

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Paula Lúcia Peixeira Costa

GESTpro-SOFT: Análise Comparativa sobre Métodos de Gestão de Projectos de Software no Mercado

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Paulo Tomé



Outubro de 2012

RESUMO

As organizações têm vindo usar cada vez mais os projetos para atingirem os seus objetivos estratégicos enquanto o ambiente socioeconómico se torna progressivamente mais complexo, incerto e competitivo.

A implementação de projetos tendencialmente mais complexos, aliados a utilizadores mais exigentes e ao aparecimento de novas tecnologias, conduziu à necessidade de melhores métodos de trabalho. Assim, nas mais diversas áreas do conhecimento, nomeadamente nos processos de desenvolvimento de software, têm vindo a ser adotadas práticas de Gestão de Projetos, estas práticas contribuem para a redução do impacto de atrasos e na gestão das mudanças que ocorrem durante a execução de um projeto assumindo um papel determinante no sucesso dos projetos.

A escolha adequada de métodos de gestão de projetos, ajustados às características do projeto em questão, pretendem evitar os maus resultados devido à falta de planeamento e ou desadequadas técnicas usadas na abordagem ou no desenvolvimento, ou pior ainda, a finalização do projeto com a obtenção de resultados com elevadíssimos custos.

Atualmente, no domínio do desenvolvimento de software, não existem métodos de gestão de projetos de aplicabilidade universal. Existem sim diversos métodos reconhecidos, que podem ser adotadas de forma total ou apenas parcialmente, para o cumprimento de projetos de desenvolvimento de software.

Estruturalmente, os métodos subdividem-se em disciplinares e ágeis, os primeiros caracterizam-se como abordagens tradicionais da gestão de projetos ao contemplar planeamento e acompanhamento de custos e recursos. Já os métodos ágeis são uma adaptação rápida às mudanças de realidade, sem a preocupação com a predição mas reagindo às necessidades imediatas.

Apresentar um enquadramento sobre a gestão de projetos, a caracterização dos principais métodos de apoio à gestão de projetos de software e a diferenciação entre os métodos apresentados de modo a identificar o melhor método a aplicar numa situação específica são os objetivos deste trabalho limitado no entanto ao âmbito do desenvolvimento de software.

ABSTRACT

Organizations have been increasingly use projects to achieve their strategic objectives while the socio-economic environment becomes increasingly complex, uncertain and competitive.

The projects implementation tend to be more complex to the need for better working methods. Thus, in several areas of knowledge, particularly in the processes of software development, practices of Project Management have been adopted. These practices play a key role in the success of the same, because with the increasing complexity of projects, users tend to be more demanding with the emergence of new technologies, it's increasingly necessary to efficient project management to ensure the success, reduce the impact of delays and manage the changes that occur during the project execution.

The choice of appropriate methods of project management, tailored to the project characteristics, want to avoid the bad results due to lack of planning or inadequate techniques used in the development or approach, or even worse, the completion of the project with getting results with very high costs.

Currently, the field of software development, there are no methods of managing projects of universal applicability. There are so many recognized methods which can be adopted in whole or partially, for the fulfillment of software development projects.

Structurally, the methods are subdivided into disciplinary and agile, the first characterized as traditional approaches to project management covering planning and monitoring of costs and resources. Already agile methods are rapid adaptation to changing reality, without concern for the prediction but reacting to immediate needs.

Presenting the characterization of the main methods to support the management of software projects and the differentiation between the methods in order to identify the best method to apply in a specific situation are the goals of this work however limited the scope to software development.

PALAVRAS CHAVE

Gestão de Projetos

Modelos

Métodos

Desenvolvimento de Software

Processos de Desenvolvimento de Software

Métodos Ágeis

Análise de Mercados

KEY WORDS

Project Management
Models
Methods
Software Development
Software Development Processes
Agile Methods
Benchmarking

AGRADECIMENTOS

O desfecho deste trabalho traduz a conclusão de mais uma etapa acadêmica, esta etapa que envolveu muito esforço e alguns sacrifícios nos últimos anos com a frequência do Mestrado em Sistemas e Tecnologias da Informação para as Organizações (MSTIO), a par com afazeres profissionais exigentes e sempre tentando não descuidar nas relações familiares e pessoais. A todos os meus familiares e amigos, obrigado!

Não posso deixar de referir o particular agradecimento ao Professor Doutor Paulo Tomé, além de professor em várias disciplinas deste curso, presidente do Conselho Científico da ESTV e orientador deste projeto, agradeço pela sua paciência, compreensão, apoio, conhecimento, orientações e persistência.

ÍNDICE GERAL

ÍNDICE GERAL	xi
ÍNDICE DE FIGURAS	xiii
ÍNDICE DE QUADROS	xv
ABREVIATURAS E SIGLAS	xvii
1. Introdução	1
1.1 Enquadramento do tema	1
1.2 Objetivos da dissertação	2
1.3 Delimitações e limitações do estudo	3
1.4 Interesses e Motivação	4
1.5 Estrutura da dissertação	4
2. Enquadramento	5
2.1 Origem e conceito de Projeto	5
2.2 Definição de Gestão de Projetos	7
2.3 Sucessos e Fracassos de Projetos	9
2.4 Maturidade em Gestão de Projetos	11
2.5 Metodologias	12
2.6 Processos de Desenvolvimento de Software (PDS)	14
3. Apoio ao Processo de Desenvolvimento de Software	17
3.1 Modelos Genéricos do PDS	18
3.1.1 Modelos Prescritivos	18
3.1.2 Modelo Iterativo	25
3.1.3 Modelos evolutivos	27
3.1.4 Modelo RUP	32
3.1.5 Critérios para Seleção do Modelo	36
3.2 Métodos Ágeis	38
3.2.1 Manifesto ágil	38
3.2.2 Métodos orientados a objetos	40

3.2.3	Fatores Críticos de Sucesso para o Desenvolvimento Ágil	81
3.3	Sistemas Integrado de orientação do PDS.....	82
3.3.1	Microsoft Solutions Framework (MSF).....	82
3.4	Desenvolvimento Tradicional <i>versus</i> Desenvolvimento Ágil.....	87
4.	Análise dos Métodos e de Benchmarking.....	91
4.1	Análise dos Métodos Ágeis.....	91
4.2	Análise Benchmarking	97
4.2.1	Conceito de Benchmarking.....	97
4.2.2	Benchmarking em Gestão de Projetos	99
5.	Considerações finais	105
5.1	Análise Crítica ao Trabalho.....	105
5.2	Trabalho Futuro	106
5.3	Conclusões.....	106
	Referências.....	109
	Anexo 1 – MANIFESTO PARA O DESENVOLVIMENTO ÁGIL DE SOFTWARE.....	115
	Anexo 2 – PRÍNCÍPIOS SUBJACENTES AO MANIFESTO ÁGIL.....	117

ÍNDICE DE FIGURAS

Figura 1 Grupo de processos do guia PMBOK (adaptado de (<i>A guide to the Project Management Body of Knowledge (PMBOK Guide)</i> , 2008)	8
Figura 2 Áreas do conhecimento do guia PMBOK (adaptado de (<i>A guide to the Project Management Body of Knowledge (PMBOK Guide)</i> , 2008).....	8
Figura 3 – modelo em cascata (adaptado de (Donato, 2010)).....	19
Figura 4 – modelo em cascata revisto (adaptado de (Leite, 2000)).....	22
Figura 5 Modelo incremental (extraído de (Martins, 2002)).....	22
Figura 6 Desenvolvimento de incrementos (adaptado de (Martins, 2002))	23
Figura 7 Modelo RAD (adaptado de (Jenny, 2012)).....	24
Figura 8 Modelo Iterativo (extraído de (Martins, 2002))	26
Figura 9 Modelo espiral (extraído de (Tomás, 2010))	28
Figura 10 Prototipagem (extraído de (Zadrozny, 2006)).....	30
Figura 11 Arquitetura do RUP (extraído de (Portier, 2007))	33
Figura 12 Estrutura básica do RUP	34
Figura 13 ciclo de vida ASD (extraído de (Rodrigo, 2011)).....	42
Figura 14 ciclo de vida do AUP (extraído de (Scott 2009)).....	45
Figura 15 Linha do projeto (extraído de (Scott 2009)).....	46
Figura 16 Distribuição dos métodos da família <i>Crystal</i> (extraído de (Cockburn, 2008))	47
Figura 17 Ciclo de vida da família <i>Crystal</i> (extraído de (Brito, 2011))	49
Figura 18 Princípios fundamentais do DSDM (extraído de (DSDM Consortium, 2008)).....	51
Figura 19 Ciclo de vida de um projeto com DSDM (extraído de (DSDM Consortium, 2008))	53
Figura 20 Síntese do método YP (extraído de (Garcia & et al, 2006))	55
Figura 21 – ciclo de vida de um projeto típico XP (extraído de (Don Wells, 2009))	65
Figura 22 Fases do método XP (extraído da pág. 18 (Warden, 2007))	65
Figura 23 XP principais tarefas e periodicidade de ocorrência (extraído de (Audy, 2012)).....	66
Figura 24 Organização do trabalho e foco do conteúdo no Openup	74
Figura 25 Ciclo de Vida de cada interação no Scrum (retirado de (Sadhan, 2010))	78
Figura 26 Modelo de equipa com os papéis mais importantes (extraído de (Microsoft, 2006))	85
Figura 27 Modelo de processo do MSF com as fases e marcos importantes (extraído de (Microsoft, 2006))	86
Figura 28 Abordagem tradicional e ágil (extraído de (Sousa, 2011))	88
Figura 29 Aspetos abordado no Estudo de <i>Benchmarking</i> em análise (extraído de (PMI Rio, 2012))	101

ÍNDICE DE QUADROS

Quadro 1 Critérios a ter em conta na seleção de um modelo	37
Quadro 2 Princípios do XP	59
Quadro 3 Vantagens e desvantagens do XP	67
Quadro 4 Análise comparativa dos métodos ágeis	93
Quadro 5 Continuação da análise comparativa dos métodos ágeis	94

ABREVIATURAS E SIGLAS

Nesta dissertação são utilizadas abreviaturas de designações comuns apenas apresentadas aquando da sua primeira utilização:

ASD	Adaptive Software Development
AUP	Agile Unified Process
CMMI	Capability Maturity Model Integration
DSDM	Dynamic System Development Method
EPF	Eclipse Process Framework
ESSUP	Essential Unified Process
ESTGV	Escola Superior de Tecnologia e Gestão de Viseu
FDD	Feature Driven Development
IBM	International Business Machines Corporation
IPV	Instituto Politécnico de Viseu
JAD	Joint Application Development
LSD	Lean Software Development
MSF	Microsoft Solutions Framework
MSF4ASD	Microsoft Solutions Framework para Agile Software Development
MSF4CMMI	Microsoft Solutions Framework para Capability Maturity Model Integration
Openup	Open Unified Process
OPM3	Organizational Project Management Maturity Model
PDS	Processo de Desenvolvimento de Software
PMBOK	Project Management Body of Knowledge
PMI	Project Management Institute
PMMM	Project Management Maturity Model
PMO	Project Management Office
PO	Product Owner
Prince2	Projects in Controlled Environment
RAD	Rapid Application Development
RUP	Rational Unified Process
SEI	Software Engineering Institute
SI	Sistemas de Informação
SM	Scrum Master
ST	Scrum Team
TI	Tecnologias de Informação
UML	Unified Modeling Language
XP	Extreme Programming
YP	EasyProcess

1. Introdução

Com o proliferar do uso de sistemas informáticos assim como o aumento de empresas que se dedicam ao desenvolvimento de software, torna-se fulcral que se faça mais e melhor nesta área. Atualmente é dada cada vez mais importância à gestão de projetos no apoio do desenvolvimento de software para uma melhor otimização dos recursos disponíveis e de modo a evidenciar vantagem competitiva perante outras empresas.

O presente capítulo pretende, através das suas secções constituintes, fazer o enquadramento ao tema do trabalho, expor os objetivos a cumprir assim como identificar as barreiras deste trabalho através a enumeração das delimitações e limitações do estudo neste documento apresentado.

1.1 Enquadramento do tema

Embora nem sempre de forma formal, a gestão de projetos sempre acompanhou a ação humana desde os primórdios da civilização, quando as pessoas começaram a coordenar as suas atividades para a construção de abrigos, navios, construções como as pirâmides e outros (Teixeira, 2009). Todas estas grandes obras envolveram o planeamento do trabalho, a divisão de tarefas, o trabalho em equipa, liderança, coordenação, características que permanecem na gestão de projetos dos tempos atuais.

O cenário de negócios atual exige às organizações uma maior velocidade, flexibilidade e consistência na implantação das suas estratégias para sobrevivência e crescimento (Serghei

Florice & et al, 2011). Para a execução destas estratégias são necessárias etapas que aliadas a recursos (humanos, financeiros e materiais) com as atividades de rotina das empresas aumentam a complexidade da gestão empresarial. Neste contexto, pode considerar-se que a gestão de projetos assume um papel estratégico nas organizações.

Para as organizações adotarem facilmente e de forma adequada a gestão de projetos estão constantemente a ser evoluídas as técnicas e métodos associados, assim como uma maior especialização tendo em conta as áreas de atuação de acordo com as atividades de cada organização.

O investimento em tecnologias e Tecnologias de Informação (TI) tem crescido significativamente ao longo dos últimos anos, introduzindo mudanças significativas em várias áreas de atividade. Diferentes tipos de sistemas de *software* e diferentes tipos de projetos foram surgindo para dar resposta às necessidades das organizações em constante evolução (Gonçalves, 2009). Para responder a esta necessidade é preciso conhecer-se bem, cada um dos sistemas de *software*, e saber como melhor proceder ao longo da implementação do sistema. Assim, e de acordo com a bibliografia em geral, além das competências técnicas de quem vai implementar o sistema é também necessário definir os melhores métodos e técnicas de gestão para que cada projeto se execute com sucesso e preferencialmente de acordo com o seguinte trinómio: tempo, custo e qualidade.

A implementação de projetos, cada vez mais complexos, conduziu à necessidade de melhores métodos de trabalho, entenda-se, no âmbito do processo de desenvolvimento de *software*, práticas ou métodos de gestão de projetos.

A escolha adequada de métodos de gestão de projetos, ajustados às características do projeto em questão, pretendem evitar os maus resultados devido à falta ou ao indevido planeamento e também a desadequados métodos usados no desenvolvimento do produto. Assim as práticas e métodos assumem um papel determinante no sucesso dos projetos.

1.2 Objetivos da dissertação

Como não existem métodos de gestão de projetos de aplicabilidade universal, ou seja, não existe uma “receita universal”, então, neste trabalho é apresentado um estudo de diversos métodos de gestão de projetos e, juntamente com a análise de um estudo de mercado, pretende-se ajudar a responder às seguintes questões:

→ Como decidir qual o método a aplicar numa situação específica?

→ Qual a importância, atribuída pelas empresas, da aplicação da gestão de projetos, nomeadamente as empresas de desenvolvimento de *software*?

Portanto, é propósito deste trabalho a exposição de um estudo aprofundado acerca dos diversos métodos de gestão de projetos no âmbito do desenvolvimento de *software*, com a diferenciação entre os vários métodos ágeis apresentados, e ainda uma análise de *benchmarking* para a identificação da utilização das práticas de gestão de projetos.

1.3 Delimitações e limitações do estudo

O estudo apresentado neste documento é restrito ao âmbito da engenharia de *software*, mais concretamente às atividades de desenvolvimento de *software*, tendo em conta as organizações de TI, e todas as práticas de gestão de projetos relacionadas com esta atividade.

A análise e interpretação de um estudo elaborado anualmente, desenvolvido pelo Project Management Institute (PMI) Brasil e, de acordo com o próprio PMI Brasil, este estudo permite a análise do impacto da gestão de projetos nas organizações. A análise deste estudo apresenta-se como uma solução de contingência em detrimento da distribuição de um questionário alusivo às práticas de gestão de projetos em empresas de referência na área de TI em atuação em Portugal, que permitiria a recolha de informação para uma análise real da adoção destas práticas neste país. Esta limitação deve-se à indisponibilidade para o efeito aliada à mentalidade reservada das organizações portuguesas juntamente com a restrita informação publicada por atores e/ou organizações portuguesas sobre este assunto.

Para a apresentação da descrição dos métodos foi necessário um levantamento e estudo antecipado da bibliografia disponível sobre o tema. Como o assunto é ainda recente, alguns tópicos abordados possuem uma escassa bibliografia editada, tendo sido necessário recorrer à pesquisa de artigos e trabalhos escritos no meio académico. Desta forma, a consulta de bibliografia na internet constituiu um recurso de pesquisa importante para o desenvolvimento deste trabalho.

1.4 Interesses e Motivação

A motivação principal para a elaboração deste trabalho, relacionado com a temática gestão de projetos, prende-se com o gosto particular pelo tema aliado ao interesse em aprofundar os conhecimentos na área de gestão de projetos de software.

Serve o presente trabalho para documentar o autoestudo realizado relativamente ao tema em questão associado ao conhecimento assimilado no decorrer do percurso académico.

1.5 Estrutura da dissertação

Como referido anteriormente esta dissertação pretende apresentar um estudo dos diversos métodos de apoio ao processo de desenvolvimento de software através da descrição dos vários métodos existentes.

O primeiro, e atual capítulo, pretende contextualizar e introduzir o âmbito do trabalho desenvolvido, nomeadamente os interesses e motivação associados à sua realização.

O segundo capítulo pretende descrever o enquadramento do tema relativamente à importância do mesmo, à sua origem e os principais conceitos associados como sucesso, fracasso e maturidade na gestão de projetos.

O terceiro capítulo contempla o estudo propriamente dito dos métodos, desde a descrição dos modelos genéricos que deram origem aos métodos usados para aplicabilidade no processo de desenvolvimento de software, dos mais tradicionais aos referidos como ágeis. Além da descrição dos modelos e métodos que apoiam o processo de desenvolvimento de software é apresentado o Manifesto Ágil que culmina com a mudança de paradigma neste âmbito.

O quarto capítulo aborda uma análise comparativa dos métodos ágeis apresentados no capítulo anterior e expõe o conceito de Benchmarking, a sua função e a análise de um estudo disponibilizado publicamente com informação da adoção da gestão de projetos e outras informações de interesse no Brasil.

Por fim, o quinto capítulo apresenta uma reflexão sobre o trabalho apresentado, enaltecendo os prós e os contras destes tipos de métodos através das conclusões do estudo e sugestões para trabalho futuro, onde são identificadas algumas linhas orientadoras para uma possível continuidade do trabalho.

2. Enquadramento

Existem alguns conceitos importantes relacionados com o tema de gestão de projetos, estes conceitos são essenciais para um melhor entendimento do estudo apresentado no capítulo seguinte.

O presente capítulo pretende descrever a importância do tema - Gestão de Projetos - primeiramente através da origem do tema, a definição do conceito de “projeto” e por sequente a descrição de gestão de projetos propriamente dita com a descrição detalhada dos principais conceitos relacionados com a gestão de projetos. É ainda apresentada a descrição do contexto dos métodos de gestão de projeto e a sua contribuição na eficiência da gestão de projetos de desenvolvimento de software.

2.1 Origem e conceito de Projeto

O termo projeto provém do latim *projectu*, “lançado”, no particípio passado de *projicere* que significa “lançar para a frente” (Infopédia, 2012), como ato ou intenção de fazer algo.

A execução de projetos vem já desde os primórdios da civilização. Tais como as caçadas organizadas e a construção de grandes obras como as grandes pirâmides construídas no antigo Egito e a Grande Muralha da China construída durante a China Imperial (Warrillow, 2011). Guardadas as devidas proporções, em relação à época em que foram realizadas essas obras, podem ser comparadas com atuais projetos como o Projeto Apollo, que compreendeu um conjunto de missões espaciais coordenadas pela NASA que permitiu ao Homem chegar à Lua.

Mas os projetos não se restringem apenas a empreendimentos grandiosos podem ser também pequenos projetos do dia-a-dia de cada indivíduo.

Ao longo da vida, todas as pessoas são confrontadas com diversas situações em que é necessário recorrer à elaboração de projetos, dos mais simples aos mais complexos, por vezes projetos do dia-a-dia de cada indivíduo que dispensam um tratamento formal.

Um projeto pode-se definir como um conjunto de atividades, com a utilização de diversos recursos, executadas para levar a cabo um determinado objetivo.

Por definição e de acordo com Rodão um projeto é uma iniciativa que é única de alguma forma, seja no produto final pretendido, no cliente do projeto, na localização, nas pessoas envolvidas ou em outro fator (Victor Sequeira Rodão, 2007). Isto diferencia projetos de operações regulares de uma qualquer empresa, por exemplo a produção em série de peças é uma operação, mas por outro lado a criação de um móvel por encomenda já é um projeto.

Um projeto tem um fim bem definido, ou seja, tem um objetivo claro que idealmente quando atingido caracteriza o final do projeto. Portanto a criação de um novo negócio pode não ser considerado um projeto, com a ideia de um novo produto ou serviço para lançar no mercado são necessários recursos e várias atividades para o desenvolvimento desse potencial negócio, nomeadamente a elaboração de um plano de negócios a conter informações do próprio produto/ serviço, sobre as forças do mercado que afetarão esse negócio (clientes, concorrência, fornecedores, etc), a análise de oportunidades e ameaças, pontos fortes e pontos fracos, a criação de um plano financeiro e de marketing e tudo o necessário para mostrar o diferencial desse produto/ serviço com os restantes no mercado. Mas a criação de um novo negócio, salvo exceções, são feitos para durar indefinidamente e não para terem um final num determinado momento.

Referidas as definições anteriores pode considerar-se que a construção de uma casa é um projeto, também o desenvolvimento de software, a construção de um móvel por encomenda, a implantação de uma nova linha de produção numa fábrica, a realização de uma viagem, escrever um livro, criar um documento, a construção de grandes obras como pontes, barragens, edifícios, fábricas, aeroportos, estradas, etc. Portanto para fazer parte da lista de projetos é necessário cumprir com duas características já mencionadas: todos têm que ter um objetivo bem definido e ser, de alguma forma, sempre único.

