursos humanos, no processo de aquisição da equipa, fosse considerada pelo XPOKs, mas à semelhança de outras metodologias ágeis, este processo não foi previsto.

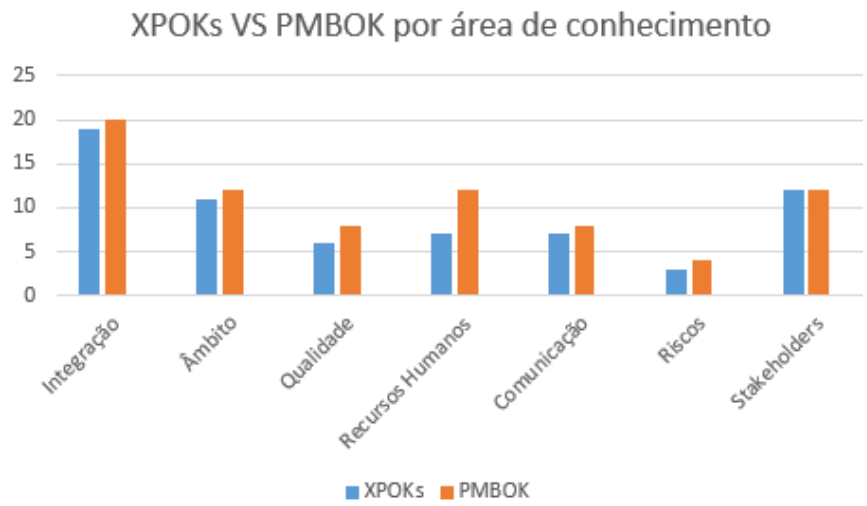


**Figura 6-14: XPOKs vs PMBOK por área de conhecimento sem o grupo de planeamento e áreas de conhecimento - custos e recursos**

## 6.3 Auto-crítica ao XPOKs

Para a criação da metodologia híbrida XPOKs foi essencial não valorizar o conhecimento adquirido com a prática profissional e analisar a literatura sem juízos de valor prévios. Por isso, esteve sempre presente a filosofia de Miyamoto Musashi "*In all things, have no preference.*" (Miyamoto, 1992), ou seja, aplicando a esta dissertação, não se deveria estar focado só em Scrum, mas alargar horizontes e considerar aspetos importantes noutras abordagens na gestão de projetos para colmatar falhas nos processos usados.

A metodologia híbrida sugerida encontra-se suportada, em grande medida, em metodologias ágeis, para manter a essência ágil, colmatando as suas lacunas através do recurso a algumas das melhores práticas do PMBOK, tal como o termo de abertura e o grupo de processos para organizar o XPOKs.

O XPOKs passou a valorizar mais:

- A fase inicial – O início de um projeto é fundamental para o seu sucesso, por isso, dedicou-se a ele um *sprint*. Nesta fase, é privilegiada a aproximação entre a equipa de desenvolvimento e o cliente, para que se possa efetuar a especificação do projeto da melhor forma possível, produzindo o termo de abertura;

- Documentação – Embora os clientes não requeiram uma documentação extensa, quase todos os eventos do XPOKs passaram a ter um documento como, por exemplo, a visão do *sprint.* Nos itens do *Backlog,* é recomendado haver a *definition of done* e mesmo a especificação técnica, ajudando assim a equipa de desenvolvimento a justificar as opções tomadas na implementação;

- Práticas de desenvolvimento usadas pela equipa de desenvolvimento – Foram aplicadas quase todas as práticas recomendadas pela metodologia XP, como por exemplo o modelo de arquitetura, por ajudarem as equipas na sua organização diária e no desenvolvimento de um projeto;

- Fim do projeto – Para auxiliar no fecho de um projeto, o XPOKs considerou a fase de fecho. Inicialmente, procede-se com à estabilização do projeto, onde é possível corrigir ou melhorar as soluções criadas. Posteriormente, é transferido o conhecimento técnico ao cliente.

Algumas das práticas recomendadas pelo PMBOK que não foram contempladas:

- Escolha das equipas de projeto – A escolha da equipa de projeto atualmente é efetuada pelos responsáveis do departamento, tendo em consideração a alocação dos seus elementos aos projetos a decorrer. Pode não ser a melhor solução, já que podemos estar a colocar pessoas num projeto numa área que não é da sua preferência. No entanto, uma empresa também não pode ter os seus colaboradores a aguardar pelo "projeto ideal".

- Gestão do risco – A abordagem para a gestão do risco no XPOKs segue abordagens de metodologias ágeis, aproximando o cliente da equipa de desenvolvimento e mantendo uma comunicação, preferencialmente diária, em vez de optarem por uma análise formal dos riscos. O facto de as metodologias ágeis incluírem a presença do cliente durante o projeto tem contribuído para o sucesso dos projetos;

- Planeamento - O XPOKs atribui uma maior importância ao planeamento, quando comparado com as metodologias ágeis, ao ter um *sprint* inicial dedicado à definição, embora não muito detalhado.

Para a validação da metodologia procedeu-se à implementação de um projeto piloto, envolvendo uma equipa de quatro pessoas, as quais já recorriam de metodologias ágeis nos ambientes onde se encontram inseridos. O facto de todos os elementos da equipa, disporem apenas de experiência com metodologias ágeis, pode revelar-se negativo. Porém a realidade, para a qual foi desenvolvido o XPOKs, já só usa abordagens ágeis.

De forma a avaliar o XPOKs, a equipa aceitou responder a um questionário, com o qual se pretende perceber se a metodologia é adequada na implementação de projetos de *software.* Contudo, requer uma validação mais ampla, com a sua aplicação a um maior número de projetos, para verificar se os resultados são também satisfatórios.

Uma vez que PMBOK reúne um corpo de conhecimento, genericamente aceite, na área de gestão de projetos, levou-se a efeito uma comparação com o XPOKs. Porém, os resultados da comparação podem ser sempre questionáveis, pois depende muito da subjetividade do autor que a efetua.

Em suma, o modelo XPOKs revelou os benefícios que advêm da compatibilização dos modelos ágeis com as práticas mais convencionais, sendo especialmente importante a inclusão do processo de abertura do projeto. Este foi o ponto que reuniu mais unanimidade no que respeita à sua utilização.

## 6.4 Sumário

Neste capítulo, foi exposta a proposta de uma nova metodologia para a gestão de projetos de *software*, o XPOKs. Este é composto por três fases, compreendendo o início, ciclos de desenvolvimento e fecho, que resulta fundamentalmente, do cruzamento de metodologias ágeis existentes e ainda do PMBOK.

No próximo capítulo será realizada uma reflexão sobre toda a dissertação, e realizada uma avaliação relativamente ao cumprimento dos objetivos e apontar novos caminhos para melhoria da metodologia XPOKs, tal como por exemplo a inclusão de novos artefactos.

# 7. Conclusões e Trabalhos Futuros

Ao longo dos anos, o desenvolvimento de *software* tem evoluído a par das metodologias que usamos na gestão dos nossos projetos. As metodologias ágeis adquiriram bastante popularidade, devido aos seus princípios, práticas e ao facto de promoverem uma comunicação constante entre o cliente e as equipas de desenvolvimento. Além das metodologias ágeis, existem também outros métodos considerados como os tradicionais, tal como o modelo em cascata. Em resultado dos problemas enfrentados na organização onde o autor exerce a sua atividade profissional, surgiu o desafio de estudar, mais detalhadamente, as diversas metodologias para definir uma metodologia híbrida que se mantivesse ágil, para colmatar o défice de documentação nos projetos, a falta de práticas de trabalho das equipas de desenvolvimento e a inexistência da definição do âmbito do projeto.

O ponto de partida consistiu em compreender quais as metodologias mais usadas, constatando-se que o método ágil mais usado é o Scrum, a par de alguns métodos híbridos suportados no Scrum. Uma vez que as metodologias ágeis têm alguns pontos fracos, como a definição dos requisitos, somente no *sprint* foi utilizado o PMBOK para ultrapassar essas dificuldades.

O aprofundamento do estudo de diversas metodologias permitiu efetuar comparações entre elas, possibilitando a outros profissionais, com este estudo, adotarem uma metodologia que vá ao encontro das suas necessidades. Uma vez que as metodologias estudadas não permitiram ultrapassar as necessidades da empresa, existiu a necessidade de ser criada uma nova metodologia que continua-se a ser ágil. Portanto foi criada a metodologia XPOKs, que combina

diversas metodologias ágeis como Kanban, Outsystems, Scrum, XP e algumas boas práticas recomendadas pelo PMBOK.