O objetivo associado a um projeto tem tipicamente três vertentes (António Miguel, 2006):

- Diz respeito a um conjunto de especificações de natureza técnica ou legal que o projeto tem que cumprir na sua execução e no seu resultado final: o “**âmbito**” e também designado de “qualidade” do projeto;
- Diz respeito a um conjunto de datas em que parte, ou a totalidade, das atividades devem estar executadas: os “**prazos**” do projeto;
- O cumprimento de critérios económicos quanto à execução do projeto ou à operação do seu resultado final: os “**custos**” do projeto.

2.2 Definição de Gestão de Projetos

Administrar, acompanhar, coordenar ou gerir um projeto é a aplicação de técnicas, conhecimento e habilidades para garantir que um projeto tenha sucesso. E gerir um projeto envolve todas as tarefas desde iniciá-lo até finalizá-lo, passando por várias etapas de planeamento, execução e atividades de controlo.

Os conceitos mais conhecidos de gestão de projetos são definidos pelo *Project Management Institute* (PMI), uma das maiores organizações de profissionais de gestão de projetos do mundo, mais disseminada na América e na Ásia, tem como principais atividades: a difusão do conhecimento em gestão de projetos como as certificações para profissionais em gestão de projetos — *Project Management Professional* e *Certified Associate in Project Management* — e a publicação de padrões globais de gestão de projetos, programas e portfólio, sendo a mais popular delas o **Guia PMBOK - Project Management Body of Knowledge**. Editado na forma de livro, o Guia PMBOK está atualmente na quarta edição de 2008 e traduzido oficialmente para diversos idiomas, excluindo o português de Portugal. As edições anteriores foram publicadas nos anos de 1996, 2000 e 2004. Mas, já em 2012 o PMI publicou, em 17 de fevereiro, o *PMBOK® Guide 5th Edition Exposure Draft*, trata-se de uma versão de avaliação, não se trata ainda da versão final. Foi disponibilizada para revisão, críticas e sugestões da comunidade de gestão de projetos até 20 de Março de 2012, a fim de serem avaliados os contributos pela equipa responsável e, de acordo com a relevância, realizados os ajustes para a versão definitiva que está planeada ser publicada em Dezembro do corrente ano. O PMBOK formaliza diversos conceitos de gestão de projetos, como a própria definição de projeto e do seu ciclo de vida. Também identifica na comunidade de gestão de projetos um

conjunto de conhecimentos grandemente reconhecidos como boas práticas, aplicáveis à maioria dos projetos na maior parte do tempo.

Segundo o PMBOK de 2008, um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. A sua natureza temporária indica um início e um fim definidos. O fim é alcançado quando os objetivos tiverem sido atingidos ou quando se concluir que esses objetivos não serão ou não poderão ser atingidos e o projeto for encerrado, ou quando o mesmo não for mais necessário. Os processos propostos pelo PMBOK para a gestão de projetos são apresentados como elementos distintos, mas na prática eles sobrepõem-se e interagem. Possui cinco grupos de processos: início, planeamento, execução, controlo e encerramento conforme representado na Figura 1.

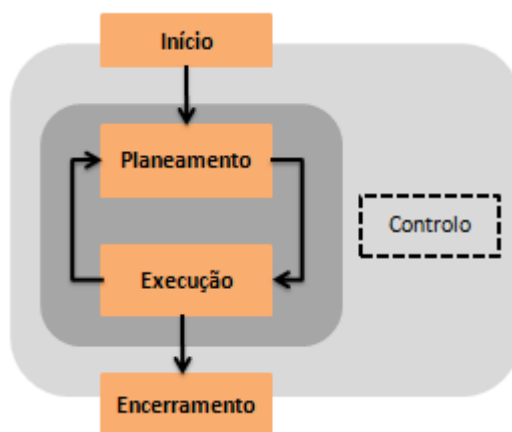


Figura 1 Grupo de processos do guia PMBOK (adaptado de *(A guide to the Project Management Body of Knowledge (PMBOK Guide), 2008)*)

Na mesma edição do Guia PMBOK, na página 6, é indicado que a gestão de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de responder aos seus requisitos.

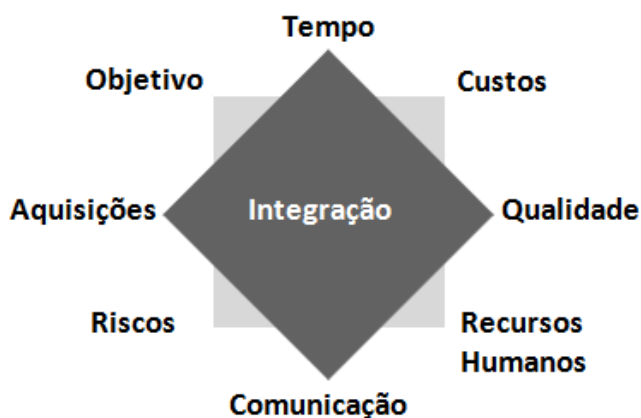


Figura 2 Áreas do conhecimento do guia PMBOK (adaptado de *(A guide to the Project Management Body of Knowledge (PMBOK Guide), 2008)*)

O PMBOK possui um total de 42 processos distribuídos em nove áreas de conhecimento, conforme a Figura 2, não é considerada uma metodologia, porque descreve processos de alto nível sem especificamente detalhar como devem ser implantados. No entanto, é considerada uma ótima base para auxiliar na elaboração de uma metodologia para a organização.

A par do PMBOK foi desenvolvida na Inglaterra uma metodologia designada de Prince2 (Projects in Controlled Environment), a diferença básica entre PMBOK Guide e PRINCE2 é que enquanto o primeiro é uma base de conhecimentos e boas práticas de gestão de projetos, o segundo é uma abordagem estruturada, com processos, papéis e responsabilidades bem definidos, que orienta o gestor e a equipa do projeto na condução do projeto. Ou seja, o guia PMBOK mostra “o que” é necessário fazer e o PRINCE2 mostra “como” fazer.

2.3 Sucessos e Fracassos de Projetos

Como referido na secção anterior, por definição um projeto pode ser considerado um empreendimento temporário, possui um ciclo de vida e objetivos definidos, mas a avaliação do sucesso de um projeto não pode apenas passar pela simples análise deste objetivos, seguidamente serão descritos os fatores que contribuem para o sucesso e fracasso de um projeto.

Inicialmente a gestão de projetos considerava que o sucesso de um projeto era medido apenas em termos técnicos, ou seja, o produto ou serviço era avaliado como adequando ou inadequado, mas essa não seria uma avaliação totalmente eficaz.

No contexto de gestão de projetos o sucesso normalmente está diretamente relacionado com o facto de se alcançar algo que se desejava e que foi planeado, ou seja, dentro do orçamento, dentro do prazo e adequado aos objetivos propostos.

Atualmente, para uma definição mais adequada de sucesso de um projeto, pode considerar-se a realização dentro do prazo, com o orçamento e qualidade previstos e se respondeu às expectativas do cliente e dos principais interessados, como os designados “patrocinadores” (*stakeholders*), equipa do projeto, entidades reguladores e ambientais.

Por outro lado, o fracasso decorre do contrário, ou seja, se os objetivos esperados não foram alcançados.

Determinar o fracasso ou o sucesso de um projeto depende da definição de padrões de desempenho que permitam comparar os resultados obtidos em relação às metas delineadas.

De acordo com o Standish Group (The Standish Group, 2012) os principais fatores que contribuem para o fracasso dos projetos são:

- Falta de envolvimento do cliente ou utilizadores e dos *stakeholders*;
- Especificações incompletas de requisitos;
- Alterações de requisitos ao longo do projeto;
- Falta de competências tecnológicas;
- Recursos escassos;
- Objetivos pouco claros do projeto;
- Expectativas e/ ou prazos irreais;
- Novas tecnologias e processos.

Para ajudar a perceber o comportamento da maioria dos projetos de TI, ou seja, se estão ou não a alcançar o sucesso, para tal são publicados anualmente relatórios, designados por *CHAOS Reports*, pelo Standish Group. O Standish Group, uma empresa localizada em Massachusetts, nos Estados Unidos da América, publica desde 1994 um estudo chamado de *CHAOS Report*. Trata-se de um amplo levantamento envolvendo milhares de projetos na área de TI. O Standish Group é composto por vários especialistas com experiência nessa área e, com base nestes estudos, pretendem identificar:

- As falhas dos projetos de software;
- Os maiores fatores que influenciam na falha de projetos de software;
- Os pontos-chave que podem reduzir estas falhas.

O *CHAOS report* revela as taxas de sucesso e fracasso dos projetos na área das TI e, com base neste estudo, classifica os projetos em três tipos:

- **Sucesso ou bem-sucedido** – quando um projeto é entregue no prazo e custos acordados e com as funcionalidade previstas;
- **Sucesso parcial ou comprometido** – quando o projeto é entregue mas sem responder totalmente ao custo, esforço ou âmbito;
- **Fracasso ou mal sucedido** – quando um projeto é cancelado ou não utilizado.

Com os dados deste relatório é possível analisar quais os fatores que influenciaram negativamente os resultados, ou seja, a que se devem os desvios dos projetos. Estes fatores

deverão ser tomados em atenção de modo a não serem cometidos os mesmos erros em novos projetos.

Atualmente os dados do *CHAOS Report* estão entre os mais utilizados para quantificar a “crise do software” e servem normalmente como referência na maioria das apresentações relacionadas com gestão de projetos e os seus métodos.

2.4 Maturidade em Gestão de Projetos

Uma empresa imatura caracteriza-se pela improvisação dos seus processos e pelo não estabelecer das devidas ligações entre as diversas áreas de conhecimento. Em contrapartida, as organizações maduras têm capacidade para a execução dos seus processos de forma previsível e padronizada.

A necessidade de maior maturidade na gestão de projetos é uma constatação que tem vindo a ganhar mais importância nos últimos anos, pois as organizações necessitam amadurecer no conhecimento e na capacidade de gestão de projetos para sua prossecução no mercado.

No entanto, a escolha das melhores práticas com a finalidade de obter maior sucesso com os projetos, por si só não é garantia de estratégia de sucesso. As práticas e técnicas de gestão de projetos são reconhecidas por muitas organizações de diversas indústrias como competências essenciais que trazem benefícios para os negócios. Estas competências são medidas através de processos de estudos, como *Benchmarking* e modelos comparativos.

Uma boa-prática é um conceito de gestão que defende a existência de determinadas técnicas e processos que são mais eficientes que outros para atingir determinados resultados. Pode assumir-se que, com os processos adequados, um determinado resultado pode ser obtido com menos dificuldades e com a máxima eficiência. A utilização de um modelo de maturidade é um método que permite avaliar se os processos usados, e a forma como são utilizados, estão de acordo com as boas-práticas em vigor.

Entre os modelos de maturidade existentes destacam-se os seguintes:

- **PMMM (Project Management Maturity Model)** - proposto pelo Dr. Harold Kerzner e alinhado ao PMBOK e ao PMI. Kerzner em 2006 considera que a avaliação do grau de maturidade é baseada em cinco níveis: 1 – Linguagem comum; 2 – Processos Comuns; 3 – Metodologia única; 4 – *Benchmarking* e 5 – Melhoria contínua.

- **OPM3 (Organizational Project Management Maturity Model)** concebido pelo PMI considera que é composto por conhecimento, avaliação e aperfeiçoamento. O nível de maturidade é atribuído por uma avaliação que possui quatro etapas (padronização, medição, controlo e melhoria contínua), possui três domínios (projeto, programa e portfolio) e é baseada em três elementos: conhecimento, desenvolvimento e avaliação.
- **CMMI (Capability Maturity Model Integration)** criado pelo Software Engineering Institute (SEI), considera que o CMMI está dividido em cinco níveis: Incompleto (*Ad-hoc*), executado, gerido e o institucionalizado. Estes níveis estão divididos em áreas de processo e estas em dois conjuntos de metas: específicas e genéricas, sendo que as metas genéricas ainda se dividem em quatro categorias enquanto as específicas focam o negócio da empresa.

2.5 Metodologias

Uma metodologia, por definição, significa o estudo dos métodos, é na prática como que uma “receita”, para as etapas a serem seguidas num determinado processo, e são fundamentais para o desenvolvimento dos projetos. Os métodos necessitam ser bem aplicados, ou seja, de acordo com as necessidades de cada empresa e de cada projeto em si, pois não existe uma receita universal para todos.

Um modelo tem como objetivo estabelecer, com base em estudos históricos e conhecimento operacional, um conjunto de boas-práticas, que devem ser utilizadas para obter um determinado resultado, podendo nem sempre ser a melhor opção. Deve ficar a cargo de cada organização estabelecer se determinada prática é, ou não, a melhor e se deve, ou não, ser utilizada.

Em regra as organizações possuem um planeamento estratégico definido, com missão e visão de onde querem chegar e, neste contexto, os projetos são peças fundamentais para que os objetivos do planeamento estratégico sejam alcançados. Gerir projetos de forma eficiente e eficaz é um fator crítico de sucesso para alcançar os objetivos planeados.

Dado que um projeto é caracterizado um empreendimento temporário, com delimitação de datas, cujo objetivo é criar ou aperfeiçoar um produto ou serviço único. Então, gerir um projeto é atuar de forma a atingir os objetivos propostos dentro dos parâmetros previamente estabelecidos.

Numa organização o sucesso da gestão de projetos depende principalmente das pessoas, da estrutura organizacional e dos métodos de gestão dos projetos. As pessoas são importantes devido às suas capacidades, experiências, conhecimentos e atitudes. A estrutura organizacional tem influência conforme a sua política, a sua estrutura física e os planos de incentivos. Já os métodos, são importantes na definição de processos, ferramentas, uso de técnicas e instrumentos na gestão de projetos.

Como o pretendido com cada projeto é o seu sucesso, então é fundamental o uso de um método, muito bem definida, executada e controlada para a gestão efetiva dos projetos da organização.

De acordo com a obra “Gestão de Projetos de software”, de António Miguel pode depreender-se que na definição de um método em gestão de projetos deve considerar alguns aspetos importantes, destacam-se os seguintes:

- **Origem nas melhores práticas do mercado** – um método com os respetivos processos, procedimentos, fluxogramas, instrumentos e ferramentas, deve estar baseado em conhecimento acumulado de profissionais do mercado. A constante inclusão de novas informações deverão contribuir para a melhoria contínua do método;
- **Ser ajustável às necessidades e realidade da organização** – as empresas passam por estágios em maturidade em gestão de projetos, além de que cada empresa possui a sua própria cultura e experiência em gestão de projetos, portanto um método deve ser ajustado ao modelo de gestão e de processos de cada empresa;
- **Ser utilizável, prática e não burocrática** – um método deve auxiliar o processo de planeamento e controlo de projeto, para que os gestores de projeto e as suas equipas possam ter o tempo, a criatividade e preocupações voltadas para o projeto em si e não na forma de como o gerir.

É inferido com base nos conceitos anteriores que um método bem definido de gestão de projetos pode trazer inúmeras vantagens para uma organização, nomeadamente:

- **Melhoria na comunicação** – com a padronização de processos, documentos, instrumentos e ferramentas deverá melhorar a comunicação dentro de cada projeto, com os restantes projetos da empresa e entre estruturas organizacionais.
- **Aumento de eficiência e produtividade** – com a condução de projetos que utilizam um bom método, baseado em princípios, técnicas, ferramentas e capacidades, os gestores de projetos poderão aumentar a efetividade da gestão, alcançando melhores resultados. Da

mesma forma a equipa poderá ser motivada, liderada e controlada para garantia de maior produtividade;

- **Planeamento detalhado e controlo adequado** – permitem tomadas de decisões mais rápidas e acertadas, baseadas em informações e indicadores gerados pelo controlo do projeto. Proporcionará uma redução nos riscos de fracasso do projeto e ganhos de qualidade;

- **Padronização de processos e de documentos** – na utilização de processos conhecidos e padronizados os projetos possuirão registos históricos, lições aprendidas e documentos apropriados, o que permitirá uma gestão do conhecimento dos projetos da organização.

A implementação de um método para a gestão de projetos é em si um processo, deverá ser adotado de forma gradual e deve-se estimular a utilização do método com o foco nos benefícios a longo prazo, tanto para as equipas como para a própria organização.

No entanto, o termo metodologia é bastante controverso nas ciências em geral, e na Engenharia de Software em particular. Muitos autores tratam metodologia e método como sinónimos, porém seria mais adequado dizer que uma metodologia envolve princípios filosóficos que guiam uma gama de métodos que utilizam ferramentas e/ ou práticas diferenciadas para realizar algo (Wikipédia, 2012). Para diminuição das ambiguidades ao longo deste trabalho, de agora em diante, será usado o termo “método”.

2.6 Processos de Desenvolvimento de Software (PDS)

O Processo de Desenvolvimento de Software (PDS) pode ser descrito como um conjunto de atividades, parcialmente ordenadas, com a finalidade de obter um produto de software e o resultado deste processo reflete a forma como o processo foi conduzido.

Independentemente da dimensão e complexidade dos projetos de desenvolvimento de software são envolvidos dois processos essenciais: a análise e a codificação. Este tema é estudado de forma aprofundada no âmbito da **Engenharia de Software**, que compreende a área da computação que abrange a especificação, desenvolvimento e manutenção de sistemas de software, com a aplicação de tecnologias e práticas de gestão de projetos e outras disciplinas, visando a organização, a produtividade e a qualidade (Roger Pressman, 2011).

De uma forma geral, apesar da existência de vários modelos de processos para o desenvolvimento de software e depois da consulta da bibliografia pode considerar-se que existem algumas atividades fundamentais comuns a todos, nomeadamente:

- **Especificação de requisitos:** definição dos requisitos, funcionalidades e restrições do software. Os requisitos devem ser identificados com o cliente para que o resultado a alcançar possa estar de acordo com as características pretendidas e descritas pelo cliente;
- **Implementação (ou codificação):** trata-se da construção propriamente dita do software, ou seja, a codificação de acordo com as especificações;
- **Testes:** a validação do software para garantir que todas as funcionalidades especificadas foram implementadas e de forma correta.

Muitas organizações desenvolvem software sem a utilização de nenhum processo, esta situação ocorre porque os processos tradicionais não são adequados às realidades das organizações, nomeadamente a pequenas e médias empresas que não possuem recursos suficientes para adotar o uso de forma mais completa dos processos. E por esta razão muitas das organizações não usam nenhum processo. Como consequência, desta falta de sistematização na produção de software, pode obter-se uma baixa qualidade do produto final, dificuldades da entrega do produto nos tempos e com os custos predefinidos e possibilidade de inviabilizar a futura evolução do software.

Além dos vários processos de software definidos na área da Engenharia de Software, é cada vez mais comum algumas organizações criarem o seu próprio processo ou adaptar outros processos à sua realidade. Entre os métodos de apoio aos vários processos existentes, existem os métodos tradicionais, que são mais orientados a documentação, e os métodos ágeis, que procuram idealmente desenvolver software com menos documentação. O importante é que tanto os métodos tradicionais como os ágeis pretendem garantir a qualidade do software produzido e, desta forma, a seleção do método correto para um determinado projeto é essencial.

3. Apoio ao Processo de Desenvolvimento de Software

Para a execução do PDS contribuem diversos modelos e métodos em função do tipo de projeto e abordagem que se pretende com cada projeto.

Este capítulo está dividido em três grandes secções. A primeira secção refere-se aos modelos genéricos de apoio ao PDS, também designados de métodos tradicionais ou disciplinares, e esta secção está subdividida tendo em conta a agregação dos principais modelos de acordo com o tipo de fluxo de atividades do processo que cada modelo compreende desde, por exemplo, os modelos prescritivos, iterativos e evolutivos. A segunda secção descreve a origem dos métodos ágeis através do manifesto ágil e apresenta um estudo dos principais métodos ágeis. Para cada um dos métodos apresentados são referidas as características, papéis e responsabilidades, comportamento do processo e as boas práticas recomendadas. E, por fim, a terceira secção descreve um sistema de orientações do PDS, mais que um método é um exemplo de uma plataforma proprietária que suporta a utilização de diferentes tipos de métodos e além das recomendações dos métodos complementa com ferramentas apropriadas para cada situação, neste caso o *Microsoft Solutions Framework*.

3.1 Modelos Genéricos do PDS

Existem uma panóplia de modelos de processos para o desenvolvimento de software, estes diferenciam-se através dos fluxos de processo, os quais descrevem como as atividades são organizadas em cada modelo. O fluxo de processo pode ser: **Linear ou prescritivo** (executa as atividades em sequência), **Iterativo** (repete uma ou mais atividades antes de ir para a seguinte) ou **Evolutivo ou evolucionário** (executa as atividades de forma circular).

3.1.1 Modelos Prescritivos

Os modelos prescritivos do processo de desenvolvimento de software definem um conjunto distinto de atividades, ações, tarefas, marcos (também designados de *milestones*) e artefactos de trabalho que são necessários para fazer um produto de software com alta qualidade. Esses modelos deste processo não são perfeitos, mas efetivamente fornecem um guia útil para o trabalho na engenharia de software, é portanto uma espécie de recomendação que pode ser adaptada e melhorada.

A abordagem tradicional (ou clássica) é também conhecida na bibliografia pela designação de pesada ou orientada a documentação. Os modelos tradicionais surgiram num contexto de desenvolvimento de software muito diferente do atual, baseado apenas num *mainframe* e em terminais “burros”, ou seja, operando apenas o *input* (teclado) e o *output* (monitor) e todo o processamento era realizado pelo mainframe. Para a época o custo de fazer alterações e correções era extramente alto, uma vez que o acesso a computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento de software, como compiladores ou analisadores de código. Por isso, o software tinha que ser todo planeado e documentado antes da sua implementação. O principal modelo tradicional, e ainda muito utilizado nos dias de hoje, é o modelo clássico ou sequencial com uma abordagem “*top-down*”, também conhecido como modelo em cascata.

3.1.1.1 Modelo em cascata

O modelo em cascata (*waterfall model*) é um modelo de desenvolvimento de software sequencial no qual o desenvolvimento é visto como um fluir constante para a frente (como uma cascata) através das fases de análise de requisitos, desenho, implementação, validação, integração e manutenção de software. A origem do termo cascata é frequentemente citado como sendo um artigo publicado em 1970 por W. W. Royce (António Miguel, 2006).

Royce defendia uma abordagem iterativa para o desenvolvimento de software e nem mesmo usou o termo cascata. Royce originalmente descreveu o que é hoje conhecido como o modelo em cascata como um exemplo de um método que o Royce argumentava ser um risco e um convite para falhas.

Apesar de muitas divergências na bibliografia publicada a este respeito, existe uma estrutura comum para a representação do modelo em cascata (ilustrado na figura abaixo), em que este modelo compreende as seguintes fases:

- **Definição de Requisitos** ou especificação de requisitos abrange a identificação das características e comportamentos a que o software deverá obedecer;
- **Conceção**: abarca toda o desenho da solução desde a arquitetura do sistema aos interfaces assegurando que o sistema de software ficará de acordo com os requisitos definidos;
- **Implementação**: inclui a construção do software propriamente dita, ou seja a codificação de acordo com concepção e requisitos definidos;
- **Validação**: inclui o desenho e execução de vários tipos de testes para validação do software criado;
- **Operação e manutenção**: esta fase posterior à entrega do software, já com a instalação do sistema em exploração, deverá garantir que o sistema opera como previsto e entregue. Esta fase final funciona como processo de melhoria e otimização do software e correção de defeitos.

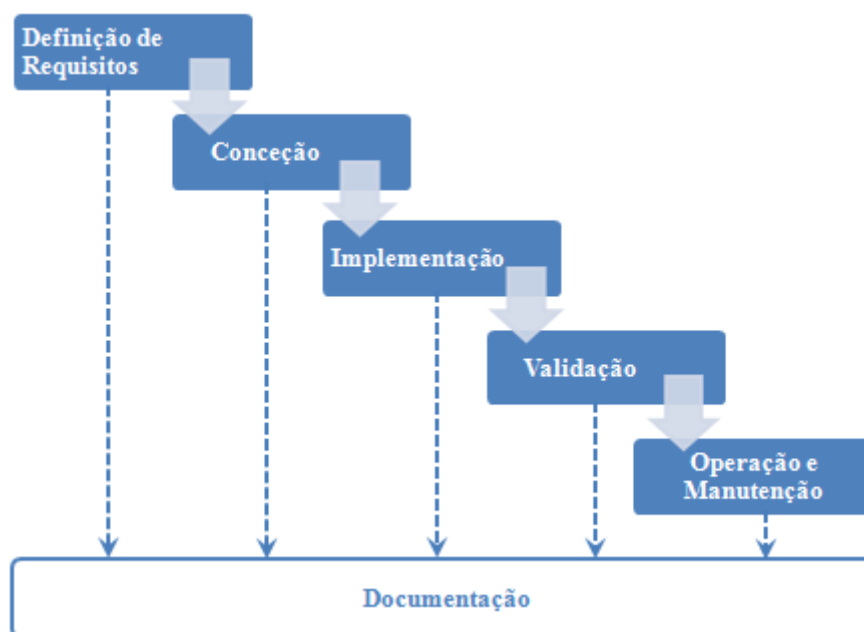


Figura 3 – modelo em cascata (adaptado de (Donato, 2010))

Este modelo é também caracterizado pela elaboração de documentação em cada uma das fases. Assim, cada fase tem associado ao seu término um, ou mesmo vários artefactos padronizados, que deverão ser aprovados para que então se inicie a etapa imediatamente a seguir, e somente se move para seguinte etapa quando a anterior está completa. Neste modelo as etapas são discretas, não havendo a possibilidade de saltar fases nem para a frente, nem para trás, ou mesmo sobreposição entre as mesmas.

Como *milestone* em cada fase concluída é concebido um documento, protótipo do software ou mesmo uma nova versão do sistema.

O modelo em cascata foi o primeiro processo publicado para o desenvolvimento de software e ainda hoje é defendido pelo PMBOK.

Este modelo dominou a forma de desenvolvimento de software até o início da década de 90, apesar de todas as advertências dos investigadores da área e programadores, que relacionaram os problemas e defeitos gerados com o adotar desta visão sequencial de tarefas.

Contudo, atualmente há vários modelos em cascata mas modificados, que podem ser incluídos como variações maiores ou menores deste processo mas já distintas do modelo tradicional que lhes deu origem.

O modelo em cascata apresenta limitações pois pressupõe que o cliente participa ativamente no projeto, que sabe muito bem o que quer e ainda explicita univocamente os requisitos. Neste modelo apenas se transita para a fase seguinte quando o cliente valida e aceita os produtos finais da fase atual. Com isto é minimizado o impacto da compreensão adquirida no decurso de um projeto, uma vez que de um processo não se pode voltar atrás de modo a alterar os modelos e as conclusões das tarefas anteriores, portanto neste sentido as novas ideias sobre o sistema não são aproveitadas.

O comportamento em sequência realça a importância dos processos definição de requisitos, concepção e respetivos documentos, como forma de minimizar o aparecimento de falhas, as quais, neste modelo, só serão detetadas na fase final do desenvolvimento.

A especificação de requisitos e desenho da solução, mesmo após o acordo entre os intervenientes estão sempre sujeitos a um amplo leque de interpretações, então o entendimento comum poderá resolvido com a criação de um protótipo, com o intuito de clarificar o entendimento de todos os interessados sobre as funcionalidades do sistema final.

Por outro, neste modelo, o processo de validação é a fase de maior risco para o projeto, dado que ocorre num estágio de desenvolvimento do projeto em que as alternativas, caso existam, envolverão muitos custos, tempo e trabalho que não estava previsto. Para colmatar esta situação a criação de completa documentação e a validação do protótipo podem reduzir este risco. No entanto, se existir atraso numa das fases, então a duração de todo o processo é afetado.

A utilização deste modelo profetiza a paciência do cliente que só terá acesso à versão funcional do projeto no final do processo de desenvolvimento. Qualquer erro ou mal-entendido, se não for detetado até que o software seja revisto, poderá ser desastroso.

Assim, o principal problema do modelo em Cascata é que é pouco flexível na divisão do projeto em fases distintas, o que dificulta possíveis alterações que são comuns na implementação de um projeto. Este é um modelo que apenas deve ser usado quando os requisitos são muito bem compreendidos ou o software a desenvolver é simples.

3.1.1.2 Modelo em cascata revisto

De modo a colmatar as limitações do modelo em cascata foi definido um novo modelo baseado no clássico modelo em cascata, designado por “modelo em cascata revisto”, cuja principal diferença consiste em prever a possibilidade de, a partir de qualquer tarefa do ciclo, poder regressar a uma tarefa anterior de forma a contemplar alterações funcionais e/ou técnicas que tenham surgido, em virtude de um maior conhecimento adquirido.

Mas também o modelo em cascata revisto apresenta limitações como o risco que, na ausência de um processo de gestão do projeto e de controlo das alterações bem definidos, pode decorrer o tempo num ciclo sem fim, sem nunca se atingir o objetivo final que é a disponibilização de um sistema a funcionar.

A figura seguinte ilustra essa possibilidade de retrocesso entre as etapas do modelo em cascata, permitindo o aproveitamento de *feedback* ao longo do projeto e não apenas a ser considerado no final do projeto pois, nesse caso, o *feedback* apenas seria considerado em projetos posteriores, esta abordagem é defendida por Jair C Leite.

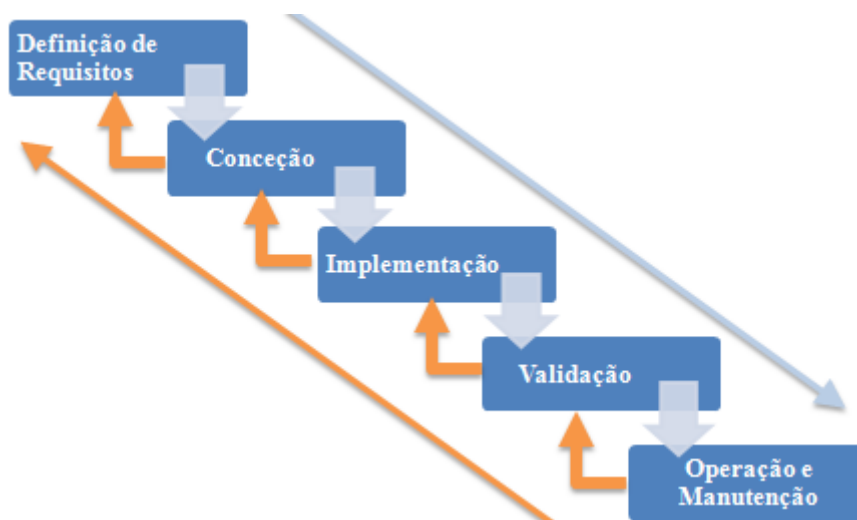


Figura 4 – modelo em cascata revisado (adaptado de (Leite, 2000))

3.1.1.3 Modelo Incremental

Tendo em vista as limitações do modelo em cascata, o modelo incremental surge como uma das alternativas, ao permitir a entrega incremental ao longo do projeto.

Neste modelo o desenvolvimento é dividido em etapas, designadas por “incrementos” que produzem incrementalmente o sistema até à sua versão final (António Miguel, 2006). Assim, cada incremento construído corresponde a uma ou várias das funcionalidades requeridas pelo sistema. Cada etapa produz um sistema totalmente funcional, apesar de ainda não cobrir todos os requisitos.

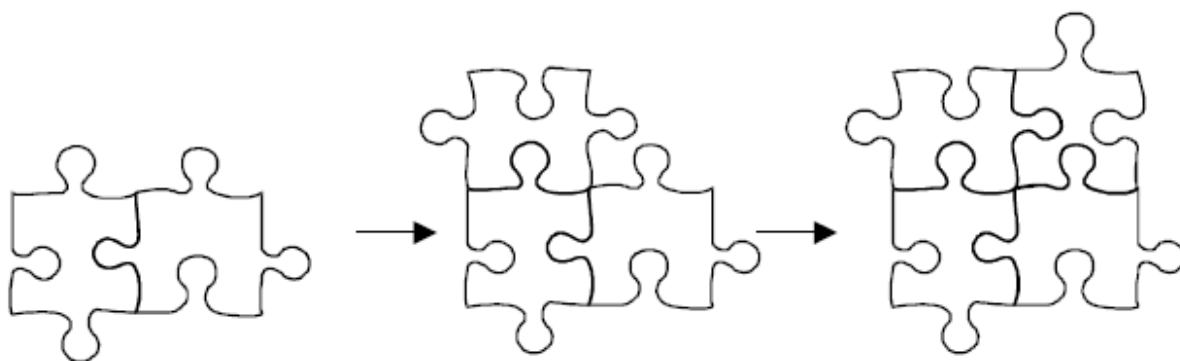


Figura 5 Modelo incremental (extraído de (Martins, 2002))

No entanto, as várias partes do sistema são desenvolvidas em paralelo, e integradas quando completas (ver Figura 5). O que não implica, requer ou pressupõe o desenvolvimento iterativo ou em cascata – ambos são estratégias de retrabalho. A alternativa ao desenvolvimento incremental é o desenvolvimento de todo o sistema com uma integração única.

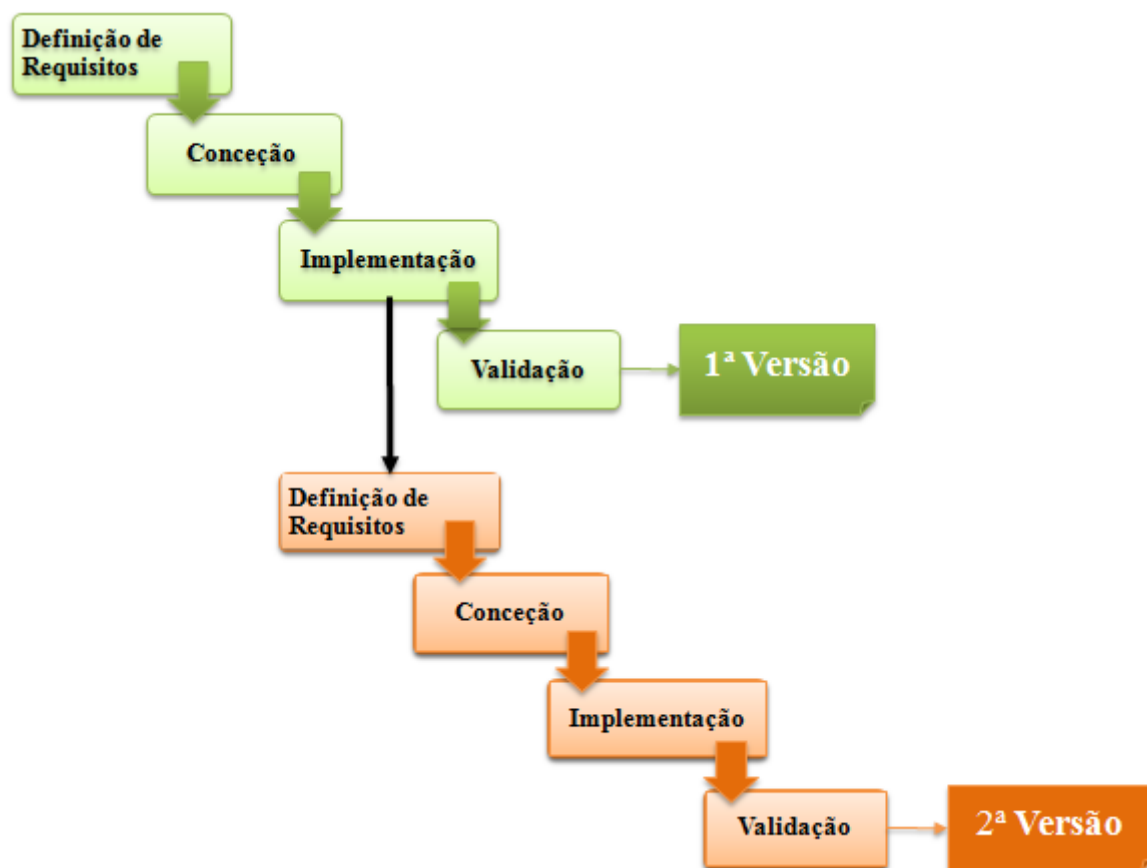


Figura 6 Desenvolvimento de incrementos (adaptado de (Martins, 2002))

Em cada incremento é realizado todo o ciclo do desenvolvimento de software, desde a especificação de requisitos aos testes de validação do sistema já em funcionamento (conforme Figura 6). Cada etapa produz um sistema totalmente funcional, apesar de ainda não cobrir todos os requisitos.

O Modelo Incremental apresenta diversas vantagens para o desenvolvimento de um software, especialmente se os requisitos não estão ainda claros inicialmente. Assim, quando o Modelo Incremental é utilizado, o primeiro incremento é normalmente constituído do núcleo do sistema, ou seja, são implementados os requisitos básicos e os detalhes suprimidos. Esse produto será entregue para avaliação, que poderá detetar, inicialmente, problemas que poderiam ser de dimensões muito maiores se apenas detetados na entrega do produto final.

No entanto, este modelo apresenta outras vantagens:

- As funcionalidades do sistema estarão disponíveis mais cedo, pois estas são entregues a partir dos incrementos;

- Incrementos iniciais agem como um protótipo para ajudar a identificar requisitos para os incrementos finais;
- Diminuem-se os riscos de falhas no projeto como um todo;
- Os serviços de prioridade mais alta do sistema tendem a receber mais testes;
- A construção de um sistema menor é sempre menos arrisca que a construção de um sistema maior.

3.1.1.4 Modelo RAD

O RAD (*Rapid Application Development*, ou seja, Desenvolvimento Rápido de Aplicação) é um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto, por exemplo: de 60 a 90 dias (Jenny, 2012). De acordo com o mesmo autor o Modelo RAD é uma adaptação de “alta velocidade” do modelo em cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes (conforme ilustrado na Figura 7).

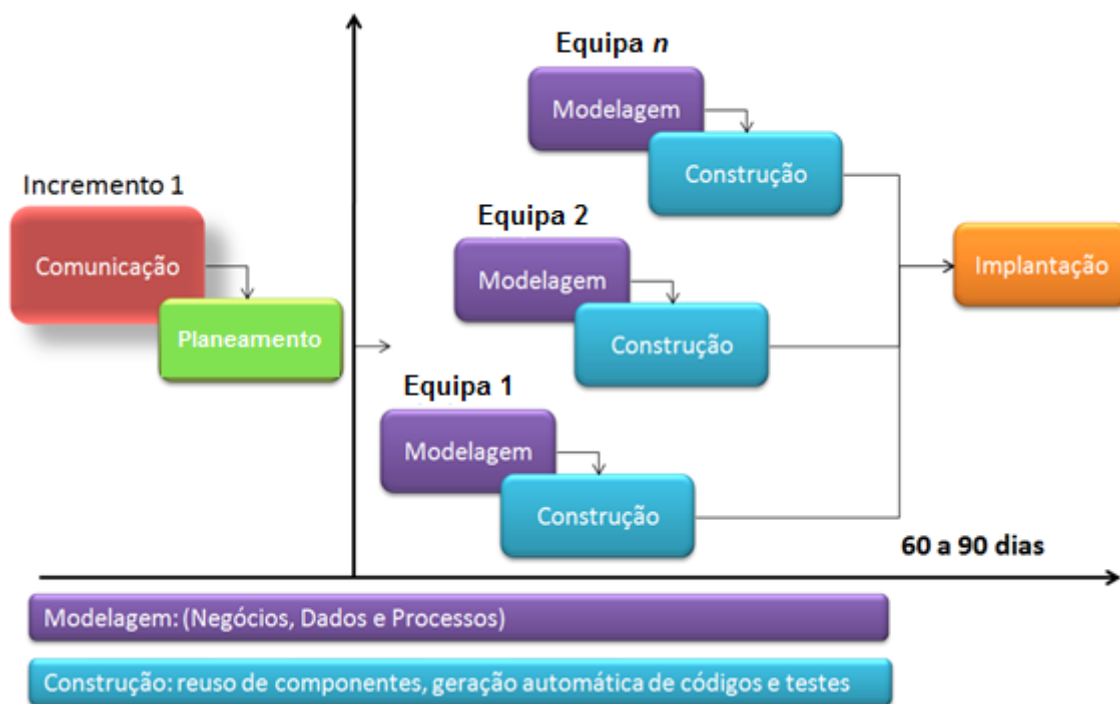


Figura 7 Modelo RAD (adaptado de (Jenny, 2012))

Neste modelo, em particular, o planeamento é essencial, porque várias equipas de software trabalham em paralelo em diferentes funções do sistema.

A conceção ou modelagem abrange três das fases principais – modelagem de negócios, modelagem dos dados e modelagem dos processos.

A construção enfatiza o uso de componentes de software pré-existent e a ferramentas de geração de códigos automático e testes.

A implantação estabelece a base das iterações subsequentes se necessárias.

Qualquer aplicação comercial, que possa ser modularizada de modo a permitir que cada função principal possa ser completada em menos de três meses, será uma candidata ao RAD.

De acordo com Pressman (Roger Pressman, 2011) a abordagem RAD apresenta as seguintes desvantagens:

- Para projetos grandes, mas passíveis de sofrer aumento, o RAD exige recursos humanos suficientes para criar um número adequado de equipas RAD;
- Se os programadores e os clientes não estiverem comprometidos com as atividades continuamente rápidas, para completar o sistema em curtíssimo espaço de tempo, então o projeto RAD falhará;
- Em projetos de alto desempenho, com interfaces de módulos muito interligadas, há maior probabilidade de erros;
- Se o sistema não puder ser adequadamente modularizado, as construções dos componentes necessários ao RAD serão problemáticas;
- O RAD pode não ser ajustado quando os riscos técnicos são altos, por exemplo, na utilização de uma nova tecnologia.

3.1.2 Modelo Iterativo

No modelo iterativo pretende-se melhorar, ou refinar, a pouco a pouco o sistema, através de iterações consecutivas. Em cada iteração a equipa de desenvolvimento identifica e especifica os requisitos relevantes, cria um projeto utilizando a arquitetura escolhida como guia, implementa o projeto em componentes e verifica se esses componentes satisfazem os requisitos. De acordo com António Miguel (António Miguel, 2006), se uma iteração atinge os seus objetivos, o desenvolvimento prosseguirá com a próxima iteração, caso contrário a equipa deverá rever as suas decisões e tentar uma nova abordagem.

Portanto, o âmbito do sistema não é alterado (conforme ilustrado na Figura 8), mas o seu detalhe vai aumentando em iterações sucessivas.



Figura 8 Modelo Iterativo (extraído de (Martins, 2002))

O modelo iterativo baseia-se fortemente na participação e numa boa comunicação entre programadores e utilizadores, pois pressupõe um grande envolvimento do utilizador e do cliente. O que, por vezes, pode levar a mal-entendidos devido a divergências entre os programadores e utilizadores que rapidamente podem ser superados, porque há resultados mais rápidos e tangíveis, e assim os utilizadores também serão capazes de dar um melhor feedback. Através dos vários resultados alcançados ao longo do projeto permitirão que os utilizadores tenham confiança em um bom resultado final.

A cada ciclo do sistema os utilizadores e cliente podem utilizar o sistema diretamente, assim eles participarão como testadores no processo de desenvolvimento e eles estarão interagindo com o sistema durante o desenvolvimento.

Quanto aos riscos, estes podem ser mais facilmente geridos, pois serão analisados e controlados por cada parte do sistema a serem desenvolvidos e em pequenos espaços de tempo. Assim, os sistemas complexos são mais claros e menos complexos para implementá-los aos poucos, sem a necessidade de começar a implementação do zero.

Ao fim de cada iteração pode-se ter um feedback para ter noção de como está o projeto, mesmo que não esteja de acordo com o planeado ainda haverá tempo para mudanças, já com o modelo Cascata isso não seria possível. Desta forma, este modelo contempla a possibilidade de incorporar rapidamente no processo de desenvolvimento alterações nos requisitos.

De acordo com as afirmações anteriores pode-se depreender que o modelo iterativo apresenta as seguintes desvantagens:

- Durante o processo de desenvolvimento necessita adaptar-se e refinar o sistema, isso poderá levar a que o produto final saia totalmente diferente da sua ideia original;
- Pode acontecer a continuação do sistema e a identificação de muitos requisitos novos, assim esse sistema nunca irá terminar, o que leva ao aumento do âmbito;
- Gestores de projetos acostumados com a forma linear do desenvolvimento de software podem ter alguns problemas na adoção desta forma mais flexível.