O XPOKs assenta o seu princípio de funcionamento em três fases - início, ciclos de desenvolvimento e fecho do projeto. O início do projeto é uma fase crucial pois é definido o âmbito do projeto, e deve ser criado o documento de visão do produto. Os ciclos de desenvolvimento são compostos por diversos *sprints*, onde os *developers* procedem com a codificação das histórias, devendo estas conter a *definition of done* e mesmo a especificação técnica, privilegiando assim a documentação. O fecho do projeto é composto por dois *sprints*, sendo o primeiro dedicado à estabilização e o segundo à formação do cliente. O XPOKs implementa também, práticas de desenvolvimento a serem usadas pela equipa de desenvolvimento, aplicando quase todas as práticas recomendadas pelo XP, sendo que a utilização de *pair programming*, foi uma das que reuniu maior consenso no questionário realizado para validar o XPOKs.

Embora o XPOKs não seja ainda a metodologia de referência da organização onde decorreu este trabalho, começam a existir evidências da sua adoção. Os projetos que usavam a metodologia Kanban passaram a utilizar práticas como *refactoring* e *pair programming*.

Atualmente, quando um projeto é ganho, é notória a preocupação com aspetos relacionados com o estabelecimento do seu âmbito. Este facto vai ao encontro do que o XPOKs defende, para o qual os projetos devem ter um ciclo inicial, onde é produzido um artefacto de extrema importância - o termo de abertura.

As práticas de desenvolvimento defendidas pelo XPOKs, particularmente as que foram inspiradas pela abordagem XP, começam a ser utilizadas por todas as equipas de desenvolvimento, salientando-se a utilização de padrões, facilitando a integração de novos elementos, a propriedade coletiva, algo que motivou os elementos, tanto os que efetuam a codificação, como aqueles que executam os testes, a sentirem o projeto como seu. Este aspeto contribui para o trabalho em conjunto, para encontrarem a melhor solução, existindo assim um compromisso mútuo.

Embora a metodologia XPOKs ainda não se encontre implementada na totalidade na organização, a forma de trabalho atual da empresa, aproxima-se do XPOKs. É assim

estimulanda a melhoria contínua dos processos, começando com o que já existe, indo ao encontro de um dos princípios que o XPOKs adotou do Kanban.

A metodologia XPOKs pode ainda evoluir de forma a incluir uma lista de recomendações para proceder a testes, assim como sugerir uma métrica para estimar os itens do *backlog* (se em horas ou *story points*), sendo *story points* a técnica mais usada no desenvolvimento ágil (Coelho e Basu, 2012). Além disso, os artefactos do XPOKs, eventos, práticas, papéis e responsabilidades podem ser revistos, para que a metodologia proposta se torne mais robusta e se adapte a um maior número de cenários.

Uma das maiores limitações deste estudo resulta das condições em que é realizada a prova de conceito, devido ao tempo reduzido que foi possível ter uma equipa a aplicar a metodologia. Este facto condicionou também o âmbito do projeto, pelo que os dados apresentados devem ser considerados como uma primeira aproximação e não uma conclusão final. Logo é importante fazer um estudo do XPOKs aplicado numa equipa, num período de tempo suficiente, para se perceber se existe um aumento de qualidade, satisfação do cliente e aumento da qualidade das entregas.

Será também importante, estudar uma prática de engenharia de *software* chamada DevOps, que se popularizou em 2009 na Bélgica (*devopsdays*, 2009), que promove a proximidade entre as equipas de desenvolvimento e as operações de IT. Uma vez que o XPOKs não contempla a área de operações pode ser verificado o esforço para a obter.

No decorrer desta dissertação recorreu-se aos relatórios produzidos pela empresa Versionone, para perceber qual a realidade da utilização de metodologias ágeis. Neste sentido, sugere-se a realização de um estudo em Portugal sobre o estado da gestão de projetos de *software*, para tentar compreender o real alcance na utilização de metodologias ágeis e também perceber se outras organizações optam por criar os seus métodos híbridos.

# REFERÊNCIAS


10 Kanban Board Examples | LeanKit - [Em linha], atual. 2017. [Consult. 16 mar. 2018]. Disponível em WWW:<URL:https://leankit.com/learn/kanban/kanban-board-examples-for-development-and-operations/>.

ABRAHAMSSON, Pekka *et al.* - Agile Software Development Methods: Review and Analysis. 2002. VTT Publications: Finland. 3:2002).

ABRAHAMSSON, Pekka *et al.* - New directions on agile methods: a comparative analysis. Em Software Engineering, 2003. Proceedings. 25th International Conference on. [S.l.] : Ieee, 2003

ADJEI, Daniel; RWAKATIWANA, Peter - Application of Traditional and Agile Project Management in Consulting Firms.: A Case Study of PricewaterhouseCoopers

AGILE - Manifesto for Agile Software Development [Em linha], atual. 2001. [Consult. 15 mar. 2018]. Disponível em WWW:<URL:http://agilemanifesto.org/>.

AgileBusiness - [Em linha], atual. 2017. [Consult. 24 mar. 2017]. Disponível em WWW:<URL:https://www.agilebusiness.org/business-agility>.

AITKEN, Ashley; ILANGO, Vishnu - A comparative analysis of traditional software engineering and agile software development. Em System Sciences (HICSS), 2013 46th Hawaii International Conference on. [S.l.] : IEEE, 2013

ALIPUI, Gilbert *et al.* - An Agile Approach to Doctoral Research Dissertation. Proceedings of Student-Faculty Research Day, CSIS, Pace University. 2014) D2.

ALMEIDA, Mauro Jorge Pereira De - Classificaçao e Comparaçao de MétodosAgeis de Desenvolvimento de Software. 2008).

ALSHAMRANI, Adel; BAHATTAB, Abdullah - A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. International Journal of Computer Science Issues (IJCSI). 12:1 (2015) 106.

AL-ZEWAIRI, Malek *et al.* - Agile software development methodologies: survey of surveys. Journal of Computer and Communications. 5:05 (2017) 74.

ANAND, R. Vijay; DINAKARAN, M. - Popular Agile Methods in Software Development: Review and Analysis. International Journal of Applied Engineering Research. 11:5 (2016) 3433–3437.

ANDERSON, David J. - Kanban: successful evolutionary change for your technology business. [S.l.] : Blue Hole Press, 2010

AVISON, David; FITZGERALD, Guy - Information systems development: methodologies, techniques and tools. [S.l.] : McGraw Hill, 2003

AZIZYAN, Gayane; MAGARIAN, Miganoush Katrin; KAJKO-MATSSON, Mira - Survey of agile tool usage and needs. Em Agile Conference (AGILE), 2011. [S.l.] : IEEE, 2011

BATRA, Dinesh *et al.* - Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. CAIS. 27:2010) 21.

BECK, Kent - Embracing change with extreme programming. Computer. 32:10 (1999) 70–77.

- Extreme programming explained: embrace change. [S.l.] : addison-wesley professional, 2000

BECK, Kent; ANDRES, Cynthia - Extreme Programming Explained: Embrace Change, (The XP Series). 2004).

BEGEL, Andrew; NAGAPPAN, Nachiappan - Usage and perceptions of agile software development in an industrial context: An exploratory study. Em Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on. [S.l.] : IEEE, 2007

BELL, Thomas E.; THAYER, Thomas A. - Software requirements: Are they really a problem? Em Proceedings of the 2nd international conference on Software engineering. [S.l.] : IEEE Computer Society Press, 1976

BOEG, Jesper - Kanban em 10 passos. Tradução de Leonardo Campos, Marcelo Costa, Lúcio Camilo, Rafael Buzon, Paulo Rebelo, Eric Fer, Ivo La Puma, Leonardo Galvão, Thiago Vespa, Manoel Pimentel e Daniel Wildt. C4Media. 2010).

BOEHM, Barry; TURNER, Richard - Balancing agility and discipline: A guide for the perplexed. [S.l.] : Addison-Wesley Professional, 2003

BOMFIN, David Ferreira; ÁVILA NUNES, Paula Cristine DE; HASTENREITER, Flávio - Gerenciamento de projetos segundo o guia PMBOK: desafios para os gestores. Revista de Gestão e Projetos-GeP. 3:3 (2012) 58–87.

CALLEGARI, Daniel Antonio *et al.* - An Integrated Model for Managerial and Productive Activities in Software Development. Em ICEIS (1)

CHARVAT, Jason - Project management methodologies: selecting, implementing, and supporting methodologies and processes for projects. [S.l.] : John Wiley & Sons, 2003

CHIN, CMM; SPOWAGE, AC - Defining & classifying project management methodologies. PM World Today. 12:5 (2010) 1–9.

COAD, Peter; LUCA, Jeff De; LEFEBVRE, Eric - Java modeling color with UML: Enterprise components and process with Cdrom. [S.l.] : Prentice Hall PTR, 1999

COCKBURN, Alistair - Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams. [S.l.] : Pearson Education, 2004

COELHO, Evita; BASU, Anirban - Effort estimation in agile software development using story points. International Journal of Applied Information Systems (IJAIS). 3:7 (2012).