3.1.2.1 *Modelo Iterativo e Modelo Incremental*

Os dois modelos, o iterativo e o incremental, podem ser facilmente confundidos, mas são profundamente diferentes.

Com base nas descrições apresentadas anteriormente dos dois modelos, iterativo e incremental, pode-se depreender que no modelo incremental, desenvolve-se uma primeira versão, sem se preocupar muito com o todo. Portanto, o desenvolvimento inicia-se sem que haja um conhecimento amplo dos requisitos do sistema. Através do feedback do utilizador, os requisitos vão sendo identificados e implementados nas versões subsequentes do sistema. Já no modelo iterativo, tem-se uma visão inicial de todos (ou quase todos) os requisitos do utilizador, mas não se implementam todos de uma vez: a cada iteração é implementado um subconjunto de requisitos. Estes requisitos são então apresentados ao utilizador, que fica satisfeito por ver evoluções do projeto.

3.1.3 Modelos evolutivos

Os modelos evolutivos enfatizam o desenvolvimento de software evoluído a partir de um protótipo inicial, em que todos os conceitos e ideias vão sendo materializados em requisitos à medida que o protótipo evolui, até atingir o produto idealizado. O objetivo é trabalhar juntamente com o cliente ou utilizador em todas as fases do desenvolvimento, a partir das especificações iniciais, no entanto os requisitos devem ser bem compreendidos.

De acordo com Pressman (Roger Pressman, 2011) os modelos evolutivos são explicitamente projetados para acomodar um produto que evolui com o tempo, em que a cada nova iteração é produzida uma versão cada vez mais completa do software.

Os prazos reduzidos de mercado tornam impossível completar-se um produto de software abrangente, mas uma versão reduzida pode ser elaborada para fazer frente à competitividade ou às pressões do negócio e, por outro lado, com o passar do tempo os requisitos do negócio e do produto mudam frequentemente, à medida que o desenvolvimento prossegue, o que dificulta o caminho direto para o produto final.

O modelo evolucionário pode trazer diversas vantagens, tais como antecipar o produto final para avaliação do cliente, manter uma comunicação entre os programadores e utilizadores para solucionar problemas técnicos e antecipar a formação dos utilizadores. No entanto, de acordo com o mesmo autor, Pressman, estes modelos acarretam os seguintes problemas:

- Dificuldade acrescida no planeamento de projeto, pois há um número incerto de ciclos para construir o produto;

- A velocidade da evolução pode trazer problemas no próprio produto;
- São processos focados na flexibilidade e extensibilidade, em detrimento de outros critérios de qualidade.

3.1.3.1 Modelo Espiral

Durante muitos anos, o modelo cascata foi a base da maior parte do desenvolvimento de projetos de software, mas em 1988 Barry Boehm sugeriu o modelo em espiral. Do modelo em espiral para desenvolvimento de software saltam a vista dois aspetos: a **análise de risco** e **prototipagem**. O modelo espiral incorpora-os de uma forma interativa permitindo que as ideias e o progresso sejam verificados e avaliados constantemente. Cada iteração à volta da espiral pode ser baseada num modelo diferente e pode ter diferentes atividades. No caso da espiral, não foi a necessidade do envolvimento dos utilizadores que inspirou a introdução de iteração mas sim a necessidade de identificar e controlar riscos. No modelo espiral para engenharia de requisitos mostra-se que as diferentes atividades são repetidas até uma decisão ser tomada e o documento de especificação de requisitos ser aceite. Se forem encontrados problemas numa versão inicial do documento, reentra-se nas fases de levantamento, análise, documentação e validação. Isto repete-se até que seja produzido um documento aceitável ou até que fatores externos, tais como prazos e falta de recursos ditem o final do processo de engenharia de requisitos (António Miguel, 2006).

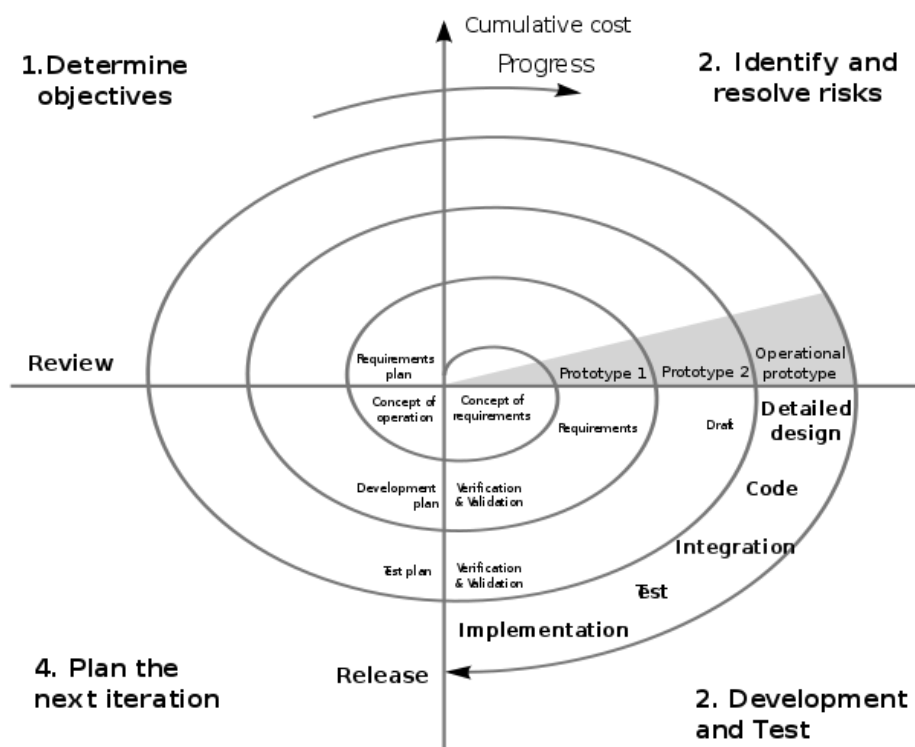


Figura 9 Modelo espiral (extraído de (Tomás, 2010))

O processo de desenvolvimento é representado como uma espiral, em vez de uma sequência de atividades (conforme Figura 9), tendo em conta esta dimensão radial a cada nova iteração, que se inicia do centro e afasta-se do mesmo, então versões progressivamente mais completas do software são construídas. Durante a primeira iteração ao redor da espiral, objetivos, alternativas e restrições são definidos e os riscos são identificados e analisados. Se a análise dos riscos indicar alguma incerteza nos requisitos, a prototipagem pode ajudar tanto o programador quanto o cliente a refinar esses requisitos.

Caso o cliente tenha alguma sugestão de alteração deverá apresentá-la à equipa de desenvolvimento. A cada iteração na espiral a conclusão da análise dos riscos resulta numa decisão de “Prosseguir/ Não Prosseguir” e se os riscos forem considerados elevados, o projeto pode ser cancelado.

O modelo espiral define quatro quadrantes, nos quais as atividades (gestão ou técnicas) de um projeto são executadas durante um ciclo na espiral. A manutenção de um software utilizando este modelo é tratado da mesma forma que o desenvolvimento.

Variações do modelo espiral, de acordo com António Miguel (António Miguel, 2006), consideram entre três a seis principais atividades da espiral, nomeadamente:

1. **Planeamento** para determinação dos objetivos, alternativas e restrições;
2. **Análise de riscos** para a análise de alternativas e identificação/ resolução dos riscos;
3. Realização de **protótipos**;
4. **Desenvolvimento** do produto de software para o cliente;
5. **Testes e disponibilização** do software;
6. Submissão do software à **avaliação pelo cliente**.

No entanto, o modelo espiral apresenta as seguintes desvantagens:

- Pelas suas características de avaliação e planeamento baseadas em riscos, exige que se tenha gestores e técnicos experientes;
- É importante ter em conta que podem existir diferenças entre o protótipo e o sistema final. O protótipo pode não cumprir os requisitos de desempenho, pode ser incompleto, e pode refletir somente alguns aspetos do sistema a ser desenvolvido;
- Pode levar ao desenvolvimento em paralelo de múltiplas partes do projeto, cada uma sendo abordada de modo diferenciado, por isso é necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

3.1.3.2 Prototipagem

A definição, pelo cliente ou utilizador, de todos os requisitos necessários para o sistema é geralmente uma tarefa muito difícil. É quase impossível prever como o sistema irá afetar o funcionamento das práticas de trabalho, como será a interação com outros sistemas e que operações dos utilizadores devem ser automatizadas. Mas para poder validar os requisitos de uma forma mais eficiente, é aconselhada a utilização de um protótipo do sistema, pois de acordo com António Miguel (António Miguel, 2006) a prototipagem constitui uma das medidas de redução do risco que se podem utilizar.

Atualmente, os limites entre a prototipagem e o desenvolvimento normal do sistema, muitas vezes, são indefinidos e muitos sistemas são desenvolvidos usando uma abordagem evolutiva. A principal função é ajudar os clientes e programadores a entender os requisitos para o sistema.

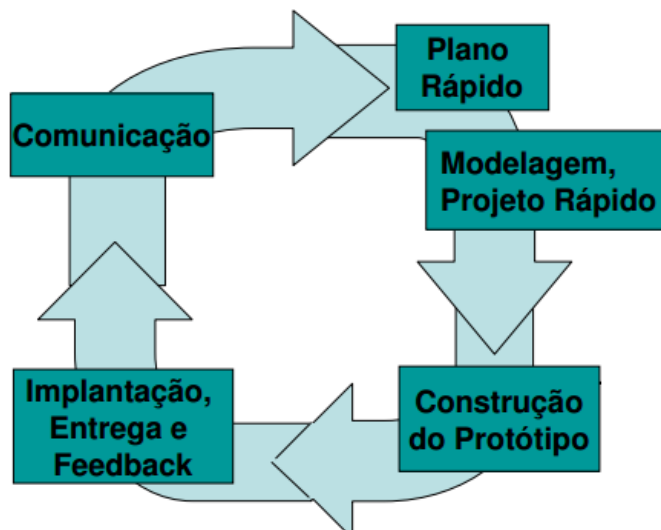


Figura 10 Prototipagem (extraído de (Zadrozny, 2006))

É importante realçar que neste modelo os utilizadores podem experimentar (conforme o fluxo na Figura 10) o protótipo para ver como o sistema pode apoiar o seu trabalho (levantamento de requisitos). Uma das principais utilidades do protótipo é o fato do mesmo poder revelar erros e omissões nos requisitos (validação de requisitos). A prototipagem pode ser considerada como uma atividade de redução de riscos que reduz os riscos nos requisitos.

Existem dois tipos de prototipagem no processo de desenvolvimento de software: Prototipagem Evolucionária e a Prototipagem Descartável. Entende-se como Prototipagem Evolucionária à abordagem para o desenvolvimento do sistema onde um protótipo inicial é produzido e refinado através de vários estágios até atingir o sistema final. O objetivo desta

prototipagem é fornecer aos utilizadores finais um sistema a funcionar. Já a Prototipagem Descartável visa à criação de um protótipo o qual é utilizável uma implementação prática do sistema é produzida para ajudar a levantar os problemas com os requisitos e depois descartado. O sistema é então desenvolvido utilizando um outro processo de desenvolvimento (Macoratti, 2005).

Quanto ao protótipo propriamente dito este pode ser apresentado ao cliente ou utilizador em diferentes formas:

- Protótipo em papel;
- Modelo executável retratando a interface homem-máquina capacitando o cliente a compreender a forma de interação com o software;
- Protótipo de trabalho que implemente um subconjunto dos requisitos indicados;
- Programa existente (pacote) que permita representar todas ou parte das funções desejadas para o software a construir.

Este modelo apresenta as seguintes vantagens:

- Modelo de desenvolvimento principalmente adequado para alguns sistemas de grande porte que representem um certo grau de dificuldade para exprimir rigorosamente os requisitos;
- Possibilidade de demonstrar o funcionamento através da construção de um protótipo do sistema;
- Possibilidade em obter uma versão, mesmo simplificada do que será o sistema, com um pequeno investimento inicial;
- A experiência adquirida com o desenvolvimento do protótipo vai ser de extrema utilidade nas etapas posteriores do desenvolvimento do sistema real, ao permitir reduzir o seu custo e resulta como num sistema melhor concebido.

No entanto, o modelo de prototipagem apresenta as seguintes desvantagens:

- Quando o cliente é informado que o produto precisa ser reconstruído, então o cliente exige que alguns acertos sejam aplicados para tornar o protótipo um produto e muito frequentemente, a gestão de desenvolvimento de software cede, havendo assim lugar a alterações de âmbito;
- O programador muitas vezes faz cedências durante a implementação a fim de colocar um protótipo em funcionamento rapidamente. Depois de algum tempo, o programador

pode familiarizar-se com essas opções e esquecer-se de todas as razões pelas quais elas são inadequadas - a opção menos ideal tornou-se então parte integrante do sistema.

Pode-se afirmar que a prototipagem, em resumo, é o desenvolvimento rápido de um sistema. O principal objetivo deste modelo é o desenvolvimento rápido de software para validar os requisitos. Este modelo é também usado frequentemente como técnica que pode ser utilizada dentro do contexto de qualquer outro modelo.

3.1.4 Modelo RUP

O RUP (Rational Unified Process, ou seja, Processo Unificado Racional) é um processo proprietário de Engenharia de software criado pela *Rational Software Corporation*, adquirida pela *IBM*, ganhou o novo nome IRUP que agora é uma abreviação de *IBM Rational Unified Process*, fornece técnicas a serem seguidas pelos membros da equipa de desenvolvimento de software com o objetivo de aumentar a sua produtividade no processo de desenvolvimento.

O RUP usa a abordagem da orientação a objetos na sua conceção e é projetado e documentado através da utilização da notação UML (Unified Modeling Language) para ilustrar os processos em ação. Utiliza técnicas e práticas aprovadas comercialmente.

É um processo considerado pesado e preferencialmente aplicável a grandes equipas de desenvolvimento e a grandes projetos, porém o fato de ser amplamente personalizado torna possível que seja adaptado para projetos de qualquer escala. Para a gestão do projeto, o RUP fornece uma solução disciplinada de como assinalar tarefas e responsabilidades dentro de uma organização de desenvolvimento de software.

O RUP é, por si só, um produto de software. É modular e automatizado, e toda a sua metodologia é apoiada por diversas ferramentas de desenvolvimento integradas e vendidas pela IBM através de seus "*Rational Suites*" (IBM, 2012).

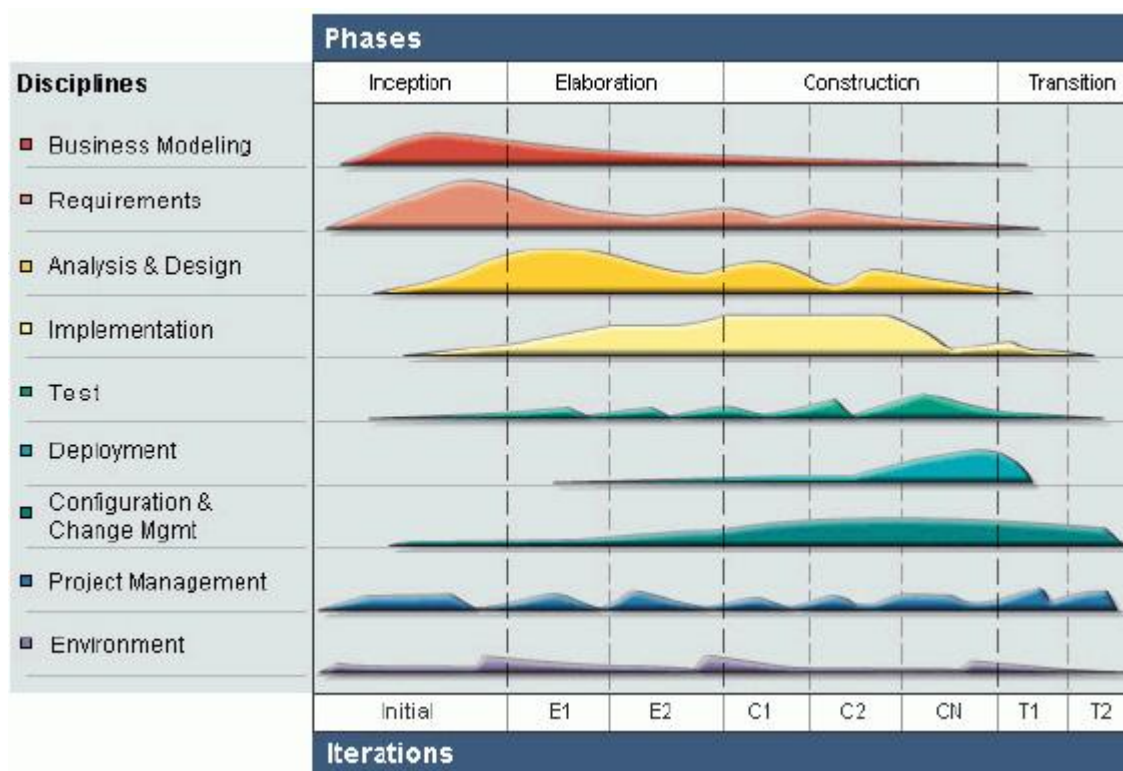


Figura 11 Arquitetura do RUP (extraído de (Portier, 2007))

Representado graficamente (conforme Figura 11), o RUP, apresenta duas dimensões que caracterizam as **fases** e as **disciplinas**:

- O eixo horizontal (fases) representa o tempo de vida de um projeto, o aspecto dinâmico do ciclo de vida do processo expressado em termos de fases, iterações e marcos. Uma iteração é descrita como uma passagem por todas as disciplinas. A partir da fase de elaboração, cada iteração deve produzir um módulo do sistema e o projeto é concluído quando todos os módulos forem produzidos e interligados, estes módulos são os componentes do sistema.

A cada iteração deve ser obtida uma versão executável, a qual é testada e aprovada pelo utilizador. Dessa forma, o risco é minimizado, pois eventuais falhas são identificadas já nas fases iniciais do projeto.

- O eixo vertical (disciplinas) representa os grupos de atividades lógicas que são realizadas durante o decorrer do tempo. Esta dimensão demonstra o aspecto estático do processo, que será composto por disciplinas, atividades, fluxos de trabalho, artefactos e papéis. A Figura 11 demonstra o esforço que é aplicado nas disciplinas ao longo do tempo. A disciplina de requisitos, por exemplo, requer mais trabalho nas fases iniciais

(iniciação e elaboração), enquanto a de implantação praticamente não existe antes das fases finais (construção e transição).



Figura 12 Estrutura básica do RUP

Assim o ciclo de vida de software do RUP é dividido em quatro fases sequenciais (conforme Figura 12):

- **Conceção ou Iniciação** – pretende-se compreender as necessidades e visão do projeto, para tal são executadas as atividades básicas de modo a definir o âmbito do projeto e preparar o ambiente do mesmo, nomeadamente avaliar a estrutura do projeto e selecionar ferramentas a usar;
- **Elaboração** – nesta fase é definida a linha base (*baseline*) para a arquitetura do sistema, a fim de determinar uma base estável para o esforço da fase seguinte. Nesta fase deve decidir-se basicamente sobre a organização do sistema, a seleção dos elementos, identificação dos riscos, previsão dos custos e cronograma do projeto;
- **Construção** - pretende-se completar o esclarecimento de requisitos e realizar o desenvolvimento propriamente dito do sistema com base na arquitetura definida;
- **Transição** – consiste na execução das várias atividades necessárias para a entrega do sistema, principalmente a validação da qualidade do produto, acompanhamento na passagem para produção do sistema já no ambiente do cliente e a formação dos utilizadores.

No final de cada fase é executada uma avaliação para determinar se os objetivos dessa fase foram alcançados. Uma avaliação satisfatória permite que o projeto passe para a próxima fase.

Na fase de Elaboração são escolhidos os elementos a utilizar no projeto. O RUP possui os seguintes elementos principais: papéis, atividades, artefactos e disciplinas.

Um papel (ou perfil) define o comportamento e as responsabilidades de um determinado indivíduo ou grupo de indivíduos que trabalham como uma equipa. Papéis não são indivíduos mas responsabilidades que estes assumem. Um indivíduo pode assumir vários papéis. São exemplos de papéis:

- **Analista de Requisitos** – indivíduo que assume este papel coordena a especificação dos requisitos e a concepção dos casos de uso com a identificação das funcionalidades do sistema e estabelece os limites do sistema;
- **Arquiteto** - indivíduo que define responsabilidades, operações, atributos, relacionamentos de uma ou mais classes e determina como elas devem ser ajustadas para serem implementadas no ambiente;
- **Testador** - responsável pelo desenho, execução e validação de testes. Implementa procedimentos de testes e avalia a abrangência dos testes, resultados e a efetividade. Uma atividade é uma unidade de trabalho que um indivíduo executa quando está a exercer um determinado papel e produz um resultado importante para o contexto do projeto.

As Atividades são as ações realizadas por um ator, que assume vários papéis através da utilização de casos de uso, que geram algum resultado significativo para o projeto. As atividades são usadas para definir o comportamento de um determinado indivíduo a desempenhar um determinado papel.

As atividades como elementos estão fortemente relacionadas aos artefactos. Os artefactos fornecem a entrada e a saída para as atividades e o mecanismo pelo qual as informações são transmitidas entre as atividades.

Os Artefactos são os produtos de trabalho finais ou intermediários produzidos e usados durante os projetos. Um artefacto pode ser um dos seguintes elementos:

- Um documento, como Caso de Negócio ou Documento de Arquitetura de Software;
- Um modelo, como o Modelo de Casos de Uso ou o Modelo de Design;
- Um elemento do modelo, ou seja, um elemento existente em um modelo, como uma classe ou um subsistema.

Esses artefactos são agrupados em disciplinas, também designadas de fluxos de trabalho, cada disciplina produz um conjunto de artefactos diferentes. As disciplinas servem para organizar logicamente as atividades de engenharia de software de acordo com sua natureza. Elas descrevem o que deve ser feito em cada iteração relativamente a atividades, papéis e artefactos.

Existem nove disciplinas básicas, em que seis são de engenharia de software e as outras três são de suporte. Estas disciplinas são:

- **Concepção do Negócio** - visa entender a estrutura e a dinâmica da organização para qual se vai desenvolver o projeto, ao permitir uma única interpretação para cliente, utilizadores e programadores;

- **Requisitos** - definir, documentar e organizar as funcionalidades do sistema, bem como gerir o âmbito e as alterações das funcionalidades;
- **Análise e Conceção** - conversão dos requisitos para numa linguagem que descreva como o sistema deve ser implementado, a linguagem utilizada é o UML;
- **Implementação** - fluxo em que o sistema é codificado na linguagem de programação especificada. Também são feitos testes e a integração com módulos desenvolvidos em outros ciclos;
- **Testes** - testa todo o sistema, verificando se este está de acordo com o que foi especificado. Valida também a integração entre objetos já implementados. Tem como objetivo encontrar defeitos antes da entrega do software;
- **Entrega** - instalação do sistema no ambiente do cliente e formação aos utilizadores finais;
- **Gestão do Projeto** - inclui a gestão de riscos, planeamento, monitoria e controlo do projeto;
- **Gestão de Configuração e Alterações** – gestão de documentos gerados no decorrer do projeto, para que estes sempre estejam em conformidade ao que já foi desenvolvido;
- **Ambiente** - define como o RUP foi configurado para ser utilizado no projeto, além da organização do ambiente de trabalho para toda a equipa de desenvolvimento.

Não existe uma forma exata de aplicar o RUP, pois este modelo pode ser aplicado de várias formas e de maneira diferente em cada projeto e organização. A bibliografia não é unânime na classificação deste método, tanto que diversos atores consideram o RUP como um método ágil. Porém, a revisão bibliográfica revelou que o mais correto será considerar o RUP como base para adaptações ágeis e por si só apenas um modelo da abordagem não ágil mas tradicional.

3.1.5 Critérios para Seleção do Modelo

A escolha de um modelo de ciclo de vida, ou modelo de processo, deve ter em consideração as características do contexto do projeto ao qual será aplicado, pois diferentes projetos de desenvolvimento possuem diferentes necessidades que devem orientar à seleção de um processo mais adequado.

Tendo em conta as anteriores descrições dos modelos prescritivos, iterativos, evolutivos e do RUP e com base em algumas considerações apresentadas na disciplina de Análise de Sistemas (ATDM, 2009), seguidamente é apresentado um quadro resumo dos principais critérios a considerar para a seleção de um modelo (ver Quadro 1).

Pretende-se analisar resumidamente o comportamento dos modelos relativamente aos seguintes critérios: clareza dos requisitos (quanto à compreensibilidade na especificação dos requisitos); domínio da tecnologia (familiarização com a tecnologia que se vai utilizar); complexidade do sistema (simplicidade ou complicação do sistema têm diferentes exigências na implementação do sistema); segurança do sistema (garantia de seguridade do sistema por vezes apenas implementada quando o cliente requiere essa funcionalidade); curto prazo (duração curta para implementação do projeto); gestão de alterações (recetividade para a inclusão de mudanças no decorrer do projeto) e visibilidade do plano (para projetar riscos e atempadamente executar as correções necessárias).

Critério	Análise e impacto nos modelos
Clareza dos requisitos	Os modelos RAD e prototipagem são os mais apropriados quando os requisitos não estão claramente definidos pelos utilizadores, assim há a interação e validação mais cedo com o sistema.
Domínio da tecnologia	Se o sistema é desenhado sem alguma familiaridade com a tecnologia base que será utilizada, então os riscos aumentam pois as ferramentas podem não satisfazer as expectativas do sistema ou podem por em causa, por exemplos, aumento dos custos para adaptação das ferramentas ou formação da equipa de implementação.
Complexidade do sistema	Sistemas complexos requerem uma análise e desenho mais detalhada. Essa análise pode ser melhor contemplada nos modelos de prototipagem, cascata e RUP pelo maior foco nas fases de análise e desenho do sistema.
Segurança do sistema	A segurança do sistema é um fator importante no desenvolvimento do sistema. No modelo de prototipagem não é dada a devida atenção porque há um maior foco na análise e desenho do sistema a não ser que seja considerada a segurança como prioridade mais elevada.
Curto prazo	O modelo RAD adequa-se bem a projetos com tempos curtos pois introduzem celeridade ao Projeto, no entanto modelos como o cascata são a pior escolha quando a duração é essencial pois este modelo não se adequa a projetos de curta duração.
Gestão de alterações	Os modelos RAD e prototipagem pressupõem um melhor enquadramento das mudanças ao longo do projeto, pois é possível refletir as alterações de forma mais rápida sem por em causa a implementação do mesmo. Por outro lado não se adequam ao modelo cascata uma vez que não pressupõe a inclusão de alterações ao longo da implementação do sistema.
Visibilidade do plano	O modelo RAD antecipa algumas das decisões críticas de desenho, consequentemente, isto vai ajudar os gestores de projeto a reconhecer e endereçar os fatores de risco e manter as expectativas altas do cliente.

Quadro 1 Critérios a ter em conta na seleção de um modelo

3.2 Métodos Ágeis

Os métodos ágeis surgiram como que em reação à adoção dos modelos considerados pesados e tiveram como principal motivação a criação de alternativas para o modelo em cascata.

É de realçar que ao longo da revisão bibliográfica foi possível verificar a fraca concordância na definição de muitos dos conceitos acerca dos diversos métodos ágeis existentes, portanto de seguida são apresentados os factos encontrados com maior coerência e de alguma forma consolidados nas fontes consultadas.

3.2.1 Manifesto ágil

Divulgado na página oficial do Manifesto Ágil, em fevereiro de 2001, reuniu-se um grupo de dezassete representantes de métodos de desenvolvimento de software considerados “leves” que estavam a trazer valor para os seus clientes. Durante dois dias estes especialistas estiveram reunidos numa estação de *ski* em Snowbird, no Utah, para identificar os pontos em comum entre os seus métodos. A partir dessa reunião, foi criado o **Manifesto Ágil**, onde são definidos os valores Ágeis e o termo “**Metodologia Ágil**” ou “**Método Ágil**” tornou-se popular.

O manifesto ágil compreende quatro valores fundamentais e doze princípios (Manifesto for Agile Software Development, 2001). Os valores fundamentais, conforme Anexo 1, deste manifesto são:

- Os **indivíduos e as suas interações** acima de processos e ferramentas;
- O **funcionamento do software** acima de documentação abrangente;
- A **colaboração dos clientes** acima da negociação de contratos;
- **Responder a mudanças** mais do que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, será de valorizar mais os itens à esquerda.

Quanto aos princípios sustentados pelo Manifesto Ágil, conforme Anexo 2, são:

- A prioridade é garantir a **satisfação do cliente** através da entrega rápida e contínua de software com valor para o cliente;
- **Mudanças nos requisitos** são bem-vindas, mesmo tardiamente no desenvolvimento. Os processos ágeis tiram proveito das mudanças visando vantagem competitiva para o cliente;
- **Entregar frequentemente** software a funcionar, de poucas semanas a poucos meses, com preferência à menor escala de tempo;

- **Pessoas do negócio e programadores** devem trabalhar diariamente em conjunto durante todo o projeto;
- Manter a **equipa motivada** ao proporcionar o ambiente, apoio e confiança necessários para a realização do trabalho;
- O **método mais eficiente e eficaz** de transmitir informações para e entre uma equipa de desenvolvimento é através de conversa cara-a-cara.
- Ter o **software a funcionar** é a melhor medida de progresso;
- Os **processos ágeis** promovem desenvolvimento sustentável. Os patrocinadores, programadores e utilizadores devem ser capazes de manter um ritmo constante indefinidamente.
- A atenção contínua à **excelência técnica** e a um bom projeto aumentam a agilidade.
- A **simplicidade** - a arte de maximizar a quantidade de trabalho não realizado - é essencial.
- As melhores arquiteturas, requisitos e *designs* provêm de **equipas auto-organizadas**.
- Em intervalos regulares, a **equipa deve refletir** sobre como se tornar mais eficaz e então ajustar-se e adaptar o comportamento em conformidade.

Não se trata, como poderia parecer à primeira vista, de um desprezo aos elementos e ferramentas tradicionais do desenvolvimento de software, mas sim do estabelecimento de uma escala de valores, na qual a flexibilidade e a colaboração são mais relevantes do que a rigidez de processos e planeamento clássicos.

Portanto, o manifesto reconhece que determinados conceitos como processos, ferramentas, documentação, contratos e planos podem ser importantes no desenvolvimento de software, mas que há conceitos mais importantes a valorizar. O manifesto ágil é uma filosofia onde muitos métodos se encaixam.

Em síntese os métodos ágeis são uma nova atitude, não se trata de um desprender total das práticas dos métodos tradicionais, como a não elaboração de documentação mas a elaboração de apenas a documentação que apresente valor.

3.2.2 Métodos orientados a objetos

Nesta secção serão enumerados e explicitados alguns dos principais métodos ágeis, apresentados por ordem alfabética, no entanto apenas se pretende fornecer uma visão geral de cada um dos métodos: ao focar as suas características principais que os destacam entre as inúmeras opções disponíveis atualmente, como os seus valores e princípios; a descrever os papéis e responsabilidades recomendados para cada método e o modo de funcionamento, ou seja, o ciclo de vida específico de cada método apresentado.

Não é intenção documentar todas as abordagens, pois cada uma delas daria um próprio relatório, descreve-se de seguida as principais características dos métodos mais conhecidos. Devido à maior popularidade dos métodos XP (maior foco na programação) e SCRUM (maior foco na gestão do projeto) seguidamente estes apresentam um maior nível de detalhe que os restantes métodos.

3.2.2.1 *Adaptive Software Development (ASD)*

O *Adaptive Software Development (ASD)*, em português Desenvolvimento Adaptativo de Software, foi proposto por James Highsmith como uma técnica para construção de software complexos. O apoio filosófico do ASD centralizou-se na colaboração humana e na auto-organização. Muitos dos princípios do ASD estão baseados nas pesquisas do autor sobre métodos iterativos de desenvolvimento (Rodrigo, 2011).

O ASD tem como base principal um método RAD, o RADical Software Development, evoluindo-o ao incorporar conceitos da teoria de sistemas adaptativos complexos. O método e as ferramentas de desenvolvimento rápido de aplicativos RAD permitem desenvolver sistemas de forma muito mais rápida, especialmente aqueles sistemas onde a interface de utilizador representa uma componente importante.

Este é um método dedicado à aprendizagem contínua, dirigido a mudanças, reavaliações e grande colaboração entre programadores e clientes.

A seguinte descrição deste método resulta maioritariamente da análise e consulta da página web, intitulada “Introdução ao Adaptive Software Development”.

3.2.2.1.1 Características

O foco principal do ASD são os problemas em desenvolvimentos complexos e extensos. O método incentiva o desenvolvimento iterativo incremental, com constante prototipagem. Fundamentalmente o objetivo do ASD é proporcionar uma ferramenta que oriente os projetos de modo a prevenir falhas que se possam tornar em caos, sem suprimir a urgência e a criatividade.

O ASD não possui muitos detalhes de como as coisas deverão ser feitas, pois apenas esboça as oportunidades que a equipa de desenvolvimento terá que ter em conta, ou seja, de acordo com o autor do método ASD, este não possui um conjunto de práticas definidas e sim um conjunto de propriedades que caracterizam o processo de desenvolvimento adaptativo. São consideradas propriedades deste método as seguintes:

- **Dirigido a missões:** em cada ciclo as atividades são encaradas como missões, onde cada iteração possui a sua missão e esta pode mudar ao longo do projeto.
- **Baseado em características:** as atividades de desenvolvimento devem ser orientadas às características, ou seja, à construção de uma funcionalidade de cada vez;
- **Iterativo:** o processo de desenvolvimento deverá ser dividido em pequenos ciclos (iterações) com o objetivo final de entregar um grupo de requisitos implementados que satisfaça a missão definida para cada iteração;
- **Prazos pré-definidos:** a ambiguidade em projetos de software pode ser reduzida fixando-se prazos finais tangíveis e numa base regular, mesmo ao forçar os participantes do projeto a tomarem decisões que possam representar algum risco no início do projeto.
- **Tolerância a mudanças:** as mudanças são frequentes portanto deve-se estar apto a adaptá-las mais do que controlá-las, para tal deve-se fazer uma constante avaliação de quais os componentes que podem mudar;
- **Orientado a riscos:** as características ou funcionalidades de risco mais elevado são as primeiras a realizar.

3.2.2.1.1 Papéis

Relativamente ao método ASD não se encontram descrições detalhadas de como deve ser formada a equipa na utilização deste método, nomeadamente os papéis na equipa e as funções a desempenhar. No entanto, o método ASD sugere os seguintes papéis:

- **Patrocinador** (*executive sponsor*): nomeado como a pessoa com responsabilidade global em relação ao produto que está a ser desenvolvido;

- **Facilitador** (*facilitator*): tem a responsabilidade de liderar e planejar as sessões de desenvolvimento;
- **Escrivão** (*scribe*): responsável por fazer as anotações necessárias;
- **Programador** (*developer*): responsável pela construção propriamente dita do software e participante nas sessões de desenvolvimento;
- **Gestor de projeto** (*Project Manager*): papel opcional com a responsabilidade de gestão do projeto, o papel pode ser repartido com o facilitador;
- **Cliente** (*Customer*): o próprio cliente, ou seu representante, deve conhecer os resultados esperados do produto em desenvolvimento.

O método ASD não possui limitações na sua aplicação, ou seja, tanto pode ser utilizado em sistemas simples como em complexos e também não estabelece um número máximo de pessoas para a sua equipa. Uma característica interessante do ASD é que ele não obriga a que os membros da equipa estejam num mesmo local como em outros métodos ágeis. Porém, o desenvolvimento distribuído está relacionado principalmente com as capacidades da equipa, que devem ser as mais especializadas possíveis. Para isso, o ASD aconselha algumas técnicas para aumentar a colaboração entre a equipa e sugere estratégias para partilhar de informação e utilização de ferramentas de comunicação para apoiar o desenvolvimento distribuído.

3.2.2.1.1 Funcionamento

Um projeto ASD é focado em resultados, não em tarefas, e os resultados são as características ou funcionalidades desenvolvidas durante as iterações.

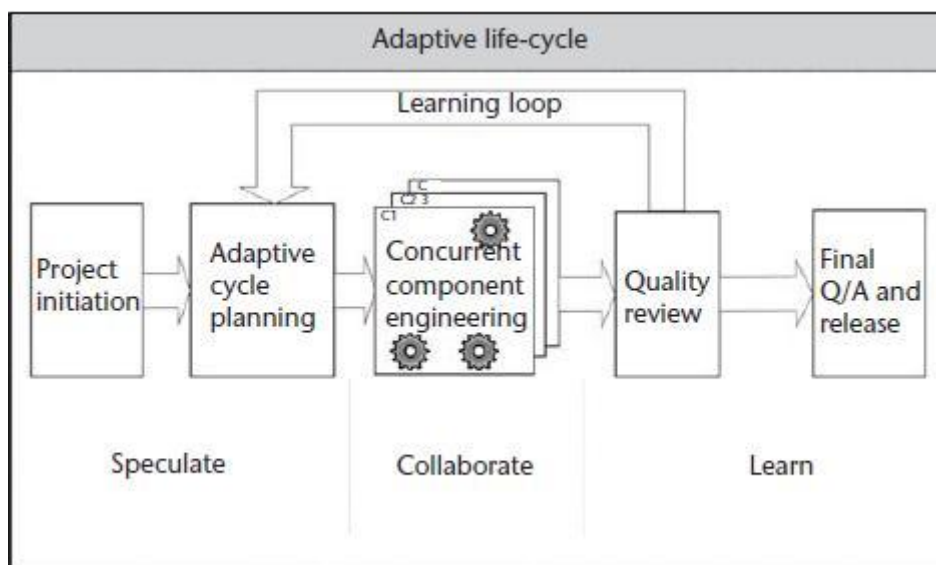


Figura 13 ciclo de vida ASD (extraído de (Rodrigo, 2011))

Um projeto ASD é levado a cabo em ciclos de três-fases (conforme Figura 13), nomeadamente:

- **Especulação** (*Speculate*): consiste na definição dos prazos, objetivos e de um plano baseado em características ou componentes, pois todo processo é focado em partes do sistema. Esta fase possui duas subfases chamadas Início do Projeto e Planeamento do Ciclo Adaptável. As características a serem desenvolvidas em cada iteração são decididas em conjunto pelos programadores e clientes, bem como a ordem em que serão implementadas. Estas definições são feitas em sessões de *Joint Application Development* (JAD), estas sessões são essencialmente *workshops*, onde programadores e representantes do cliente reúnem-se para discutir as características do produto desejado, e para melhorar a comunicação. O ASD não propõe o estabelecimento de uma agenda para as secções de JAD, mas salienta a importância de que aconteça no início do projeto.
- **Colaboração** (*Collaborate*): enfatiza a importância da equipa desenvolver software altamente flexível, esta fase é composta por várias iterações, onde acontece o desenvolvimento das características escolhidas para fazerem parte da mesma. Estas características podem ser desenvolvidas simultaneamente. Como o ASD é explicitamente orientado a componentes, ao contrário de orientado a tarefas, isso significa que o foco está mais nos seus resultados e qualidade do que em tarefas e os processos usados para produzir o resultado.
- **Aprendizagem** (*Learn*): realça a necessidade de reconhecer e reagir aos erros, e o facto que os requisitos também podem mudar durante o desenvolvimento. Esta fase divide-se em duas subfases: Revisões de Qualidade e Entrega Final. As revisões de qualidade acontecem no final de cada iteração, onde é apresentado um software que contempla os requisitos escolhidos para serem desenvolvidos durante a iteração. Estas revisões possuem o objetivo de verificar: a qualidade funcional do sistema através da definição de grupos de clientes para testar a aplicação; a qualidade técnica, onde um par de programadores é responsável por rever e avaliar o código do sistema; o desempenho da equipa, através de revisões das tarefas realizadas durante a iteração; o progresso geral do projeto, onde são verificados se os objetivos da iteração foram alcançados e a subfase de Entrega Final consiste na entrega final do sistema para o cliente.

Estes ciclos dinâmicos fornecem um contínuo ciclo de aprendizagem e adaptação à fase do projeto. Cada ciclo tem a duração em média de quatro a oito semanas.

3.2.2.2 Agile Unified Process (AUP)

O *Agile Unified Process* (AUP), em português Processo Unificado Ágil, é uma versão simplificada do RUP (ver 3.1.4 Modelo RUP) mas que incorpora princípios ágeis (Scott 2009).

O AUP descreve uma aproximação ao desenvolvimento de software, de forma simples, através da utilização de técnicas ágeis e conceitos vindos do RUP.

A seguinte descrição deste método resulta maioritariamente da análise e consulta da página web, intitulada “Agile Modeling and the Rational Unified Process (RUP)” de Scott W. Ambler.

3.2.2.2.1 Características

Pelas características deste método, este pode ser enquadrado entre os métodos XP (ver 3.2.2.7 Extreme Programming (XP)) e o RUP. O XP perde por não mostrar explicitamente como se criam alguns artefactos que os gestores querem, por outro lado, na perspetiva dos programadores, o RUP é extenso no número de artefactos que disponibiliza. Portanto o AUP situa-se entre estes dois métodos, ao adotar muitas das metodologias ágeis do XP mas também alguma formalidade do RUP.

A filosofia do AUP vai no sentido de tentar agilizar ao máximo o processo de desenvolvimento de software através dos seguintes conceitos:

- Não há a necessidade de estudar extensivamente a documentação do processo por parte da equipa de desenvolvimento, bastará algumas orientações ou formação de vez em quando;
- Evitar a documentação extensa, procurando antes descrever tudo de forma simples e concisa;
- Não obrigar a uma rígida utilização de ferramentas predefinidas;
- Basear-se nos valores e princípios do manifesto ágil (ver secção 3.2.1 Manifesto ágil);
- O foco apenas nas atividades em curso em detrimento da tentativa de atender, ao mesmo tempo, outras as possíveis atividades que ocorrerão ao longo do projeto.

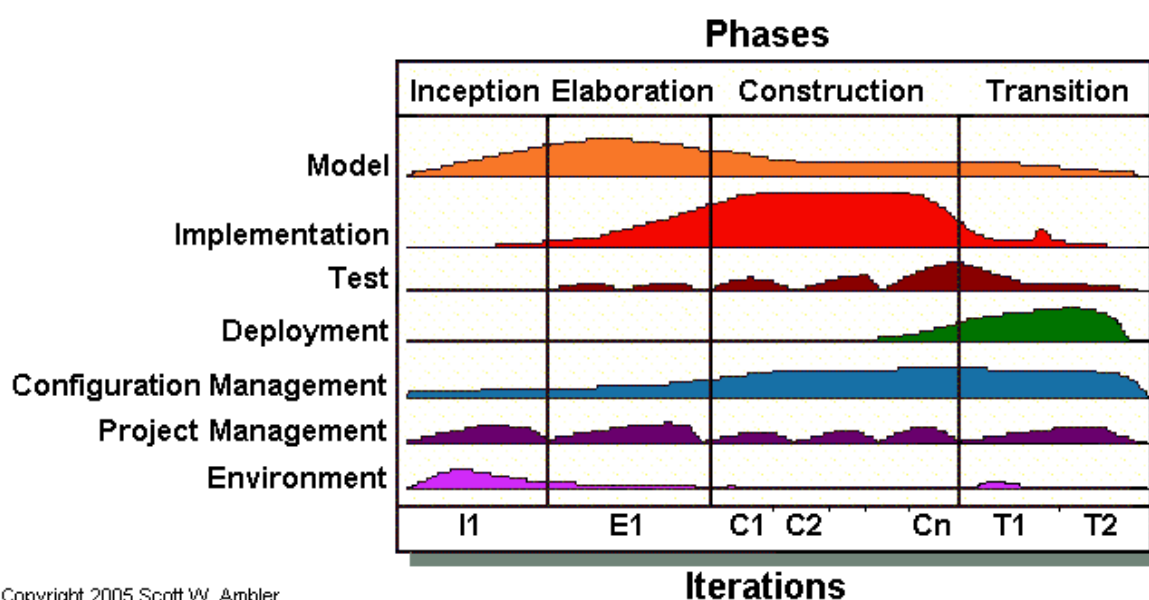
3.2.2.2.2 Papéis

O AUP não menciona papéis ou aborda este tema, nesta conjectura podem ser adotados os mesmos que os recomendados no modelo RUP.