COHN, Mike - Agile estimating and planning. [S.l.] : Pearson Education, 2005

DAI, Christine Xiaoyi; WELLS, William G. - An exploration of project management office features and their relationship to project performance. International Journal of Project Management. 22:7 (2004) 523–532.

devopsdays - [Em linha], atual. 2009. [Consult. 1 out. 2017]. Disponível em WWW:<URL:https://www.devopsdays.org/>.

Examining the Agile Manifesto - [Em linha], atual. 2017. [Consult. 15 mar. 2018]. Disponível em WWW:<URL:http://www.ambysoft.com/essays/agileManifesto.html>.

FAHAD, Muhammad *et al.* - A Comparative Analysis of DXPRUM and DSDM. IJCSNS. 17:5 (2017) 259.

FERREIRA, João Carlos Loureiro Mendes - Abordagens AGILE na gestão de projectos. 2013.

FOWLER, Martin - The new methodology. Wuhan University Journal of Natural Sciences. 6:1–2 (2001) 12–24.

FREITAS, João Capucho Correia De - Metodologias de planeamento-Análise comparativa. [S.l.] : ISA/UL, 2015 PhD Thesis.

GIDO, Jack; CLEMENTS, James P. - Successful Project Management (with Microsoft Project and InfoTrac). [S.l.] : South-Western College Publishing, 2008

GLAIEL, Firas S.; MOULTON, Allen; MADNICK, Stuart E. - Agile project dynamics: A system dynamics investigation of agile software development methods. 2014).

GLAZER, Hillel *et al.* - CMMI or agile: why not embrace both! 2008).

GROSS, John M.; MCINNIS, Kenneth R. - Kanban made simple: demystifying and applying Toyota's legendary manufacturing process. [S.l.] : AMACOM Div American Mgmt Assn, 2003

HASTIE, Shane; WOJEWODA, Stéphane - Standish group 2015 chaos report-q&a with Jennifer Lynch [Em linha], atual. 2015. [Consult. 7 mar. 2017]. Disponível em WWW:<URL:https://www.infoq.com/articles/standish-chaos2015>.

HAYATA, Tomohiro; HAN, Jianchao - A hybrid model for IT project with Scrum. Em Service Operations, Logistics, and Informatics (SOLI), 2011 IEEE International Conference on. [S.l.] : IEEE, 2011

HIGHSMITH, Jim - Adaptive software development: a collaborative approach to managing complex systems. [S.l.] : Addison-Wesley, 2013

HIGHSMITH, Jim; COCKBURN, Alistair - Agile software development: The business of innovation. Computer. 34:9 (2001) 120–127.

HIWARKAR, Kiran; DOSHI, Aditya; CHINTA, Rahul - Comparative Analysis of Agile Software Development Methodologies-A Review. Int. Journal of Engineering Research and Applications. . ISSN 2248-9622. 6:3 (2016) 80–85.

INSTITUTE, Project Management - A Guide to the Project Management Body of Knowledge: PMBOK GuidePMBOK® Guide Series. [Em linha]. [S.l.] : Project Management Institute, 2013 Disponível em WWW:<URL:https://books.google.pt/books?id=FpatMQEACAAJ>. ISBN 978-1-935589-67-9.

ISO/IEC; SOCIETY, IEEE Computer - International Standard ISO/IEC 12207: systems and software engineering-software life cycle processes

JAMALI, Gholamreza; OVEISI, Mina - A Study on Project Management Based on PMBOK and PRINCE2. Modern Applied Science. 10:6 (2016) 142.

JEFFRIES, Ron - What is Extreme Programming? [Em linha], atual. 2011. [Consult. 16 mar. 2018]. Disponível em WWW:<URL:https://ronjeffries.com/xprog/what-is-extreme-programming/>.

JEFFRIES, Ron; ANDERSON, Ann; HENDRICKSON, Chet - Extreme programming installed. [S.l.] : Addison-Wesley Professional, 2001

KARAMAN, Ersin; KURT, Murat - Comparison of project management methodologies: prince 2 versus PMBOK for it projects. Int. Journal of Applied Sciences and Engineering Research. 4:5 (2015) 657–664.

KERZNER, Harold - Strategic planning for project management using a project management maturity model. [S.l.] : John Wiley & Sons, 2002

KNIBERG, Henrik - Scrum e XP direto das Trincheiras. Estocolmo: C4Media Inc. 2015).

KUMAR, Gaurav; BHATIA, Pradeep Kumar - Impact of Agile methodology on software development process. International Journal of Computer Technology and Electronics Engineering (IJCTEE). 2:4 (2012).

LANDRY, Jeffrey; MCDANIEL, Rachel - Agile Preparation Within a Traditional Project Management Course. Information Systems Education Journal. 14:6 (2016) 27.

LARMAN, Craig; BASILI, Victor R. - Iterative and incremental developments. a brief history. Computer. 36:6 (2003) 47–56.

LEAU, Yu Beng *et al.* - Software development life cycle AGILE vs traditional approaches. Em International Conference on Information and Network Technology

LEI, Howard *et al.* - A statistical analysis of the effects of Scrum and Kanban on software development projects. Robotics and Computer-Integrated Manufacturing. 43:2017) 59–67.

LINDSTROM, Lowell; JEFFRIES, Ron - Extreme programming and agile software development methodologies. Information systems management. 21:3 (2004) 41–52.

MAREK, Richard P.; ELKINS, Debra A.; SMITH, Donald R. - Manufacturing controls: understanding the fundamentals of Kanban and CONWIP pull systems using simulation. Em Proceedings of the 33nd conference on Winter simulation. [S.l.] : IEEE Computer Society, 2001

MARKS, Adam - Information Systems Development Methodolgies in Developing Higher Education. Creative Education. 3:01 (2012) 114.

MATOS, Sandra; LOPES, Eurico - Prince2 or PMBOK–a question of choice. Procedia Technology. 9:2013) 787–794.

MCLAUGHLIN, Mike - Agile Methodologies for Software Development [Em linha], atual. 2016. [Consult. 6 mar. 2017]. Disponível em WWW:<URL:https://www.versionone.com/agile-101/agile-methodologies/>.

MEREDITH, Jack R.; MANTEL JR, Samuel J. - Project management: a managerial approach. [S.l.] : John Wiley & Sons, 2011

MIGUEL, António - Gestão de Projectos de Software. FCA-Editora Informática, Ed. 2003).

MIYAMOTO - The book of five rings. [S.l.] : Shambhala Publications, 1992

MUJUMDAR, Ashwini; MASIWAL, Gayatri; CHAWAN, PM - Analysis of various software process models. International Journal of Engineering Research and Applications. 2:3 (2012) 2015–2021.

MUNASSAR, Nabil Mohammed Ali; GOVARDHAN, A. - A comparison between five models of software engineering. IJCSI. 5:2010) 95–101.

MUSHTAQ, Zaigham; QURESHI, M. Rizwan Jameel - Novel hybrid model: Integrating Scrum and XP. IJ Information Technology and Computer Science. 6:2012) 39–44.

NOUREDDINE, Adam A.; DAMODARAN, Meledath; YOUNES, Samira - A Framework for Harnessing the Best of Both Worlds in Software Project Management: Agile and Traditional. Em Information Systems Education Conference 2009. [S.l.] : Citeseer, 2009

OATES, Briony J. - Researching information systems and computing. [S.l.] : Sage, 2005

OCAMB, Scott - Scrum Alliance - What Does the Agile Manifesto Mean? [Em linha], atual. 2013. [Consult. 7 mar. 2017]. Disponível em WWW:<URL:https://www.scrumalliance.org/community/articles/2013/april/what-does-the-agile-manifesto-mean>.

OSTERWALDER, Alexander; PIGNEUR, Yves - Business model generation: a handbook for visionaries, game changers, and challengers. [S.l.] : John Wiley & Sons, 2010

OUTSYSTEMS - The #1 Low-Code Platform for Digital Transformation [Em linha], atual. 2010. [Consult. 24 mar. 2017]. Disponível em WWW:<URL:https://www.outsystems.com/home/downloadsdetail/25/182/>.

PAHUJA, Savita - What is Scrumban? [Em linha], atual. 2012. [Consult. 16 mar. 2018]. Disponível em WWW:<URL:https://www.solutionsiq.com/learning/blog-post/what-is-scrumban/>.

PAULSON, Linda Dailey - Adapting methodologies for doing software right. IT Professional. 3:4 (2001) 13–15.

PETER - Agile by the Numbers 2013 [Em linha], atual. 7 nov. 2013. [Consult. 7 mar. 2017]. Disponível em WWW:<URL:https://agilescout.com/agile-numbers-2013/>.