3.2.2.2.3 Funcionamento

Em discordância com o modelo RUP, o AUP (conforme Figura 14) considera apenas sete disciplinas. Essas disciplinas são realizadas de forma iterativa, definindo as atividades que os membros da equipa de desenvolvimento irão executar para construir, validar e entregar o software de trabalho que atenda às necessidades das partes interessadas. As disciplinas são:

1. **Modelo** (*Model*): em que se pretende conhecer o negócio da organização, o domínio de problema a ser abordado pelo projeto, e identificar uma solução viável para resolver o domínio do problema;
2. **Implementação** (*Implementation*): a transformação do modelo em código executável e executar um nível de básico de teste, em particular, testes unitários;
3. **Teste** (*Test*): validação da implementação de modo a garantir a qualidade do sistema;
4. **Instalação** (*Deployment*): forma de planejar e disponibilizar o sistema para os utilizadores finais;
5. **Gestão de Configuração** (*Configuration Management*): gestão de acessos e de versões dos artefactos do projeto;
6. **Gestão de Projetos** (*Project Management*): gestão e monitoria das atividades que acontecem no projeto;
7. **Meio Ambiente** (*Environment*): apoio de esforço de modo a garantir que os processos a seguir e as ferramentas estão disponíveis para a equipa de projeto usar.



Copyright 2005 Scott W. Ambler

Figura 14 ciclo de vida do AUP (extraído de (Scott 2009))

O método AUP refere quatro fases, estas fases são as mesmas que as sugeridas pelo RUP, nomeadamente: conceção, elaboração, construção e transição.

No entanto, o AUP distingue dois tipos de iterações, uma no ambiente de testes e outra no ambiente de produção. A libertação da solução desenvolvida resulta na instalação num ambiente para estágio da solução, em que esta possa ser validada, tanto por testes unitários pela equipa de desenvolvimento ou por testes de aceitação através da execução de testes pelo cliente. A outra iteração diz respeito à instalação da solução, para o ambiente de produção. Esta diferenciação de ambientes para instalação da solução é um refinamento significativo relativamente ao RUP.

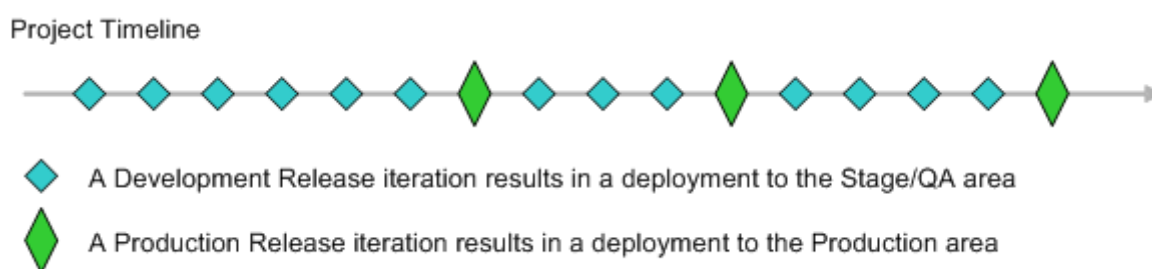


Figura 15 Linha do projeto (extraído de (Scott 2009))

3.2.2.3 *Crystal*

Crystal é o nome de uma família de métodos que são adotados de acordo com as características específicas de cada projeto, dessa forma, pode ser aplicado a projetos de tamanhos e complexidades diferentes (Cockburn, 2008).

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web, criada pelo próprio autor do método o Alistair Cockburn.

3.2.2.3.1 Características

Dentro da família Crystal cada método é moldado para ter a quantidade suficientemente adequada de processos e capaz de responder aos projetos a partir da análise de três fatores: a carga de comunicação (representada pelo número de pessoas), a criticidade do sistema e a prioridade do projeto. Na Figura 16 é demonstrado, numa matriz, a forma como cada método da família Crystal deve ser utilizado. O Crystal caracteriza os projetos ao definir cores diferentes a cada um, tais cores variam entre amarelo, laranja e vermelho, ou seja, *Clear*, *Yellow*, *Orange*, *Red* respetivamente. As cores são definidas com base na complexidade, tamanho e importância dos projetos, em que quanto maior e mais rigoroso é o projeto então a

sua cor será mais forte entre as anteriores. Esse peso é representado pela quantidade de artefactos e rigidez da gestão do projeto.

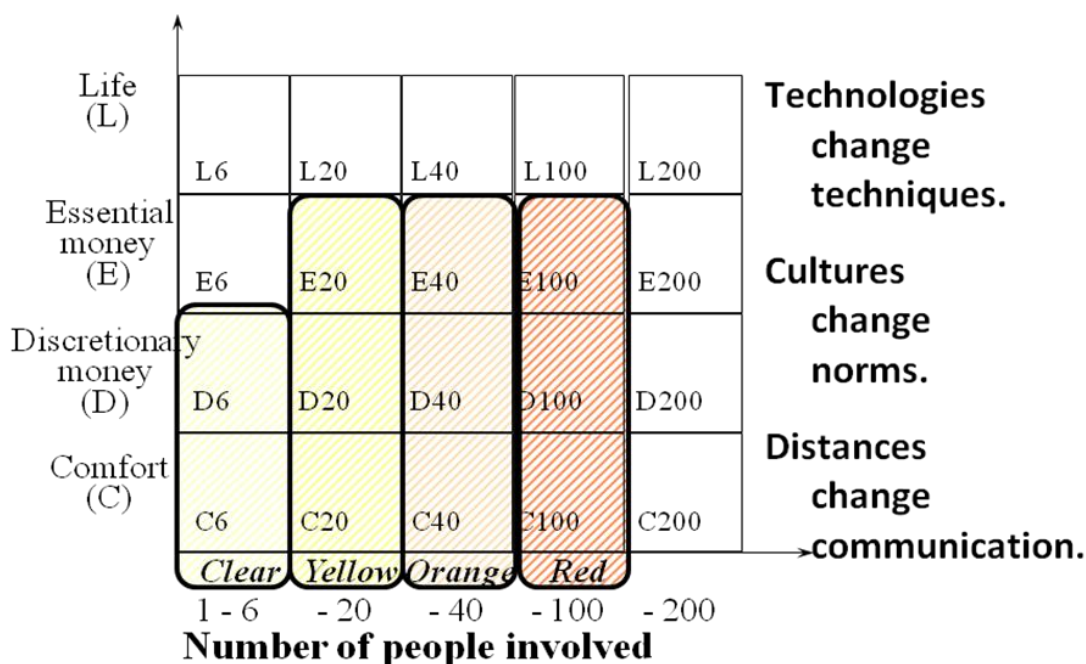


Figura 16 Distribuição dos métodos da família *Crystal* (extraído de (Cockburn, 2008))

Apesar das diferenças entre as complexidades dos projetos abordados pelos métodos da família *Crystal*, eles apresentam em comum valores, princípios e a capacidade de serem ajustados durante a sua utilização (*on-the-fly*). Os valores indicam a visão do desenvolvimento de software como um método iterativo, criativo e de comunicação, com o objetivo primário de entregar algo útil e, em segundo, preparar para a próxima incremento. Dessa forma, os valores estipulam que os métodos são centrados nas pessoas e na comunicação, além de serem altamente tolerantes a mudanças. No entanto, existem duas regras que devem ser seguidas, limitando essas mudanças: o desenvolvimento deve ser incremental, com incrementos de até quatro meses, e devem ser realizados workshops de retrospectivas antes e depois de cada incremento.

De acordo com o autor, este método deve ter em conta os seguintes sete princípios:

1. Preferir **comunicação iterativa**, comunicação cara-a-cara como o canal mais barato e rápido para troca de informações;
2. O **excesso de peso** do método é custoso;
3. **Equipas com maior tamanho** ou distribuídas necessitam de métodos mais pesados;
4. Usar **mais formalidades** para projetos de maior criticidade;

5. Ao **aumentar a retroalimentação** (com *feedback*) e comunicação é reduzida a necessidade de entregas intermediárias;
6. **Disciplina, capacidade** para entender os processos, formalidades e documentação;
7. Pode se perder eficiência em atividades que não são o foco do processo.

3.2.2.3.2 Papéis

Num método baseado em pessoas e não em processos, então, os papéis e as suas responsabilidades são aspetos muito importantes. A família de métodos Crystal aborda explicitamente papéis em dois dos seus métodos (Crystal *Clear* e Crystal *Orange*, referidos daqui a diante por apenas *Clear* e *Orange* respetivamente). Tanto o *Clear* como o *Orange* possuem diversos papéis em comum. No entanto, o *Orange*, ao ser um método mais “pesado” requer assim mais papéis, mas as diferenças entre estes métodos não terminam pela quantidade de papéis, a principal diferença entre estes métodos é o número de equipas que abrangem. Enquanto o *Clear* não possui uma estrutura de comunicação adequada para suportar mais do que uma equipa isto não acontece no *Orange*, que possibilita e preceitua o uso de diversas equipas multifuncionais, ou seja, com profissionais com diversos papéis juntos.

Nestes métodos uma pessoa pode assumir mais que um papel, ou seja, permite a acumulação de responsabilidades. Os principais papéis, necessários para ambos os métodos são (Brito, 2011):

- **Patrocinador**, ou seja, quem financia o projeto;
- **Utilizador**, apenas a tempo parcial, é quem utiliza o software desenvolvido;
- **Programador/ Analista Sênior** é o membro mais experiente em programação e análise/ projeto de sistemas;
- **Programador/Analista** é o membro menos experiente, mas capaz de programar e conceber/ analisar um sistema.

O Crystal *Orange*, entretanto, inclui um variado número de papéis a este conjunto, sendo a grande maioria de profissionais mais especializados, como *Designer* de Interfaces com o utilizador, analista de bases de dados, gestor do projeto, arquiteto, analista de requisitos, analista do negócio, entre outros. Entre tantos papéis, são de salientar dois que merecem um pouco mais de atenção, são eles:

- **Escritor** (*writer*): responsável pela produção de documentação externa, como manual de utilização, por exemplo;

- **Analista de negócio** (*business expert*): responsável por negociar e comunicar com os utilizadores de modo a obter o que deve ser especificado em termos de requisitos e interfaces, e também para rever o *design* do projeto.

Uma equipa de trabalho no *Crystal Orange* deve ser composta por, no mínimo: um analista de negócio, um *designer* de interfaces com o utilizador, dois a três programadores/ analistas, um analista de base de dados e, se possível, um testador.

3.2.2.3.3 Funcionamento

Existem algumas características comuns à família Crystal, tais como o desenvolvimento incremental com ciclos de, no máximo, como já referido quatro meses, ênfase maior na comunicação e cooperação das pessoas, não limitação de quaisquer práticas de desenvolvimento, ferramentas ou produtos de trabalho e incorporação de objetivos para reduzir produtos de trabalho intermediários e desenvolve-los como projetos evoluídos.

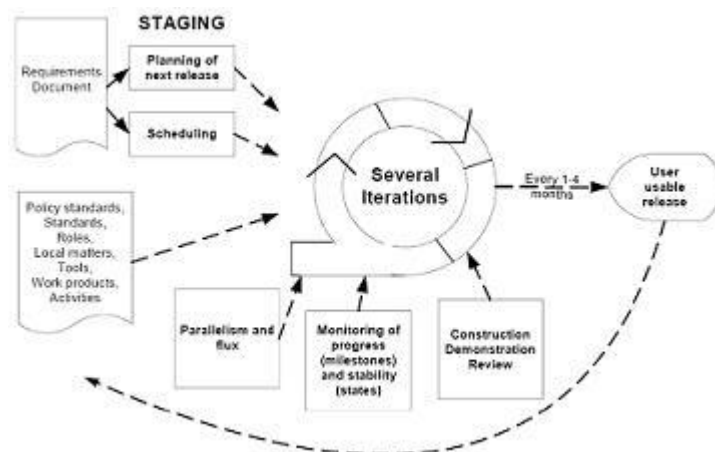


Figura 17 Ciclo de vida da família Crystal (extraído de (Brito, 2011))

O ciclo de vida desta família de métodos (Figura 17) é baseado nas seguintes práticas:

- Estágio com o planeamento e agendamento;
- Regras, padronizações, procedimentos, papéis, produtos de trabalho e ferramentas;
- Monitoria do progresso;
- Paralelismo;
- Construção, demonstração e Revisões;

- Construção, demonstração e revisão dos objetivos do incremento;
- Incremento utilizável para o utilizador.

No *Orange* a etapa em que o documento de requisitos é criado é obrigatória, já que é um documento importante para a gestão de várias equipas. No entanto, não é necessário no *Clear* porque pressupões apenas uma equipa a trabalhar no mesmo local. Outras etapas, como agendamento (*Scheduling*) e revisões (*Reviews*) são mais aprofundadas no *Orange*, e mais brandas no *Clear*. As práticas de paralelismo e fluxo, apenas se aplicam ao *Orange*, já que no *Clear* não há múltiplas equipas, por isso não existe a necessidade de paralelismo nas atividades.

3.2.2.4 Dynamic System Development Method (DSDM)

O Dynamic Systems Development Method (DSDM), em português Método de Desenvolvimento de Sistemas Dinâmicos, é um método de desenvolvimento de software baseado em desenvolvimento rápido de aplicações, ou seja, baseado em RAD. O DSDM é um método de desenvolvimento iterativo e incremental que enfatiza o envolvimento constante do utilizador.

Este método foi desenvolvido, nos anos 90, por um consórcio de vendedores e peritos na área de Sistemas de Informação (SI), no qual partilharam e combinaram as suas melhores técnicas. Atualmente a versão em utilização é o *DSDM Aftern* (DSDM Consortium, 2008).

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web, e materiais disponíveis a título académico, disponibilizados pelo consórcio criador do método, o DSDM Consortium.

3.2.2.4.1 Características

O objetivo principal do DSDM é a entrega de software no tempo e com o custo estimados, através do controlo de requisitos ao longo do desenvolvimento. Este método é mais adequado a projetos caracterizados por prazos e orçamentos apertados. Para conseguir atingir o objetivo o DSDM foca-se na interação com o cliente, utilizador final e a equipa de programadores, com a realização de entregas de protótipos frequentes, com uma equipa de desenvolvimento autónoma, com testes padronizados e com a definição de prioridades, dos requisitos, identificada pelos clientes.

A DSDM possui alguns princípios fundamentais (Figura 18), que delimitam as bases do desenvolvimento.

- Foco nas **necessidades de negócio**;
- **Entrega na data prevista**, dentro do orçamento previsto e com boa qualidade;
- **Colaboração** e cooperação entre os membros da equipa de desenvolvimento;
- **Nunca comprometer a qualidade** garantindo a execução de todas as atividades necessárias;
- **Construir gradativamente** a partir de bases sólidas;
- Desenvolvimento de **forma iterativa**;
- **Comunicação contínua** e clara;
- **Demonstrar controlo** através de monitoria continua.

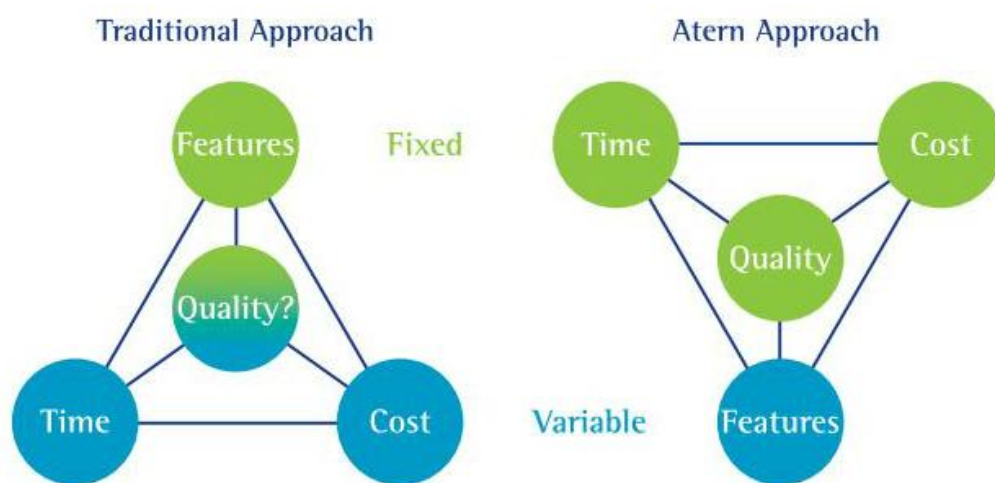


Figura 18 Princípios fundamentais do DSDM (extraído de (DSDM Consortium, 2008))

Além dos princípios enumerados existem também algumas técnicas, estas são utilizadas durante a execução de um projeto com DSDM:

- **Time-boxes**: definição de um período fixo para a execução do projeto e definir datas para a entrega do produto;
- **MoSCoW**: regra básica para priorização dos requisitos, de modo a considerar para implementação em primeiro os requisitos mais importantes deixando os menos importantes para depois;
- **Modelagem**: deve ser usada para entender melhor o problema e a solução;
- **Prototipagem**: forma de verificar a adequação dos requisitos e facilitar a análise com o cliente;
- **Testes**: esta atividade deverá ser executada sistematicamente e de forma contínua durante o projeto;

- **Gestão de configurações:** este registo é muito importante porque com este método pressupõem-se a entrega de produtos com muita frequência.

3.2.2.4.2 Papéis

O DSDM enfatiza a importância da comunicação entre todos os elementos envolvidos no projeto e identifica os seguintes papéis a desempenhar na utilização deste método:

- **Gestor executivo** (*Business Sponsor*): papel importante na definição de prazos e recursos e tem a última palavra na tomada de decisões;
- **Visionário** (*Business Visionary*): tem a percepção apurada dos objetivos de negócio do sistema e projeto;
- **Gestor do projeto** (*Project Manager*): gere o projeto com um todo do início ao fim do projeto;
- **Coordenado técnico** (*Technical Co-ordinator*): responsável no desenho da arquitetura do sistema e controlo da qualidade técnica do projeto;
- **Líder de equipa** (*Team Leader*): lidera a equipa e mantém a harmonia do projeto e trabalho em grupo;
- **Programador** (*Solution Developer*): interpreta o modelo e requisitos do sistema incluindo desenvolvimento de artefactos de código e construção de protótipos;
- **Testador** (*Solution Tester*): verifica o funcionamento da parte técnica através da execução de algumas tarefas de teste;
- **Facilitador** (*Workshop Facilitator*): gere o progresso dos grupos de trabalho, age como motor de preparação e comunicação;
- **Papéis específicos:** Arquiteto de negócios, Gestor de Qualidade, Integrador de Sistema, etc.

3.2.2.4.3 Funcionamento

O ciclo de vida de um projeto a usar o método DSDM (Figura 19) consiste em três fases sequenciais: Pré-projeto, Projeto e Pós-projeto, em que a segunda fase, “Projeto”, é a mais elaborada das três fases.

Fase 1: O Pré-projeto

No pré-projeto são tratadas as questões iniciais, como o plano financeiro e o cronograma, assegurando-se um compromisso de realização. O tratamento destas questões nesta fase evita problemas futuros nas fases mais avançadas do desenvolvimento do projeto.

Fase 2: O Projeto

Nesta fase há 5 níveis que a equipa de desenvolvimento terá que percorrer para criar o software. Esses níveis são: Estudo de Viabilidade; Estudo de Negócio; Análise Funcional; Desenho e Implementação.

Fase 3: O Pós-projeto

Esta fase corresponde à manutenção do sistema e assegura que o software funciona corretamente.

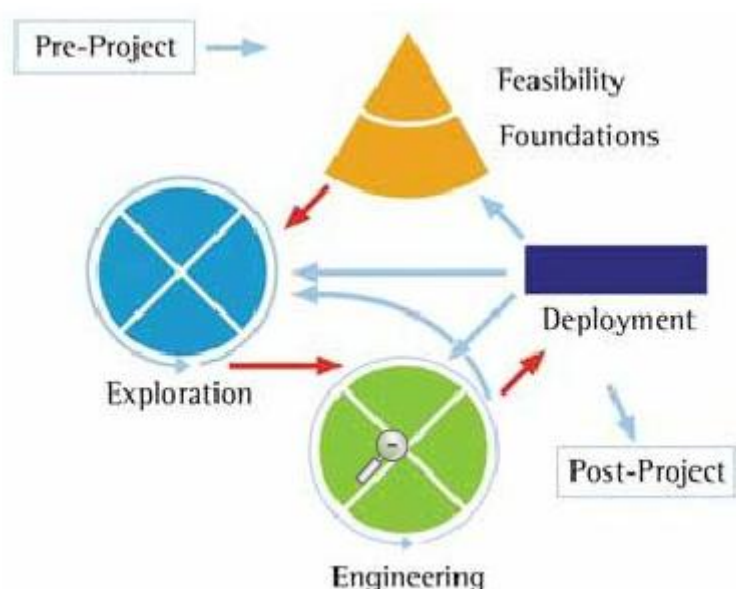


Figura 19 Ciclo de vida de um projeto com DSDM (extraído de (DSDM Consortium, 2008))

3.2.2.5 EasyProcess (YP)

O EasyProcess (YP), em português Processo Fácil, é um método com origem no Brasil e foi criado com o intuito de satisfazer as necessidades do desenvolvimento de projetos em ambiente académico. Este método foi idealizado pela Professora Dr^a Francilene Procópio Garcia da Universidade Federal de Campina Grande (UFCG), no ano de 2003. Este método está apoiado em práticas do XP, RUP e no Manifesto Ágil, na utilização do que há, de mais adequado, em cada um dos métodos enumerados em relação ao ambiente académico (Garcia & et al, 2006).

A seguinte descrição deste método resulta maioritariamente da consulta da página web, e materiais disponibilizados pela equipa de desenvolvimento que deu origem ao método.