PHILLIPS, Joseph - CAPM/PMP Project Management All-in-One Exam Guide. [S.l.] : McGraw-Hill, Inc., 2007

PRESSMAN, Roger S. - Software engineering: a practitioner's approach. [S.l.] : Palgrave Macmillan, 2005

PRIKLADNICKI, RAFAEL; ORTH, AFONSO INACIO - Planejamento & gerência de projetos. [S.l.] : EDIPUCRS, 2009

RAJAGOPALAN, Sriram; MATHEW, Saji K. - Choice of agile methodologies in software development: A vendor perspective. Journal of International Technology and Information Management. 25:1 (2016) 3.

RAJU, HK; KRISHNEGOWDA, YT - Kanban pull and flow-a transparent workflow for improved quality and productivity in software development. 2013).

REINERTSEN, Donald; BELLINSON, Tom - The principles of product development flow: second generation lean product development. [S.l.] : Celeritas Publishing–2008 Додаток Б Додаток В Додаток, 2014

RICO, David F.; SAYANI, Hasan H.; SONE, Saya - The business value of Agile software methods. Maximizing ROI with just-in-time processes and documentation, Fort Lauderdale, FL: J. Ross Pub. 10:2009).

ROBIOLO, Gabriela; GRANE, Daniel - Do agile methods increase productivity and quality? American Journal of Software Engineering and Applications. 3:1 (2014) 1–11.

ROYCE, Winston W. - Managing the development of large systems: Concepts and techniques. Em 9th International Conference on Software Engineering. ACM

SALAMEH, Hanadi - What, When, Why, and How? A Comparison between Agile Project Management and Traditional Project Management Methods. International Journal of Business and Management Review. 2014).

SCHWABER, K.; BEEDLE, M. - Agile Software Development with ScrumAgile Software Development. [Em linha]. [S.l.] : Prentice Hall, 2002 Disponível em WWW:<URL:https://books.google.pt/books?id=BpFYAAAAYAAJ>. ISBN 978-0-13-067634-4.

SCHWABER, Ken; SUTHERLAND, Jeff - Guia do Scrum. Julho de. 2013).

SCHWALBE, Kathy - Managing a Project Using an Agile Approach and the PMBOK® Guide. Em Proceedings of the Information Systems Educators Conference ISSN

SCHWALBE, Kathy - Information technology project management. [S.l.] : Cengage Learning, 2015

SEMEDO, Maria João Moreno; OTHERS - Ganhos de produtividade e de sucesso de Metodologias Ágeis VS Metodologias em Cascata no desenvolvimento de projectos de software. 2012).

SHARMA, Mayanka; KOTWAL, Iqbal - A Concept Note on Impact of Agile Software Project Management Methods in Midsized IT Product Development Companies. We'Ken-International Journal of Basic and Applied Sciences. 1:3 (2016) 110–116.

SIDDIQUE, Lubna; HUSSEIN, Bassam A. - Managing risks in Norwegian Agile Software Projects: Project Managers perspective. Examination of the challenges in agile projects from the suppliers 'perspective in Norways software industry Insight and recommendations. 2017) 125.

SINGH, Ravinder; LANO, Kevin - Defining and formalising project management models and processes. Em Science and Information Conference (SAI), 2014. [S.l.] : IEEE, 2014

SNYDER, Cynthia Stackpole - A guide to the project management body of knowledge: PMBOK (®) guide. Project Management Institute: Newtown Square, PA, USA. 2014).

SOARES, Michel Dos Santos - Comparação entre metodologias Ágeis e tradicionais para o desenvolvimento de software. INFOCOMP. 3:2 (2004) 8–13.

SOL, Henk G. - Information systems design methodologies: A feature analysis. ELSEVIER SCI. PUBL., P. O. BOX 211, 1000 AE AMSTERDAM, NETHERLANDS, 1983, 266. 1983).

SOMMERVILLE, I. - Software EngineeringInternational computer science series Software engineering. [Em linha]. [S.l.] : Addison-Wesley, 2007 Disponível em WWW:<URL:https://books.google.pt/books?id=B7idKfL0H64C>. ISBN 978-0-321-31379-9.

SONG, Xiping; OSTERWEIL, Leon J. - Experience with an approach to comparing software design methodologies. IEEE Transactions on Software Engineering. 20:5 (1994) 364–384.

STOICA, Marian *et al.* - Analyzing Agile Development-from Waterfall Style to Scrumban. Informatica Economica. 20:4 (2016) 5.

STOICA, Marian; MIRCEA, Marinela; GHILIC-MICU, Bogdan - Software development: Agile vs. traditional. Informatica Economica. 17:4 (2013) 64.

SUTHERLAND, Jeff - The scrum papers: nut, bolts, and origins of an Agile framework. Scrum inc. 2012).

TANVEER, Mohsan - Agile for large scale projects—A hybrid approach. Em Software Engineering Conference (NSEC), 2015 National. [S.l.] : IEEE, 2015

TEIXEIRA, Bruno Afonso - Agile and traditional project management: bridge between two worlds to manage IT projects PhD Thesis.

UIKEY, Nitin; SUMAN, Ugrasen - An empirical study to design an effective agile project management framework. Em Proceedings of the CUBE International Information Technology Conference. [S.l.] : ACM, 2012

VERSIONONE - Version One [Em linha], atual. 2016. [Consult. 9 mar. 2017]. Disponível em WWW:<URL:http://info.versionone.com/state-of-agile-report-thank-you.html>.

VRIENS, Christ - Certifying for CMM Level 2 and IS09001 with XP@ Scrum. Em Agile Development Conference, 2003. ADC 2003. Proceedings of the. [S.l.] : IEEE, 2003

WALLACE, Nathan; BAILEY, Peter; ASHWORTH, Neil - Managing xp with multiple or remote customers. Em Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002)

WEBSTER, Francis - Pm 101: according to the olde curmudgeon-an introduction to the basic concepts of modern project management. Em . [S.l.] : Project Management Institute, 2000

What is the Kanban Method? | Lean Kanban - [Em linha], atual. 2017. [Consult. 16 mar. 2018]. Disponível em WWW:<URL:http://leankanban.com/project/what-is-km/>.

WILLIAMS, Laurie; COCKBURN, Alistair - Guest Editors' Introduction: Agile Software Development: It's about Feedback and Change. Computer. 36:6 (2003) 39–43.

YILMAZ, Murat; O'CONNOR, Rory V. - A Scrumban integrated gamification approach to guide software process improvement: a Turkish case study. Tehnički vjesnik. 23:1 (2016) 237–245.

# ANEXO 1 – PROCESSOS DO PMBOK

| Áreas de conhecimento | Grupos de processos | | | | |
|---|---|---|---|---|---|
| | Início | Planeamento | Execução | Monitorização e Controlo | Fecho |
| Integração | 1 Definir o termo de abertura do projeto | 2 Desenvolver o plano de gestão do projeto | 3 Executar o plano de projeto | 4 Monitorizar e controlar o projeto;<br>5 Controlo de mudanças | 6 Encerrar o projeto |
| Âmbito | | 1 Planear a gestão do âmbito;<br>2 Reunir os requisitos;<br>3 Definir o âmbito;<br>4 Criar a EAP | | 5 Validar o âmbito;<br>6 Controlar o âmbito | |
| Tempo | | 1 Planeamento da gestão do tempo;<br>2 Definir atividades;<br>3 Sequenciar as atividades;<br>4 Estimar os recursos das atividades;<br>5 Estimar duração das atividades;<br>6 Desenvolver cronogramas | | 7 Controlar o cronograma | |
| Custos | | 1 Planear a gestão dos custos;<br>2 Estimar custos;<br>3 Determinar orçamento | | 4 Controlar os custos | |
| Qualidade | | 1 Planear a gestão da qualidade | 2 Realizar garantia da qualidade | 3 Controlar a qualidade | |
| Recursos Humanos | | 1 Planeamento da gestão dos recursos humanos | 2 Adquirir equipa de projeto;<br>3 Desenvolver a equipa de projeto;<br>4 Gerir a equipa de projeto | | |
| Comunicação | | 1 Planear a gestão da comunicação | 2 Gerir a comunicação | 3 Controlar as comunicações | |

| Áreas de conhecimento | Grupos de processos | | | | |
|---|---|---|---|---|---|
| | Início | Planeamento | Execução | Monitorização e Controlo | Fecho |
| Riscos | | 1 Planear a gestão de riscos;<br>2 Identificar riscos;<br>3 Realizar análise qualitativa dos riscos;<br>4 Realizar análise quantitativa dos riscos;<br>5 Planear a resposta ao risco | | 6 Controlar os riscos | |
| Recursos | | 1 Planear a gestão de aquisições | 2 Conduzir aquisições | 3 Controlar as aquisições | 4 Encerrar as aquisições |
| *Stakeholders* | 1 Identificar os *stakeholders* | 2 Planear a gestão dos *stakeholders* | 3 Gerir *stakeholders* | 4 Controlar o envolvimento dos *stakeholders* | |

# ANEXO 2 – METODOLOGIAS ÁGEIS

## Adpative Software Development

A metodologia Adpative Software Development (ASD), foi proposta por Jym Highsmith (Highsmith, 2013) para ajudar em projetos de grandes dimensões e de complexidade elevada. A filosofia desta metodologia é promover a colaboração e as equipas serem auto-organizadas.

O ciclo definido por Highsmith (Highsmith, 2000) é composto por três fases: especular, colaborar e aprender, formando um ciclo entre si, como podemos observar na Figura 1.

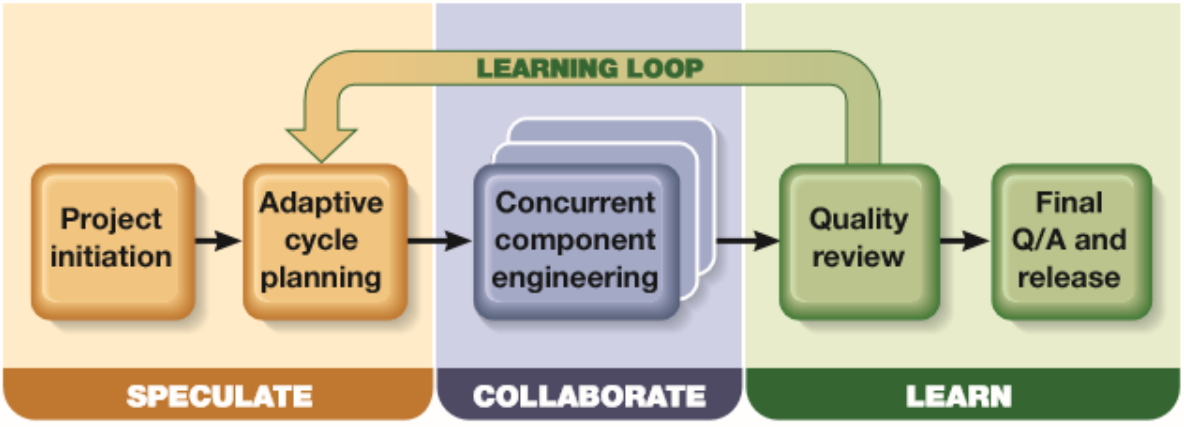


**Figura 1 - Ciclo ASD**
**Fonte: (Highsmith, 2000)**

A fase de especulação divide-se em duas. A inicial contém as restrições do projeto, os requisitos básicos e os ciclos a considerar. Já a fase de planeamento de cada ciclo, inclui em detalhe as tarefas a desenvolver no decorrer do ciclo para entregar ao cliente, assim como *feedback* obtido no ciclo anterior, de forma a refletir a realidade que a equipa consegue cumprir.

A fase de colaboração tem como objetivo entregar o que foi definido no planeamento. Para isso, toda a equipa compromete-se com a entrega. Não existe nenhuma recomendação de como a equipa deve trabalhar, podendo inclusivamente adotar práticas de outras metodologias. A comunicação entre todas as partes é a chave da colaboração.

A aprendizagem vai-se alcançando à medida que os ciclos vão terminando, uma vez que no término de cada ciclo deve existir *feedback* por parte do cliente, do que foi desenvolvido.

## Crystal

O método de desenvolvimento Crystal, que podemos analisar na Figura 2, é na realidade uma família de métodos, constituída por Crystal Clear, Crystal Yellow, Crystal Orange, Crystal Red e Crystal Marron, desenvolvida por Alistair Cockburn e Jim Highsmith (Highsmith e Cockburn, 2001). As cores da Crystal servem para determinar qual a melhor metodologia a implementar num projeto, atendendo a alguns fatores como a sua importância, dimensão, coordenação e comunicação. Quanto mais escura for a cor, maior o grau de complexidade do projeto. Então, para selecionar a metodologia, é necessário ligar o projeto a um dos quatro níveis críticos: conforto, dinheiro disponibilidade, dinheiro essencial e vida. Além dos níveis, é necessário saber quantas pessoas irão trabalhar no projeto para que se possa escolher o método (Boehm e Turner, 2003).

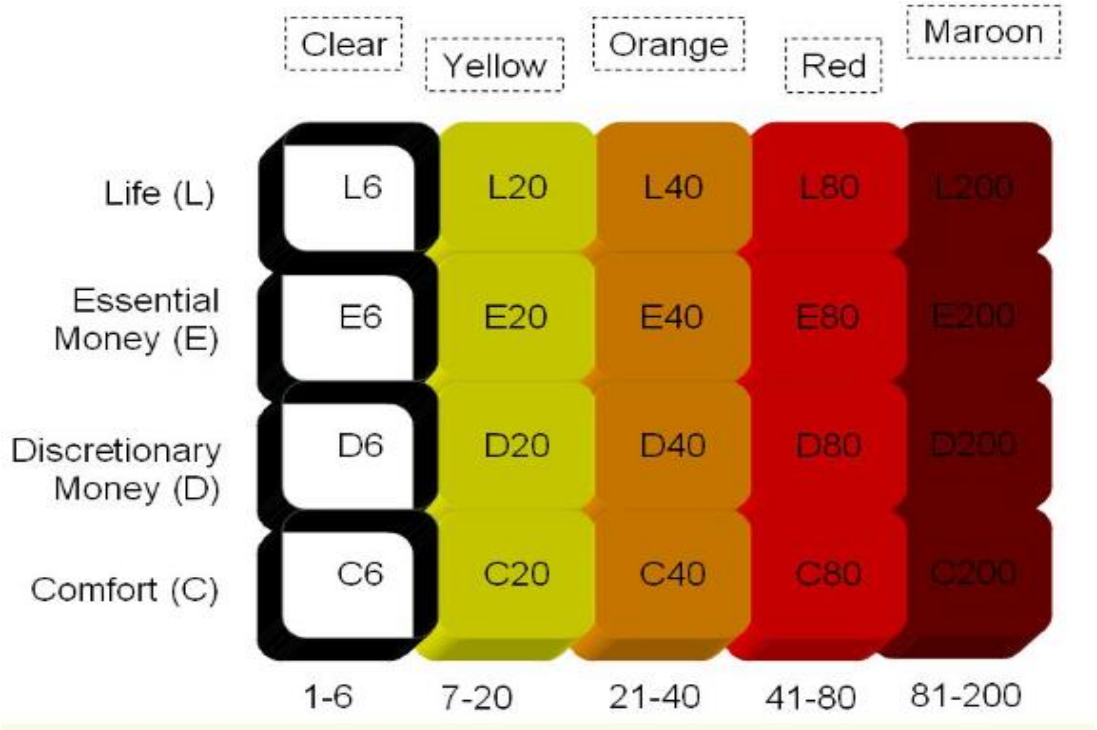


**Figura 2 - Metodologia Crystal**
**Fonte: (Highsmith, 2000)**

Os princípios fundamentais da Crystal assentam em entregas frequentes, as quais podem variar entre um a quatro meses. São necessárias reflexões após cada entrega, onde a equipa tenta identificar possíveis melhorias e avalia o estado do projeto, sendo primordial a comunicação entre a equipa e o cliente (Boehm e Turner, 2003).

Para seguirem a metodologia Crystal, as equipas devem respeitar um conjunto de estratégias (Boehm e Turner, 2003):

- Exploração 360º – No início do projeto, por norma, no planeamento, a equipa necessita verificar se o projeto é exequível, utilizando as ferramentas propostas. Para tal, é necessário analisar o valor para o negócio, os requisitos, as tecnologias a usar, a equipa de trabalho e o processo a utilizar. Esta fase pode durar entre alguns dias a semanas.

- Vitória antecipada – A vitória antecipada contribui para a autoconfiança da equipa e pode ser tão simples como inserir um registo na base de dados, ou apresentar um *mockup* ao cliente e receber um elogio.

- Protótipo funcional – Não é mais do que uma pequena implementação do sistema, mesmo que não use a arquitetura do projeto.

- Desenvolvimento incremental - O sistema irá evoluir com base no protótipo e irá sofrer alterações com base em novos requisitos ou *feedback* do cliente. De salientar que o sistema nunca pode deixar de funcionar.

- Painel de informações – Deve estar disponível para toda a equipa um painel contendo todas as informações relevantes para o projeto.

## Dynamic Systems Development Method

A metodologia Dynamic Systems Development Method (DSDM) surge para ajudar os projetos, cujo prazo de entrega é limitado. Para conseguir manter a satisfação do cliente, recorre à prototipagem incremental. Este processo segue o princípio dos 80-20, onde os 80% correspondem à percentagem da aplicação que deve ser entregue e os 20% do tempo de projeto (Pressman, 2005), (AgileBusiness, 2017).

A DSDM Consortium (AgileBusiness, 2017) é um grupo de empresas mundiais que assume ser o detentor do método e define o ciclo de vida do DSDM em três fases sequenciais: Pré-projeto, Projeto e Pós-projeto, como podemos avaliar na Figura 3.