3.2.2.5.1 Características

O YP foi moldado com base nas características de um projeto de desenvolvimento de software em âmbito académico, para se adequar a esta utilização este método deveria satisfazer as seguintes condições:

- **Adequado às características destes projetos de desenvolvimento**, com âmbito pequeno, em virtude da duração de um semestre lectivo (quatro meses), e a utilização de tecnologias consideradas estado da arte. Além disso, deveria possibilitar uma avaliação contínua e iterativa ao longo do projeto;
- **Reduzida curva de aprendizagem** para uma boa produtividade do projeto, bom uso de ferramentas de apoio com o mínimo de artefactos elaborados, mas suficientes para o entendimento e consolidação das práticas e conceitos da engenharia de software;
- Ser de fácil entendimento e simples de se seguir, mas **robusto e completo** o suficiente para gerar produtos de qualidade. Ao mesmo tempo deverá ter em conta a inexperiência da equipa de desenvolvimento e a imaturidade na gestão de projetos de desenvolvimento sem o devido acompanhamento;

3.2.2.5.2 Papéis

O EasyProcess recomenda a utilização de cinco papéis: cliente, utilizador, gestor do projeto, programador e testador. Estes papéis podem ser acumulados entre os membros da equipa.

- **Cliente:** é a pessoa que solicitou o desenvolvimento do software. Terá a responsabilidade de definir os requisitos funcionais do sistema, definir prioridades e aprovar os prazos para entrega, assim como aprovar os protótipos e outros artefactos necessários;
- **Utilizador:** é a pessoa que irá utilizar o sistema produzido, portanto o desenvolvimento do software deve atentar para as características desta pessoa. Logo a sua participação ao longo do projeto será crucial para o sucesso do mesmo;
- **Gestor de projeto:** É aquele que toma decisões ao longo do projeto e é também o responsável pela coordenação das atividades de todos os outros membros do projeto;
- **Programador:** é responsável pelo levantamento de requisitos, conceção e construção do sistema de acordo com os requisitos definidos e execução de testes unitários;

- **Testador:** é responsável pela realização de testes de integração do sistema, por rever o código dos programadores, simplificar o código quando possível garantindo a qualidade do sistema.

3.2.2.5.3 Funcionamento

A Figura 20 resume as principais atividades e o fluxo de trabalho que ocorre num projeto a utilizar o YP.

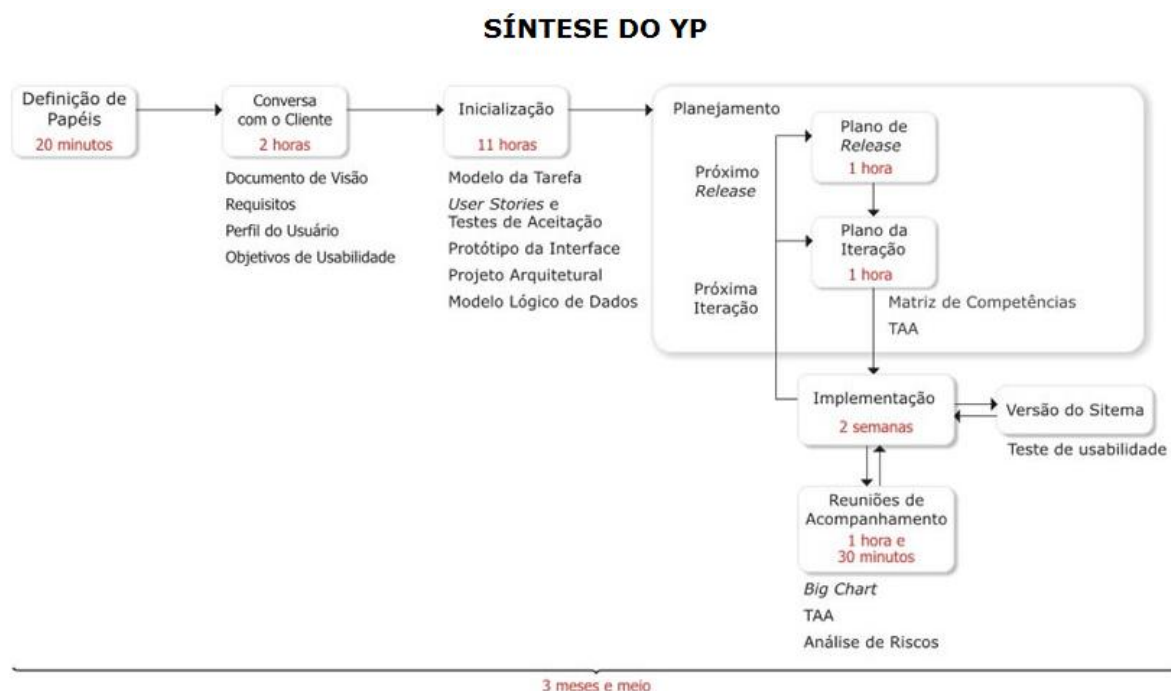


Figura 20 Síntese do método YP (extraído de (Garcia & et al, 2006))

De acordo com o fluxo ilustrado acima as principais fases para o desenvolvimento de software são:

1. **Identificação do âmbito** do problema;
2. **Definição dos papéis**;
3. **Conversa com o cliente** para levantamento de necessidades do sistema;
4. **Inicialização** com o levantamento das user stories, criação do protótipo de interface e elaboração do modelo lógico de dados;
5. **Planeamento** para a alocação das user stories nas *releases*, definição de cada *release* e das iterações necessárias;
6. **Implementação** com a codificação do sistema e execução de testes unitários e de integração;
7. **Finalização da iteração** com a realização de uma reunião de acompanhamento.

Os passos de 5 a 7 deverão ser repetidos tantas vezes quantas *releases* ou as iterações forem definidas. Por fim, o resultado será uma versão do produto desenvolvido depois da realização de testes de aceitação pelo cliente ou utilizador.

3.2.2.6 Essential Unified Process (ESSUP)

O método Essential Unified Process (ESSUP), em português Processo Unificado Essencial, foi criado por Ivar Jacobson como uma melhoria ao método RAP. Este método identifica práticas, tais como casos de uso, desenvolvimento iterativo, arquitetura de desenvolvimento orientações práticas, a equipa e as práticas de processo, que são originados dos modelos RUP, CMMI e pelo manifesto ágil. De uma forma geral a ideia do ESSUP é poder escolher as práticas a aplicar em cada situação e combiná-los no seu próprio processo de desenvolvimento de software. Esta ideia é considerada uma melhoria em relação ao modelo RUP, porque as práticas RUP estão todas interligadas e não podem ser consideradas isoladamente. Por isso o ESSUP é considerado um método muito mais simples, flexível e extensível que as anteriores expressões do processo unificado.

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web, e materiais disponíveis a título académico, disponibilizados pelo criador do método, na página de “Ivar Jacobson International”.

3.2.2.6.1 Características

A utilização do ESSUP pressupõe apenas o uso das práticas que realmente precisa e uma adaptação do processo às necessidades de cada projeto, em vez de usar um modelo mais formal e complexo, e assim utiliza uma taxonomia simples e sensata. Assim, com a utilização de um pequeno número de categorias o ESSUP permite práticas de processos diversos e os seus artefactos são facilmente descritos, organizados e comunicados.

A abordagem do ESSUP, pelo seu criador, é que este método centra-se na essência e fornece referências para as comunidades fechadas de desenvolvimento de software, em vez de fornecer páginas e páginas de casos de uso, ou arquitetura ou padrões que simplesmente referencia os melhores papéis, livros na área. Isto é feito explicitamente para evitar qualquer tentativa de aparecer como a fonte de autoridade normativa, para tornar claro que os profissionais consultores deste método têm conhecimento que é essencial para ser capaz de realizar as práticas de desenvolvimento de software (Thomas, 2006).

No entanto, como referido na página oficial do criador do método, está a ser explorada a melhor maneira de criar uma comunidade ESSUP num formato aberto para permitir que outros a partilhar as suas práticas. Dessa forma, será permitida a disponibilização aberta de informações das práticas para consumo pela comunidade em geral.

3.2.2.6.2 Papéis

Apesar de não detalhadamente o método refere que a equipa deverá ter elementos com competências em: *design* de software, programação, testes e integração. No entanto, informações mais completas sobre os papéis e o funcionamento do processo não estão disponíveis de forma aberta para o público em geral.

3.2.2.7 *Extreme Programming (XP)*

O Extreme Programming (XP), ou Programação Extrema na tradução direta, é provavelmente o método ágil mais famoso e discutido atualmente.

Este método foi criado durante a década de 90 por Kent Beck (um dos participantes no Manifesto Ágil), é direcionado a projetos a curto e médio prazo e com pequenas e médias equipas que desenvolvem software baseado em requisitos imprecisos e instáveis (Jeffries, 2012).

Segundo o autor do método, o XP propõe um processo leve, centrado no desenvolvimento iterativo e com a entrega constante de pequenas partes da funcionalidade do software. As partes devem ser incrementais e requerem a melhoria constante do código (retrabalho). O XP parte do pressuposto que a melhor documentação é o código fonte, pois qualquer outra documentação fica logo desatualizada e por esse motivo perde confiabilidade. O XP destina-se idealmente a grupos de dois a dez programadores, a projetos com duração prevista entre um mês a três anos e com produção de cerca de mil a duzentos e cinquenta mil linhas de código.

A seguinte descrição deste método resulta maioritariamente da consulta de páginas web, disponibilizados pelas comunidades que usam o método, de destacar: <http://xprogramming.com> e <http://www.extremeprogramming.org>.

3.2.2.7.1 Características

Para atingir o sucesso, nos projetos de desenvolvimento de software, o XP emprega quatro valores fundamentais, alguns princípios e doze práticas que, conforme o próprio nome do método sugere, estas são levadas ao extremo.

Em relação aos valores, estes definem como as equipas vão proceder durante o processo de desenvolvimento, o XP aplica os seguintes:

- **Comunicação:** promover a comunicação entre as partes envolvidas do projeto, ao preferir conversas pessoais em detrimento de outros meios de comunicação e o trabalho em conjunto em vez da revisão do resultado final;
- **Simplicidade:** ao fazer algo da forma mais simples, possível e funcional, o *design* é simplificado continuamente e até o próprio processo é adaptado a cada dia, caso alguém preveja como torna-lo mais simples. A simplicidade visa a construção do código mais simples e não sem inclusão de funcionalidades desnecessárias, portanto apenas são implementados os requisitos atuais e evita-se adicionar funcionalidades que apenas serão importantes no futuro. O XP aposta no fazer algo simples hoje e pagar mais um pouco mais amanhã caso seja necessário fazer alterações em detrimento de implementar algo complicado hoje que talvez não venha a ser usado, sempre com a premissa que os requisitos são mutáveis;
- **Feedback:** permitir *feedback*, ou seja, a retroalimentação de informação de forma rápida e frequente, isto que permite a evidenciar um problema o mais cedo possível para que possa ser corrigido atempadamente, assim como as oportunidades são descobertas o mais cedo possível e assim aproveitadas também o mais cedo possível;
- **Coragem:** capacidade de assumir riscos e desafios em favor do projeto, é necessária coragem para apontar um problema no projeto, para pedir ajuda quando necessário, simplificar código que estava já a funcionar, comunicar ao cliente que um dado requisito não vai ser implementado no prazo estimado, entre outras situações que apesar de ser o mais certo a fazer-se não seria de todo o mais desejável.

Apesar de normalmente serem colocados em segundo plano o XP define também um conjunto de princípios, identificados no Quadro 2, com o objetivo de concretizar os referidos valores e de modo a facilitar a sua adoção pelas pessoas envolvidas. Estes princípios existem para servir de ponte entre os valores (crenças fundamentais dos processos) e as práticas (atividades do quotidiano) e servem como guias que se aplicam a um domínio específico.

Princípios Básicos	Outros princípios	
• <i>Feedback</i> rápido; • Simplicidade assumida; • Mudanças incrementais; • Receção de mudanças; • Trabalho com qualidade.	• Humanidade; • Economia; • Beneficio Mutuo; • Similaridade própria; • Progresso; • Diversidade; • Reflexão; • Fluxo;	• Oportunidade; • Redundância; • Falha; • Responsabilidade aceita; • Passos de bebé; • Medir honestamente; • Comunicação aberta e honesta; • Adaptação local.

Quadro 2 Princípios do XP

Esses princípios não indicam que o programador, quando estiver a trabalhar, apenas comece a escrever código rápido e desesperadamente, pelo contrário, é preciso ser disciplinado nesse sentido ou o efeito será contrário ao desejado.

A aplicação desses princípios num projeto de desenvolvimento de software é a base para a fixação do método XP.

Os valores e os princípios são o espírito do XP, mas é necessário mais que isso para criar um processo eficaz. Nesse âmbito foram definidas as práticas que detalham de maneira mais clara como e quando cada programador deve atuar e também organizar a equipa de desenvolvimento para tornar o processo do XP sólido. A seguir são apresentadas essas práticas:

• **Propriedade coletiva** (*Colletive Code Ownership*): o código do projeto pertence a todos os membros da equipa, portanto a equipa como um todo é responsável por cada arquivo de código. O que significa que qualquer membro da equipa, e por isso a equipa deverá ter competências multidisciplinares, que perceba que pode adicionar valor, mesmo que ele próprio não o tenha desenvolvido e pode fazê-lo sem pedir autorização para o fazer desde que faça também todos os testes necessários. Uma vantagem desta prática é que, caso um membro da equipa a deixe antes do término do projeto, a equipa consegue continuar o projeto sem dificuldades, pois todos conhecem todas as partes do software, mesmo que não seja de forma detalhada.

• **Planeamento** (*Planning Game*): consiste em decidir o que é necessário ser feito e o que pode ser adiado no projeto, ao combinar as prioridades do negócio pelo cliente e as estimativas técnicas pela equipa de desenvolvimento.

O cliente é essencial neste processo e assim ele fica a par do que está a suceder e o que vai acontecer futuramente no projeto. Como o âmbito é periodicamente reavaliado, o projeto é regido por um contrato de âmbito negociável, que difere significativamente das formas tradicionais de contratação de projetos de software. Desta forma, enquanto a área de negócio decide sobre o âmbito, a composição das versões e as datas de entrega, os programadores decidem sobre as estimativas de prazos, o processo de desenvolvimento e o cronograma detalhado para que o software seja entregue as datas específicas.

- **Testes** (*Customer Tests*): o XP evidencia a validação do projeto durante todo o processo de desenvolvimento. Os testes devem ser feitos de forma automatizada, os testes unitários devem ser feitos pelos programadores e os testes de aceitação devem ser executados pelos clientes, estes devem executar os testes funcionais que constituem os critérios de aceitação do software.

- **Entregas frequentes** (*Small Releases*): visa a construção de um software simples, e conforme surgem os requisitos, haverá lugar a atualizações de software. Cada versão entregue deverá ter o menor tamanho possível e os requisitos de maior valor para o negócio. Idealmente devem ser entregues versões a cada mês, ou no máximo a cada dois meses aumentando a possibilidade de feedback rápido do cliente. Assim, o software tem a possibilidade de evoluir com as avaliações do cliente o que aumenta a probabilidade do software final estar de acordo com o pretendido pelo cliente.

- **Projeto simples** (*Simple Design*): o design deve ser feito da maneira mais simples possível, ao focar nas tarefas atuais e não pensando em possibilidades futuras, ou seja, deve ser o mais simples possível de modo a satisfazer os requisitos atuais sem a preocupação de requisitos futuros. Um erro comum ao adotar essa prática é a confusão por parte dos programadores de código simples e código fácil. Nem sempre o código mais fácil de ser desenvolvido levará a solução mais simples por parte de projeto. Esse entendimento é fundamental para o bom funcionamento do XP. Código fácil deve ser identificado e substituído por código simples.

- **Programação em pares** (*Pair Programming*): a produção do código deve ser feita em duplas, num único computador, uma pessoa escreve o código e a outra auxilia e pensa de forma mais estratégica. Geralmente a dupla é formada por um iniciante na linguagem e outra pessoa funciona como um instrutor. Como é apenas um computador, o novato é que fica à frente a fazer a codificação, e o instrutor acompanha e ajuda a desenvolver suas habilidades. Desta forma o programa sempre é revisto por duas pessoas, o que evita e

diminui assim a possibilidade de defeitos. Com isto procura-se sempre a evolução da equipa e a melhoria da qualidade do código fonte gerado.

- **Integração contínua** (*Continuous Integration*): os diversos módulos do sistema são integrados diversas vezes por dia e os testes unitários executados com sucesso. Integrar apenas um conjunto de alterações ou novas funcionalidades de cada vez é uma prática que funciona bem porque assim quem deve fazer as correções quando os testes falham é a última equipa que integrou código novo ao software. Esta prática é facilitada com a utilização de apenas uma máquina para integração, que deve ter livre acesso a todos os membros da equipa. Integrar de forma contínua permite saber o estado real da programação.

- **Código padrão** (*Coding Standards*): a equipa de desenvolvimento necessita de estabelecer regras, a padronização da arquitetura do código permite que o mesmo seja partilhado entre toda equipa de desenvolvimento. Desta forma parecerá que todo o código fonte foi editado pela mesma pessoa, mesmo quando a equipa possui dez ou até cem elementos.

- **Metáfora** (*Metaphor*): são as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento de software. Esta prática procura facilitar a comunicação com o cliente, entendendo a realidade dele. Pois um mesmo conceito poderá ter interpretações diferentes consoante a sua contextualização e domínio de atuação, pelo que é preciso traduzir as palavras do cliente para o significado que ele espera dentro do projeto.

- **Refatoração** (*Refactoring*): é um processo que permite a melhoria contínua da programação, com o mínimo de introdução de erros e mantendo a compatibilidade com o código já existente. A cada nova funcionalidade adicionada o código deverá ser trabalhado até ficar na sua forma mais simples. Assim, o código é continuamente reestruturado pelos programadores, através da divisão em módulos mais coesos e de maior reaproveitamento, excluindo a duplicação de código e tornando-o mais simples após uma alteração o que facilita a clareza e leitura do código.

- **Ritmo Sustentável** (*Sustainable Pace*): trabalhar com qualidade, procurando ter um ritmo de trabalho saudável, entenda-se quarenta horas por semana em oito horas por dia, sem horas extras. As horas extras são permitidas quando trouxerem produtividade para a execução do projeto. Mas o XP assume que não se devem fazer horas extras constantemente, e caso seja necessário fazer mais de 40 horas pela segunda semana

consecutiva então existe um problema serio no projeto que deve ser resolvido não com o aumento de horas trabalhadas mas, por exemplo com um melhor planeamento. Esta prática procura ratificar o foco nas pessoas e não nos processos e planeamento, caso necessário os planos deverão ser alterados em vez de sobrecarregar as pessoas. Outra costume que se verifica neste processo é a prática de trabalho energizado e motivado. Para isso o ambiente de trabalho e a motivação da equipa devem estar sempre em harmonia.

- **Cliente presente** (*The Costumer Is Always Available*): é fundamental a participação do cliente durante todo o desenvolvimento do projeto, portanto o cliente deverá estar sempre disponível para resolver dúvidas evitando atrasos e até mesmo construções erradas, alterar o âmbito de uma iteração e definir prioridades. O cliente poderá mesmo fazer parte integrante da equipa de desenvolvimento.

Muitas destas práticas são alvos frequentes de críticas, como a programação em pares, considerado um desperdício de pessoal, e o refatoração que seria uma forma de retrabalho devido a um projeto mal feito. No entanto, não é adequado julgar as práticas de forma isolada, já que cada uma delas tem um papel no processo, por esse mesmo motivo, alterar alguma prática necessita um profundo entendimento do XP e da equipa de trabalho, para não criar um problema sério no processo.

3.2.2.7.2 Papéis

Numa equipa de XP existem papéis que podem ser desempenhados por um ou mais programadores, os papéis e responsabilidades no XP são:

- **Cliente** (*XP Gold Owner*): define o que é o projeto e como deve ser desenvolvido, o termo de cliente pode não estar associado a uma pessoa individual, mas sim a um conjunto de pessoas que define e dá prioridade as “*user stories*”, define os requisitos do projeto e descreve os testes que cada fase de desenvolvimento tem de passar antes de se passar à fase seguinte. O cliente poderá ser ou não o utilizador final do software, no entanto a sua participação é tão valiosa quanto for o seu conhecimento sobre o domínio do negócio e é ainda melhor quando os clientes são utilizadores dos sistemas e têm experiência com sistemas semelhantes no passado.
- **Gestor do projeto**: é o responsável pelos assuntos administrativos do projeto. Deve possuir grande aptidão para lidar com pessoas, visto que é ele que trata dos assuntos ligados ao projeto com a equipa e com os clientes. É um tipo de mediador entre os

interesses da equipa e do cliente. É também responsável por tentar manter o cliente informado do projeto em questão e, ainda mais importante para a empresa, participa no desenvolvimento de forma direta. Obviamente, um bom gestor deve acreditar em todos os valores de XP para que possa cobrar isto da sua equipa. Outro papel do gestor é manter as ameaças externas longe da equipa sob sua supervisão para tal deve filtrar assuntos não relevantes aos programadores e aspetos que só devam ser implementados em iterações futuras.

- **Treinador** (*XP Coach*): é o responsável pelas questões técnicas do projeto de software. Orienta a equipa, tentando garantir que esta se mantém fiel à metodologia XP, para isso o treinador deve possuir grande conhecimento em todas as fases do XP, sentido de liderança e conhecer bem os membros da equipa. É o responsável por estimular a equipe a dar o melhor sempre que possível. Ao conhecer e aplicar os conceitos de XP, deve estar apto para sinalizar a sua equipa os erros cometidos ao longo do projeto e dar os passos necessários para consertá-los. O seu papel diminui à medida que a equipa evolui.

- **Programador**: é o papel principal no XP. O programador é responsável por analisar, conceber e escrever o código para o sistema em desenvolvimento. Numa equipa de programadores que utilizam o XP, os programadores estão todos no mesmo nível. Por exemplo, não existe diferença entre analista e programador uma vez que eventualmente todos executarão uma dessas atividades. Os programadores estão presentes em todas as fases do desenvolvimento, desde em conjunto com o cliente na criação das “user stories”, no desenvolvimento do código e testes e participam nas reuniões periódicas de planeamento.