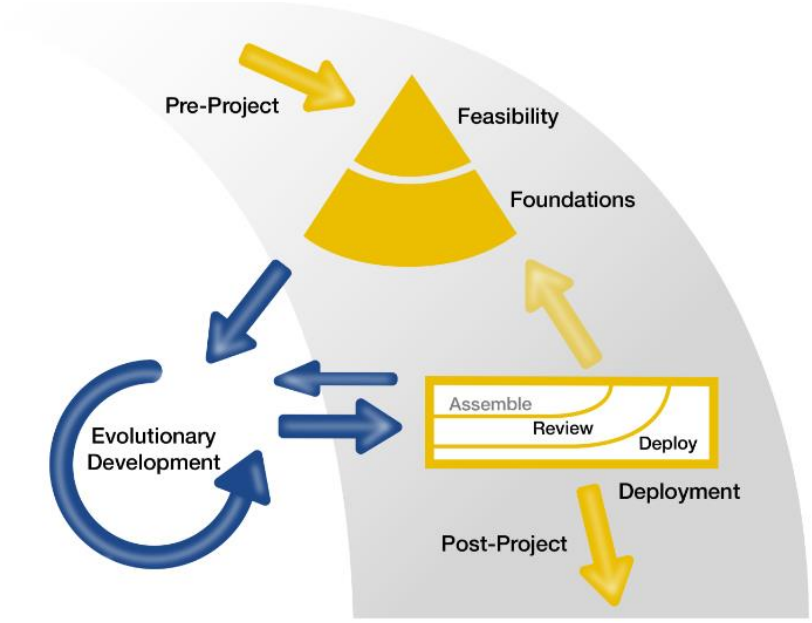


**Figura 3 - Metodologia DSDM**
**Fonte: (AgileBusiness, 2017)**

A fase do pré-projeto assegura que os projetos sejam corretamente criados com base num objetivo bem claro. Nesta fase são também tratadas questões como o plano financeiro e o cronograma (AgileBusiness, 2017).

A fase do projeto encontra-se dividida em cinco níveis (AgileBusiness, 2017):

- Estudo de viabilidade - Serve para verificar se o projeto é viável tecnicamente e do ponto de vista de negócio. O esforço nesta fase deve ser o suficiente para decidir se o projeto deve ser investigado mais profundamente ou interrompido;

- Fase das fundações - Esta fase já se pretende obter uma compreensão da lógica de negócio, mas não de forma detalhada, uma vez que o detalhe de cada requisito será estabelecido no desenvolvimento. Esta fase deve ser de curta duração, não devendo prolongar-se por mais que algumas semanas. Em qualquer fase do projeto, pode-se voltar a esta fase, já que podem existir alterações pedidas pelo cliente. Também é nesta fase que se determina qual o ciclo de vida do projeto, onde será realizado, por quem e quando;

- Fase do desenvolvimento - Tem como objetivo evoluir a solução, criando incrementos com detalhe dos requisitos e testando-os continuamente. Para isso, recorre a algumas práticas como:

- *Time-boxes*, onde define o período para a execução da entrega;
- Priorização dos requisitos, para serem realizados os de maior importância;
- Prototipagem, para ajudar a compreender melhor a solução e facilitar a análise com o cliente;
- Testes para validar.

- Fase da implementação - É a entrega ao cliente da solução final ou parcial. Nesta fase, é essencial garantir que não se compromete as entregas anteriores, como tal, é necessário efetuar uma revisão das funcionalidades, de forma a garantir o correto funcionamento. É neste momento que se dá o encerramento, reunindo-se a equipa para realizar uma retrospetiva do desempenho geral do projeto;

- Fase do pós-projeto - É a última fase, onde são validados o cumprimento dos objetivos e se resultaram benefícios para o negócio. É assegurada a manutenção e melhoramentos de acordo com os princípios do DSDM.

Esta metodologia determina alguns papéis para o seu correto funcionamento, como podemos observar na Figura 4. Os itens cor de laranja da Figura 4, são os interesses comerciais e que representam a visão do negócio. Os itens cor verde são os interesses técnicos que representam as equipas que irão desenvolver. Os itens cor azul representam os interesses de gestão de projeto. A cinza, os interesses processuais que representam a visão do processo e, por fim, os que contêm duas cores, são papéis que assumem duas áreas (AgileBusiness, 2017).

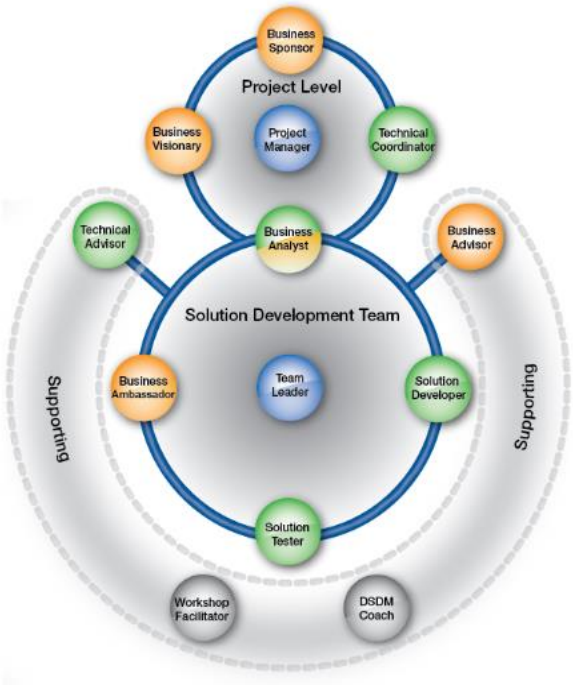


**Figura 4 - Papeis DSDM**
**Fonte: (AgileBusiness, 2017)**

## Feature Driven Development

A metodologia Feature Driven Development (FDD) foi desenvolvida inicialmente por Jeff De Luca e por Perter Coad. Os dois autores juntaram os seus conhecimentos, uma vez que De Luca desenvolvia sistemas com base em metodologias lineares e Peter Coad desenvolvia sistemas com base em modelos orientados a objetos. A junção dos conceitos e processos originou a Metodologia *Ágil Feature Driven Development* (Coad, Luca e Lefebvre, 1999), (Pressman, 2005).

Mais tarde, Stephen Plamer estendeu o trabalho realizado por Coad, para adaptar a metodologia a projetos de média e grandes dimensões (Coad, Luca e Lefebvre, 1999), (Pressman, 2005).

Esta metodologia releva o desenvolvimento iterativo, a qualidade e uma gestão de projetos flexível e adaptável, para ser usada por clientes e pela equipa de desenvolvimento (Rico, Sayani e Sone, 2009). Na Figura 5 podemos ver o ciclo de vida da metodologia FDD, que consiste em duas fases e cinco processos:

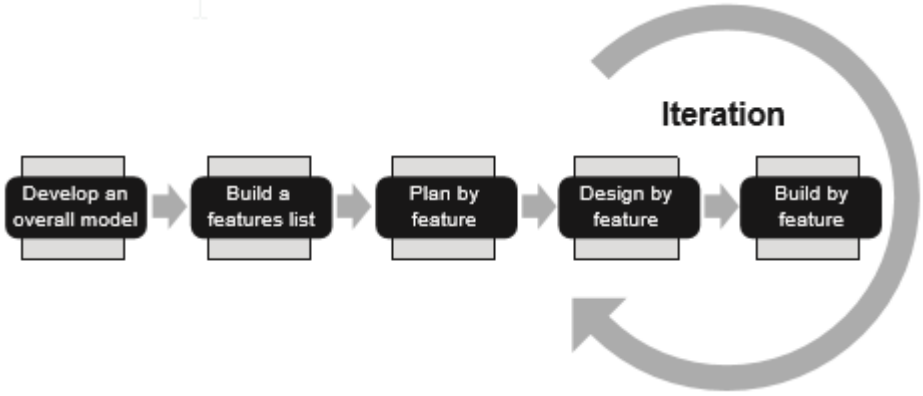


**Figura 5 - Ciclo de vida FDD**
**Fonte: (Rico, 2009)**

- Conceção e planeamento

    - Desenvolver um modelo abrangente – Esta atividade inicial pretende analisar todo o projeto. Por isso, devem estar envolvidos os elementos seniores das equipas de desenvolvimento com experiência que entendam de lógica e de negócio;

    - Construir a lista de funcionalidades – Esta atividade pretende identificar todas as funcionalidades necessárias que satisfaçam os requisitos do projeto. A equipa que determina as funcionalidades é geralmente composta por elementos seniores que estiveram envolvidos na fase anterior;

    - Planear as funcionalidades – Nesta tarefa encontram-se reunidos os elementos da gestão do projeto e alguns elementos da equipa de desenvolvimento. Têm como função ordenar as funcionalidades com base nas suas dependências, carga de trabalho e complexidade.

- Desenvolvimento

    - Detalhar as funcionalidades – É feita para cada funcionalidade. A cada iteração são selecionadas algumas funcionalidades pelos programadores seniores e atribuídas a todos os elementos da equipa. Nesta etapa são detalhados todos os requisitos, assim como os testes;

- Desenvolver as funcionalidades – O programador responsável pela funcionalidade deve realizar a sua codificação e implementar todos os testes. Pode também haver revisão do código por um outro membro da equipa. Após a funcionalidade estar fechada, é incrementada ao projeto.

# Referências

AGILEBUSINESS, Consortium - Home page [Em linha], atual. 2017. [Consult. 24 mar. 2017]. Disponível em WWW:<URL:https://www.agilebusiness.org/>.

HIGHSMITH, J. - Retiring Lifecycle Dinosaurs: Using Adaptive Software Development to Meet the Challenges of a High-Speed, High-Change Environment (pp. 22–28). Software Testing & Quality Engineering. 2000).

HIGHSMITH, Jim; COCKBURN, Alistair - Agile software development: The business of innovation. Computer. 34:9 (2001) 120–127.

RICO, David F.; SAYANI, Hasan H.; SONE, Saya - The business value of agile software methods: maximizing ROI with just-in-time processes and documentation. [S.l.] : J. Ross Publishing, 2009

# ANEXO 3 – EQUIVALÊNCIAS ENTRE PMBOK E SCRUM

| Grupo | Área de conhecimento | Processo | Equivalência no Scrum | 0 Não Atende 4 Atende completamente |
|---|---|---|---|---|
| Monitorização e controlo | Âmbito | Validar o âmbito | O Scrum usa as *Review* para controlar o âmbito e as expectativas do cliente | 4 |
| Monitorização e controlo | Âmbito | Controlar o âmbito | Controla o âmbito usando os *sprints* e backlog | 4 |
| Monitorização e controlo | Âmbito | Controlar o cronograma | Controla, verificando as atividades feitas num *sprint* | 3 |
| Planeamento | Âmbito | Planear a gestão do âmbito | Cria o backlog e vai colocando itens em *sprints* | 2 |
| Planeamento | Âmbito | Reunir os requisitos | O Scrum determina que o PO é responsável por reunir os requisitos, mas não determina técnicas e ferramentas | 2 |
| Planeamento | Âmbito | Definir o âmbito | Para o Scrum é a criação do Backlog, sendo da responsabilidade do PO toda a sua ordenação. Além disso, em todos os *sprints*, é revisto o âmbito | 4 |
| Planeamento | Âmbito | Criar a EAP | No Scrum podemos ver o Backlog e o seu progresso | 1 |
| Execução | Comunicação | Gerir comunicação | O PO garante que toda a equipa recebe a informação | 3 |
| Monitorização e controlo | Comunicação | Controlar as comunicações | O Scrum master garante que todas as cerimónias acontecem e nelas estão presentes os *stakeholders* | 4 |
| Planeamento | Comunicação | Planear a gestão da comunicação | As comunicações no Scrum já estão planeadas (reuniões de sprint, reuniões diárias, etc), mas não obriga que todos os *stakeholders* a estejam presentes | 3 |
| Monitorização e controlo | Custos | Controlar os custos | - | 0 |

| Grupo | Área de conhecimento | Processo | Equivalência no Scrum | 0 Não Atende 4 Atende completamente |
|---|---|---|---|---|
| Planeamento | Custos | Planear a gestão dos custos | - | 0 |
| Planeamento | Custos | Estimar custos | - | 0 |
| Planeamento | Custos | Determinar orçamento | - | 0 |
| Execução | Integração | Executar o plano do projeto | Acompanha em detalhe o projeto | 4 |
| Fecho | Integração | Encerrar o projeto | Não determina como encerrar o projeto | 2 |
| Início | Integração | Definir termo de abertura | Podemos considerar a visão do projeto e atribuição de papéis | 1 |
| Monitorização e controlo | Integração | Monitorizar e controlar projeto | Verificação diária do trabalho nas daily Scrum e acompanhamento no fim de *sprint* | 4 |
| Monitorização e controlo | Integração | Controlar as mudanças | Verifica as mudanças a cada *sprint*, e as prioridades entre os itens do backlog | 4 |
| Planeamento | Integração | Desenvolver o plano de gestão | O Scrum determina como serão feitas as entregas e com que itens | 3 |
| Execução | Qualidade | Realizar a garantia da qualidade | Durante a execução de um *sprint* os isseus só são classificados como realizados, quando cumprem a definição de concluídos | 2 |
| Monitorização e controlo | Qualidade | Controlar a qualidade | Verifica a qualidade após a entrega ao cliente | 2 |
| Planeamento | Qualidade | Planear a gestão da qualidade | Todos os itens de backlog contêm o conceito de DONE | 3 |
| Execução | Recursos | Conduzir aquisições | - | 0 |
| Fecho | Recursos | Encerrar as aquisições | - | 0 |
| Monitorização e controlo | Recursos | Controlar as aquisições | - | 0 |
| Planeamento | Recursos | Planear a gestão de aquisições | - | 0 |

| Grupo | Área de conhecimento | Processo | Equivalência no Scrum | 0 Não Atende 4 Atende completamente |
|---|---|---|---|---|
| Execução | Recursos Humanos | Adquirir a equipa de projeto | - | 0 |
| Execução | Recursos Humanos | Desenvolver a equipa de projeto | Diariamente, a equipa verifica o que a equipa precisa | 4 |
| Execução | Recursos Humanos | Gerir a equipa de projeto | Não define claramente como o PO ou Scrum Master gerem conflitos. As equipas são auto-organizadas | 3 |
| Planeamento | Recursos Humanos | Planeamento dos recursos humanos | Define quem são os recursos e as suas responsabilidades, mas não indica como se deve angariar os recursos | 3 |
| Monitorização e controlo | Riscos | Controlar os riscos | Controla através das reuniões diárias e de fim de *sprint* | 2 |
| Planeamento | Riscos | Planear a gestão dos riscos | - | 0 |
| Planeamento | Riscos | Identificar os riscos | Identifica os impedimentos do projeto | 1 |
| Planeamento | Riscos | Realizar a análise qualitativa do risco | - | 0 |
| Planeamento | Riscos | Realizar a análise quantitativa do risco | - | 0 |
| Planeamento | Riscos | Planear a resposta ao risco | - | 0 |
| Execução | *Stakeholders* | Gerir *stakeholders* | Durante os eventos do Scrum, os *stakeholders* estão envolvidos, não garantido a participação de todos | 4 |
| Início | *Stakeholders* | Identificar os *stakeholders* | Os *stakeholders* encontram-se identificadas | 4 |
| Monitorização e controlo | *Stakeholders* | Controlar o envolvimento dos *stakeholders* | O Scrum Master e o PO garantem o envolvimento dos *stakeholders*, mas não de todos | 4 |
| Planeamento | *Stakeholders* | Planear a gestão dos *stakeholders* | No Scrum, os *stakeholders* estão identificados e sabem de todas as cerimónias | 4 |

| Grupo | Área de conhecimento | Processo | Equivalência no Scrum | 0 Não Atende<br>4 Atende completamente |
|---|---|---|---|---|
| Planeamento | Tempo | Planeamento da gestão do Tempo | O planeamento do tempo é realizado com base no número de elementos da equipa, e em Scrum, é fixo | 3 |
| Planeamento | Tempo | Definir as atividades | Em cada *sprint* a equipa de projeto define todas as atividades | 4 |
| Planeamento | Tempo | Sequenciar as atividades | O PO determina a ordem do Backlog e a equipa a ordem das atividades num *sprint*, mas não determina ligações lógicas | 3 |
| Planeamento | Tempo | Estimar os recursos das atividades | Com base nos elementos da equipa determina-se o que é possível fazer | 4 |
| Planeamento | Tempo | Estimar duração das atividades | Verifica quantos itens do Backlog é possível implementar num *sprint* | 2 |
| Planeamento | Tempo | Desenvolver cronograma | Não tem um cronograma só o backlog | 2 |

# ANEXO 4 – PROPOSTA PROJETO PILOTO

## Registo horas

Pretende-se uma aplicação para gerir e controlar as horas de trabalho de um conjunto de colaboradores, devendo a aplicação ser web e mobile.

### Pontos fundamentais

#### 1. Registar horas

Um utilizador, durante o seu dia de trabalho possui, equipamentos mobile consigo onde deve dar inicio e fim de cada atividade que faz, para assim existir um controlo correto dos tempos gastos. Devem ser considerados três tipos de períodos:

a. Registar pausas

Sempre que um colaborador efetua uma pausa para almoço, lanche ou outros tipos, deve indiciar o seu início e fim.

b. Registar tempos mortos

Sempre que se encontra sem trabalho, tanto por avaria do material de trabalho, ou por falta de trabalho, deve indicar o seu início e fim.

c. Registar horas trabalhadas

Quando faz trabalhos para um projeto o colaborador deve indicar o inicio e o fim do trabalho e qual o projeto

#### 2. Aprovar horas

As horas devem ser validadas por entidades diferentes:

a. Chefia

As chefias devem verificar se os seus colaboradores registaram todas as horas dos seus subordinados, após esta aprovação, não deve ser possível nenhuma alteração.

b. Recursos Humanos

No início do mês os recursos humanos devem verificar se todos os colaboradores têm horas e passar o registo das mesmas para o sistema de ERP.

## 3. Relatórios de gestão

Deve ser possível extrair relatórios com as horas, nomeadamente:

a. Por projeto,
b. Por colaborador,
c. Por departamento.

## 4. Sincronizações

A aplicação é um facilitador para os colaboradores, devendo ler e escrever do sistema central da empresa.

a. Leitura
   Estrutura orgânica; Projetos; classificação horas.

b. Escrita
   Horas inseridas na aplicação.

# ANEXO 5 - QUESTIONÁRIO

| Questionário | | |
|---|---|---|
| Grupo | Questão | Resultado<br>0 Se não teve melhorias<br>5 Se melhorou todos os aspetos |
| Genérica | O desenvolvimento com XPOKs aumentou a eficácia do desenvolvimento?<br>Ao desenvolver *software* com o XPOKs diminuíram os erros. Os itens do *backlog* continham a informação necessária, ajudando a equipa de desenvolvimento a ganhar tempo na sua perceção. | |
| | O desenvolvimento com XPOKs aumentou a qualidade do produto?<br>O cliente demonstrou que as entregas correspondiam ao seu pedido, não necessitando, após a entrega, pedir alterações de trabalho por especificações insuficientes. Reportou uma quantidade reduzida de erros. | |
| | O desenvolvimento com XPOKs aumenta a colaboração?<br>Existe uma maior aproximação entre o cliente e a equipa de desenvolvimento. A própria equipa trabalhou em união para a resolução dos problemas. | |
| | O desenvolvimento com XPOKs torna o trabalho mais organizado/ planeado?<br>A equipa de qualidade não necessitou de explicações por parte de quem desenvolvia para proceder com testes, assim como a equipa de desenvolvimento não necessitava de recorrer constantemente ao EM. Todos os elementos sabiam sempre o foco de cada *sprint.* | |
| | O desenvolvimento com XPOKs permite a deteção precoce de erros / defeitos?<br>As práticas implementadas, como o *definition of done*, ajudou na prevenção de erros, ou seja a equipa de qualidade não teve necessidade de reportar tantos erros. O facto do cliente interagir diariamente, diminuiu o número de erros pedidos, após a entrega. | |
| Metodologia XPOKs | Considera fundamental a divisão do projeto em 3 fases?<br>O facto do projeto ter três fases diferentes: início, ciclos de desenvolvimento e fecho contribuíram para seu sucesso. | |
| | Práticas da primeira fase, início do ciclo?<br>Esta pergunta pretende saber se considera as seguintes práticas importantes durante o início do projeto e se são uteis durante o seu decorrer. | - |
| | Termo de abertura | |
| | Visão do produto | |
| | *Product Backlog* | |
| | Modelo de arquitetura | |
| | Práticas dos ciclos de desenvolvimento?<br>Esta pergunta pretende saber se considera as seguintes práticas importantes durante o decorrer do projeto. | - |

| | | |
|---|---|---|
| | Documentação | |
| | Reuniões diárias com o cliente | |
| | Visão do *sprint* | |
| | *Pair programming* | |
| | *Refactoring* | |
| | Integrações contínuas | |
| | Práticas do fecho do ciclo?<br>É importante esta fase estar dividida em formação e estabilização. O facto de, neste momento, não existir planeamento como era feito durante os ciclos de desenvolvimento é benéfico. | - |
| | Formação | |
| | Estabilização | |
| | Ausência de planeamento | |

| | | |
|---|---|---|
| Utilização Futura | Se pudesse escolher uma metodologia para o seu projeto, optava pelo XPOKs? | |
| | Uma vez que o XPOKs ainda não foi aplicado a um número significativo de projetos, apresentá-lo-ia hoje aos seus superiores, para aplicarem em trabalhos futuros? | |

# ANEXO 6 – PMBOK VS SCRUM E XPOKS

| Grupo | Área de conhecimento | Processo | 0 Não Atende<br>4 Atende completamente | |
|---|---|---|---|---|
| | | | Scrum | XPOKs |
| **Monitorização e controlo** | Âmbito | Validar o âmbito | 4 | 4 |
| **Monitorização e controlo** | Âmbito | Controlar o âmbito | 4 | 4 |
| **Monitorização e controlo** | Âmbito | Controlar o cronograma | 3 | 3 |
| **Planeamento** | Âmbito | Planear a gestão do âmbito | 2 | 3 |
| **Planeamento** | Âmbito | Reunir os requisitos | 2 | 3 |
| **Planeamento** | Âmbito | Definir o âmbito | 4 | 4 |
| **Planeamento** | Âmbito | Criar a EAP | 1 | 1 |
| **Execução** | Comunicação | Gerir comunicação | 3 | 3 |
| **Monitorização e controlo** | Comunicação | Controlar as comunicações | 4 | 4 |
| **Planeamento** | Comunicação | Planear a gestão da comunicação | 3 | 3 |
| **Monitorização e controlo** | Custos | Controlar os custos | 0 | 0 |
| **Planeamento** | Custos | Planear a gestão dos custos | 0 | 0 |
| **Planeamento** | Custos | Estimar custos | 0 | 0 |
| **Planeamento** | Custos | Determinar orçamento | 0 | 0 |
| **Execução** | Integração | Executar o plano do projeto | 4 | 4 |
| **Fecho** | Integração | Encerrar o projeto | 2 | 4 |
| **Início** | Integração | Definir termo de abertura | 1 | 3 |
| **Monitorização e controlo** | Integração | Monitorizar e controlar projeto | 4 | 4 |
| **Monitorização e controlo** | Integração | Controlar as mudanças | 4 | 4 |
| **Planeamento** | Integração | Desenvolver o plano de gestão | 3 | 3 |
| **Execução** | Qualidade | Empreender a garantia da qualidade | 2 | 3 |
| **Monitorização e controlo** | Qualidade | Controlar a qualidade | 2 | 3 |
| **Planeamento** | Qualidade | Planear a gestão da qualidade | 3 | 3 |

| Grupo | Área de conhecimento | Processo | 0 Não Atende<br>4 Atende completamente | |
|---|---|---|---|---|
| | | | Scrum | XPOKs |
| **Execução** | Recursos | Conduzir aquisições | 0 | 0 |
| **Fecho** | Recursos | Encerrar as aquisições | 0 | 0 |
| **Monitorização e controlo** | Recursos | Controlar as aquisições | 0 | 0 |
| **Planeamento** | Recursos | Planear a gestão de aquisições | 0 | 0 |
| **Execução** | Recursos Humanos | Adquirir equipa de projeto | 0 | 0 |
| **Execução** | Recursos Humanos | Desenvolver a equipa de projeto | 4 | 4 |
| **Execução** | Recursos Humanos | Gerir a equipa de projeto | 3 | 3 |
| **Planeamento** | Recursos Humanos | Planeamento dos recursos humanos | 3 | 3 |
| **Monitorização e controlo** | Riscos | Controlar os riscos | 2 | 3 |
| **Planeamento** | Riscos | Planear a gestão dos riscos | 0 | 0 |
| **Planeamento** | Riscos | Identificar os riscos | 1 | 3 |
| **Planeamento** | Riscos | Realizar a análise qualitativa do risco | 0 | 0 |
| **Planeamento** | Riscos | Realizar a análise quantitativa do risco | 0 | 0 |
| **Planeamento** | Riscos | Planear a resposta ao risco | 0 | 0 |
| **Execução** | *Stakeholders* | Gerir *stakeholders* | 4 | 4 |
| **Início** | *Stakeholders* | Identificar os *stakeholders* | 4 | 4 |
| **Monitorização e controlo** | *Stakeholders* | Controlar o envolvimento dos *stakeholders* | 4 | 4 |
| **Planeamento** | *Stakeholders* | Planear a gestão dos *stakeholders* | 4 | 4 |
| **Planeamento** | Tempo | Planeamento da gestão do Tempo | 3 | 3 |
| **Planeamento** | Tempo | Definir as atividades | 4 | 4 |
| **Planeamento** | Tempo | Sequenciar as atividades | 3 | 3 |
| **Planeamento** | Tempo | Estimar os recursos das atividades | 4 | 4 |
| **Planeamento** | Tempo | Estimar duração das atividades | 2 | 2 |
| **Planeamento** | Tempo | Desenvolver cronograma | 2 | 2 |