- **Analista de testes**: responsável por garantir a qualidade do sistema desenvolvido através da atividade de testes do sistema. Normalmente o analista de testes ajuda o cliente a escrever os casos de testes necessários para garantir que os requisitos serão devidamente satisfeitos. No final de cada iteração, deve realizar os testes no software a ser entregue com o objetivo de verificar se existem defeitos e requisitos não-cumpridos. É perigoso usar o programador do software para testá-lo e construir casos de teste junto ao cliente, pois o que pode acontecer é que, pelo facto do programador conhecer o código, ele terá uma visão tendenciosa, e poderá evitar os pontos que ele sabe serem sensíveis no sistema. Por outro lado, o analista de testes deve ter uma visão muito parecida com a do cliente.

- **Testador:** tem como objetivo realizar testes funcionais do projeto em desenvolvimento, podendo esses testes serem efetuados junto do cliente, se o desenvolvimento foi efetuado corretamente existem poucas hipóteses de chegarem defeitos graves nesta fase, pois o código já se encontra testado à priori pelos programadores à medida que o vão escrevendo.
- **Redator técnico:** pessoa responsável por cuidar da documentação do projeto. Dessa forma, a equipa de programadores pode concentrar-se exclusivamente na implementação do código. Para que a documentação esteja de acordo com o código escrito e as necessidades do cliente, o redator deve estar sempre em comunicação com os programadores para saber o que estão a implementar e como o estão a fazer.
- **Fiscal (XP Tracker):** acompanha a conformidade das estimativas realizadas pela equipa de desenvolvimento do sistema, como estimativas de esforço, e informa o quanto precisas são as tais estimativas, com o intuito de melhorar as futuras estimativas. A atualiza periodicamente o tempo total previsto para a duração do projeto, com dados que recolhe da equipa elabora gráficos e informações que mostrem o andamento do projeto e ajudem a equipa a tomar decisões de implementação, arquitetura e *design*. Tem ainda a responsabilidade de guardar um histórico dos testes funcionais criando um registo dos problemas encontrados, assim como o número de classes métodos e linhas de código geradas. Por vezes o próprio Treinador faz papel de *Tracker*, outras vezes a equipa escolhe sozinha quem exercerá este papel.
- **Consultor:** é um elemento externo à equipa com conhecimento técnico específico necessário para o projeto em questão. O consultor também ajuda a equipa a resolver problemas específicos.

3.2.2.7.3 Funcionamento

O ciclo de vida de um projeto típico com utilização do método XP é bastante curto, conforme ilustrado na Figura 21.



Extreme Programming Project

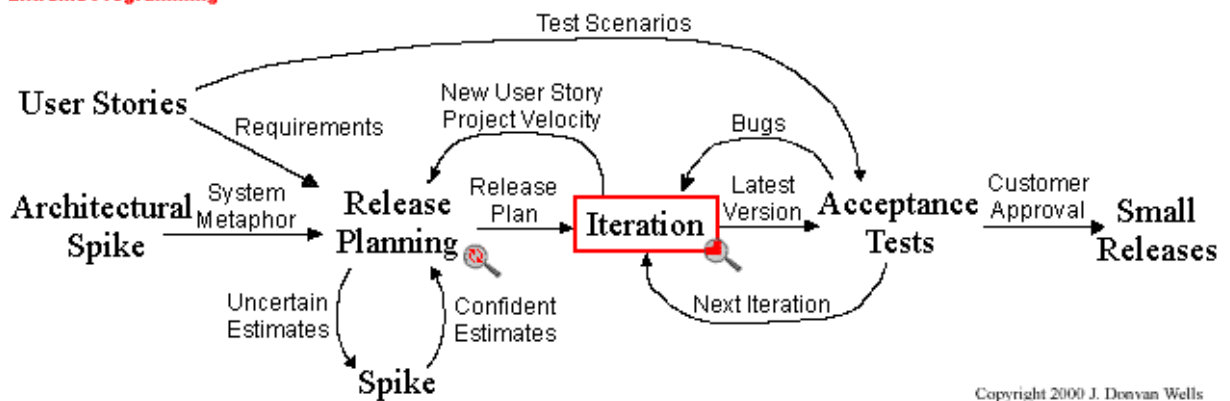


Figura 21 – ciclo de vida de um projeto típico XP (extraído de (Don Wells, 2009))

De acordo com o ciclo de vida apresentado cada iteração é decomposta em várias fases conforme ilustrado na Figura 22 e Figura 23.

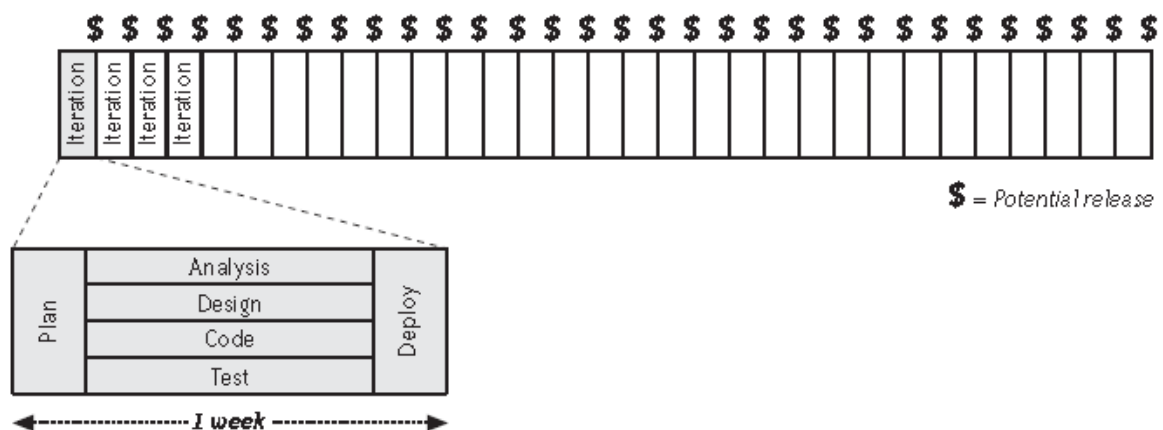


Figura 22 Fases do método XP (extraído da pág. 18 (Warden, 2007))

O XP usa iterações que duram cerca de uma semana e a cada iteração, a equipa XP faz um pouco de cada fase (planeamento, análise, conceção, construção, testes e instalação). A equipa trabalha com histórias que são partes muito pequenas de funcionalidades que dão valor para o cliente. É comum entregar-se entre quatro a dez histórias por iteração, mas isso depende de equipa para equipa. Ao término da implementação da história é feito a execução de testes internos e ao término da iteração a instalação e entrega para o cliente.



Figura 23 XP principais tarefas e periodicidade de ocorrência (extraído de (Audy, 2012))

A adoção deste método defronta-se com algumas dificuldades, como por exemplo:

- **Ultrapassar barreiras culturais**, como a partilha e permissão para que outros mexam no seu código e programar em pares;
- **Ter coragem** para admitir que não se sabe, e assim pedir ajuda e vencer hábitos antigos;
- **Manter as coisas simples** e não tentar prever o futuro ao implementar um *design* flexível;
- **Desfazer-se de código** desnecessário;
- **Definir os testes antes** da construção do código;
- **Sem medo de reestruturar** com frequência o código de modo a simplificá-lo.

O método XP não deve ser adotado quando:

- Quando o cliente não aceita as práticas do método;
- Quando o cliente quer uma especificação detalhada do sistema antes de começar a desenvolver;
- Quando os programadores não estão dispostos a seguir (todas) as regras;
- Se (quase) todos os programadores da equipa têm pouca experiência;
- Grupos grandes (mais que dez programadores);

- Quando feedback rápido não é possível, em que o sistema demora muitas horas a compilar, demora na execução de testes e exigências de certificação que possam levar meses;
- Quando o custo de mudanças é muito elevado;
- Quando não é possível a realização de testes;
- Quando equipas são grandes ou estão espalhadas geograficamente.

Ao considerar quando o método XP deve ou não ser utilizado devem ser tidas em conta as seguintes vantagens e desvantagens do método apresentadas no quadro seguinte.

Vantagens	Desvantagens
Produção rápida de protótipos	Problemas de escalabilidade
Processo iterativo	Mais focando nos resultados
Feedback frequente do cliente	Programação em pares não é rentável
Incentiva a experiência de <i>design</i>	A abordagem ao “test-drive” aumenta o tempo de desenvolvimento
Aumento da confiabilidade do sistema	Falta de requisitos específicos para garantir a satisfação do cliente
Refatoração	Não estruturado
Alta produção de código	Imprevisível
Código de qualidade	Planeamento não vigiado
O código é propriedade da equipa	Necessários programadores experientes
Ambiente de desenvolvimento coeso	Acesso a clientes comprometidos
Flexibilidade	Falta de documentação
Mais eficácia em projetos de pequeno e médio tamanho	Não é eficaz em projetos de grande escala

Quadro 3 Vantagens e desvantagens do XP

3.2.2.8 Feature Driven Development (FDD)

O método Feature Driven Development (FDD), em português Desenvolvimento Guiado por Funcionalidades, cujas orientações contribuíram também para redigir o manifesto ágil. Este método teve origem num projeto em Singapura entre 1997 e 1999 a partir do método Coad, um modelo completo para Análise, Desenho e Programação Orientada por Objetos,

desenvolvida por Peter Coad nas décadas de 1980 e 1990, e para qual contribuiu a experiência de análise e modelagem orientadas por objetos de Peter Coad, e de gestão de projetos de Jeff De Luca.

A FDD não se centraliza na programação ou na abrangência de um modelo bem definido, mas utiliza um planejamento iterativo, que tem por objetivo abstrair e satisfazer as principais necessidades do negócio, que determinará a forma de atuação da equipa de desenvolvimento. O lema do FDD é "**Resultados frequentes, tangíveis e funcionais**" (De Luca, 2002).

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web, e materiais, disponibilizados por um dos criados, Jeff De Luca, na página oficial da comunidade FDD.

3.2.2.8.1 Características

O FDD focaliza-se na entrega frequente de pequenas funcionalidades mas que funcionem e faz uso dos métodos de desenvolvimento de software iterativo e incremental.

Para ajudar, numa melhor atuação do método, o FDD defende algumas práticas, estas não são extremamente rígidas, pelo que permitem a adaptação em ambiente de desenvolvimento. No entanto, existe um conjunto de práticas que são fundamentais e que definem o FDD:

- **Modelação de dados** em objetos de domínio: consiste em construir um diagrama de classes básico com os objetos de domínio e suas relações, e definir assim uma arquitetura básica para o modelo do sistema;
- **Desenvolvimento por funcionalidades**: a implementação deve ser orientada pelas características funcionais do sistema;
- **Autoria individual**: o código é da autoria de um “dono” da classe, o que permite uma maior rapidez na implementação das tarefas associadas;
- **Funcionalidades por equipa**: para a implementação de uma determinada característica, o chefe programador recruta os “donos” das classes que serão usadas. Esse grupo de pessoas é a equipa da funcionalidade;
- **Auditorias**: inspecionar o código como forma de verificação de qualidade do código e do projeto;
- **Integrações regulares**: num determinado período de tempo fixo devem ser integradas as funcionalidades já terminadas, para permitir a verificação de erros e também criação de uma versão atual que possa ser demonstrada ao cliente;

- **Gestão de configurações:** ao existirem entregas faseadas é importante manter um registo de todas as versões do sistema;
- Acompanhar e **mostrar os resultados:** permite que se conheça o progresso do projeto.

A FDD chama a atenção por algumas características peculiares:

- Resultados úteis a cada duas semanas ou menos;
- Blocos bem pequenos de funcionalidade valorizada pelo cliente, chamados "Features";
- Planeamento detalhado e guia para medição de resultados;
- Rastreabilidade e relatórios com incrível precisão;
- Monitoria detalhada dentro do projeto, com resumos de alto nível para clientes e gestores, tudo em termos de negócio;
- Fornece uma forma de saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa são sólidos.

3.2.2.8.2 Papéis

O FDD recomenda a utilização de vários papéis. Os elementos da equipa assumem pelo menos um papel, podendo alguns elementos assumir mais que um papel. Os papéis recomendados são:

- **Arquiteto líder** (*Chief architect*): alguém com experiência em modelação de dados;
- **Programador líder** (*Chief programmer*): é o responsável por liderar, distribuir as tarefas entre os programadores de determinada equipa e participar na definição da ordem pela qual as funcionalidades serão implementadas, o próprio é também programador;
- **Especialistas do domínio** (*Domain experts*): responsável por providenciar todas as informações necessárias relativas às áreas de domínio, visão geral e modelação destes;
- **Programador** (*Developer*): coordenado pelo programador líder fica responsável por uma ou mais classes no desenvolvimento das funcionalidades;
- **Gestor de Projeto** (*Project Manager*): gere o projeto em conjunto com o gestor do desenvolvimento e o programador líder, participa no planeamento da ordem com que as funcionalidades serão implementadas;
- **Gestor do desenvolvimento** (*Development Manager*): assim como o gestor de projeto e o programador líder, participa no planeamento da ordem com que as funcionalidades serão implementadas, coordena as atividades do(s) programador(es) líder(es);

3.2.2.8.3 Funcionamento

Um projeto de desenvolvimento de software ao utilizar o FDD é composto por cinco etapas, sendo que as primeiras três apenas são executadas uma única vez e as duas últimas são executadas a cada iteração. Segundo De Luca, cada etapa possui padrões que devem ser seguidos, chamados de ETVX, oriundo de Entrada (*Entry*), Tarefa (*Task*), Verificação (*Verification*) e saída (*eXit*). Estas etapas correspondem a:

- Entrada: especifica e define os critérios de entrada para as etapas;
- Tarefa: é composto por uma lista com as tarefas que deverão ser realizadas;
- Verificação: define os tipos de avaliações (internas e externas) e inspeções de projeto e código;
- Saída: especifica os critérios de saída e define os produtos tangíveis.

Para cumprir estas etapas o FDD defende cinco processos:

1. Desenvolver um Modelo Abrangente (*Develop na Overall Model*): Análise Orientada por Objetos;
2. Construir a Lista de Funcionalidades (*Build a Feature List*): Decomposição funcional;
3. Planear por Funcionalidade (*Plan by Feature*): Planeamento incremental;
4. Detalhar por Funcionalidade (*Design by Feature*): Desenho orientado por objetos;
5. Construir por Funcionalidade (*Build by Feature*): Programação e teste orientados por objetos.

O FDD possui um ciclo de vida curto e é mais indicado para sistemas que podem mudar de requisitos rapidamente, no entanto existe a facilidade para inspeções efetivas devido às *features* serem pequenas.

3.2.2.9 Lean Software Development (LSD)

O método Lean Software Development (LSD), em português e em tradução à letra Desenvolvimento Magro de Software, mas mais conhecido com simplesmente *Lean*. Este método foi baseado no sistema de produção da Toyota e também conhecido como *Lean Manufacturing*. Surgiu no Japão, na fábrica de automóveis Toyota, logo após a Segunda Guerra Mundial, nessa altura a indústria japonesa tinha uma produtividade muito baixa e uma enorme falta de recursos, o que naturalmente a impedia adotar o modelo de produção em massa. O movimento *Lean* revolucionou a indústria e, mais recentemente, o desenvolvimento de software. O termo *Lean Software Development* teve sua origem em 2003 com a publicação do livro homónimo de Tom e Mary Poppendieck, quando apresentaram como aplicar os seus

princípios no desenvolvimento de software. Quando é devidamente aplicado, o desenvolvimento é de alta qualidade, além de ser realizado rapidamente e possuir um baixo custo. Além disso, o desenvolvimento ágil possui um grande sucesso devido ao Lean (Poppendieck & Poppendieck, 2011).

Mary e Tom Poppendieck, definiram os paralelos entre os valores e práticas *Lean* com o desenvolvimento de software, fornecendo sete princípios: “Eliminar Desperdícios, Incluir a Qualidade no Processo, Criar Conhecimento, Adiar Comprometimentos, Entregar Rápido, Respeitar as Pessoas e Otimizar o Todo”.

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web e materiais, disponibilizados por Mary e Tom Poppendieck.

3.2.2.9.1 Características

Lean é uma mentalidade em crescente utilização que pretende oferecer o que o cliente quer, onde e quando quiser sem haver qualquer desperdício. Para atingir esse objetivo, *Lean* possui alguns princípios. Nos trabalhos publicados por Mary e Tom Poppendieck, é feito o mapeamento dos sete princípios de *Lean* para o desenvolvimento de software. Em conjunto todos esses princípios oferecem entregas rápidas, com mais qualidades e com baixo custo, ao mesmo tempo. A seguir é apresentado cada um destes sete princípios:

- **Eliminar o desperdício:** ao observar o processo produtivo por inteiro;
- **Criar conhecimento:** aumentando a retroalimentação, através de ciclos rápidos e iterativos;
- **Adiar comprometimentos:** deixar as opções disponíveis pelo maior tempo possível, para que as decisões sejam feitas com o máximo de conhecimento sobre o assunto;
- **Entregar rapidamente:** as entregas devem estar alinhadas com as expectativas do cliente num determinado momento de tempo. A demora para entregar pode fazer com que o resultado não seja mais o que o cliente precise ou deseje;
- **Respeitar as pessoas:** a equipa de desenvolvimento deve ter o poder de decidir, algo necessário devido à rapidez da implementação;
- **Construa com integridade:** o software deve satisfazer o cliente – ao realizar o que ele deseja – (integridade percebida), ter um núcleo coeso (integridade conceitual) e manter-se útil com o tempo;

- **Otimizar o todo:** a visão dos implementadores deve ser do produto como um todo e não apenas de uma parte ou subsistema.

3.2.2.9.2 Papeis

O Lean recomenda a utilização dos seguintes papéis:

- **Patrocinador** (*Sponsor*): responsável por definir a visão, o rumo e as prioridades a nível financeiro além de patrocinar o projeto;
- **Proprietário do negócio** (*Business Product Owner*): define o valor do ponto de vista do negócio, atribui prioridades aos requisitos e define a constituição de cada incremento;
- **Proprietário Técnico** (*Technical Owner*): representa a integridade do sistema e evoluções do mesmo, faz as entregas do software;
- **Lean-Agile PM:** garante que o processo é entendido e seguido, facilita a eficiência da equipa e garante que o progresso é medido;
- **Toda a equipa:** produz e entrega o produto de software em desenvolvimento.

3.2.2.9.3 Funcionamento

Em relação ao processo de desenvolvimento, são definidas três etapas principais: o **início**, o estado de crescimento regular (*steady-state*), e transição e evolução. No início o foco é a análise do valor ao cliente, o estudo do negócio e projeto de viabilidade, com isso é feita uma análise de riscos e outras etapas da gestão do âmbito do projeto.

Na etapa de estado de **crescimento regular** o objetivo é realizar o desenvolvimento de forma iterativa e incremental em períodos fechados (*time-boxes*) de, no máximo, 90 dias. Cada iteração realiza um pouco das fases tradicionais de desenvolvimento (análise, projeto, implementação e testes), e integra continuamente o software. A grande diferença em relação aos outros métodos é a utilização em larga escala de *templates* (soluções de domínio) para aumentar a velocidade de produção e sua reutilização.

Na etapa de **transição** é feita a entrega, incluindo a criação de documentação e formação, mas tendo já em conta possíveis futuras evoluções do sistema.

3.2.2.10 Open Unified Process (Openup)

O Open Unified Process (Openup), em português Processo Unificado Aberto, é um processo opensource que faz parte do Eclipse Process Framework (EPF), desenvolvido dentro da Eclipse Foundation. O Openup preserva as características essenciais do modelo RUP, como a aplicação de abordagens iterativas e incrementais dentro de um ciclo de vida estruturado,

utiliza casos de uso e cenários, gestão de risco e uma abordagem centrada na arquitetura orientada ao desenvolvimento. O Openup adota uma filosofia pragmática, ágil, que se centra na natureza colaborativa do desenvolvimento de software (The Eclipse Foundation, 2012).

A seguinte descrição deste método resulta maioritariamente da consulta e análise da página web e materiais, disponibilizados pela fundação do eclipse (“The Eclipse Foundation”).

3.2.2.10.1 Características

O Openup é baseado em quatro princípios fundamentais, cada um dos quais mapeável com os valores do manifesto ágil:

- Balanceamento de prioridades concorrentes para maximizar o valor do cliente;
- Colaboração para alinhar interesses e compartilhar entendimento;
- Foco antecipado na arquitetura para minimizar os riscos e organizar o desenvolvimento;
- Envolver para continuamente obter o *feedback* e melhorar.

O Openup é organizado em duas dimensões diferentes e correlacionadas: conteúdos de métodos e conteúdo de processos. O conteúdo de método é onde os elementos do método (papéis, tarefas, artefactos e orientações) são definidos, independentemente de como eles são utilizados no ciclo de vida do projeto. O conteúdo do processo é onde os elementos do método são aplicados no sentido temporal. Muitos ciclos de vida diferentes para tipos de projetos diferentes podem ser criados do mesmo conjunto de elementos de métodos.

3.2.2.10.2 Papéis

O Openup recomenda o uso de alguns papéis para melhor definir as responsabilidades dos elementos que irão desempenhar as tarefas que por sua vez darão origem aos produtos de trabalho. Esses papéis são:

1. **Analistas:** representam os interesses do cliente e utilizador final e ajudam na definição de prioridades dos requisitos;
2. **Qualquer papel:** representa alguém que na equipa pode realizar as tarefas gerais;
3. **Arquiteto:** responsável por projetar a arquitetura do sistema;
4. **Programador:** responsável pela conceção de acordo com a arquitetura, pela construção do sistema, execução dos testes unitários e integração dos componentes que compõem o sistema;
5. **Gestor de Projeto:** lidera o planeamento do projeto em colaboração com o patrocinador e a equipa e ainda coordena as interações com o patrocinador;

6. **Patrocinador** (*Stakeholder*): representa o interesse do grupo cujas necessidades devem ser satisfeitas pelo projeto;
7. **Testador**: responsável pelas atividades de testes, tais como identificar, definir e executar os testes.

3.2.2.10.3 Funcionamento

O método Openup é representado por três camadas: micro-incrementos, ciclo de vida da iteração e ciclo de vida do projeto e participam no conteúdo e organização do trabalho a nível pessoal, de equipa e o patrocinador (Figura 24).

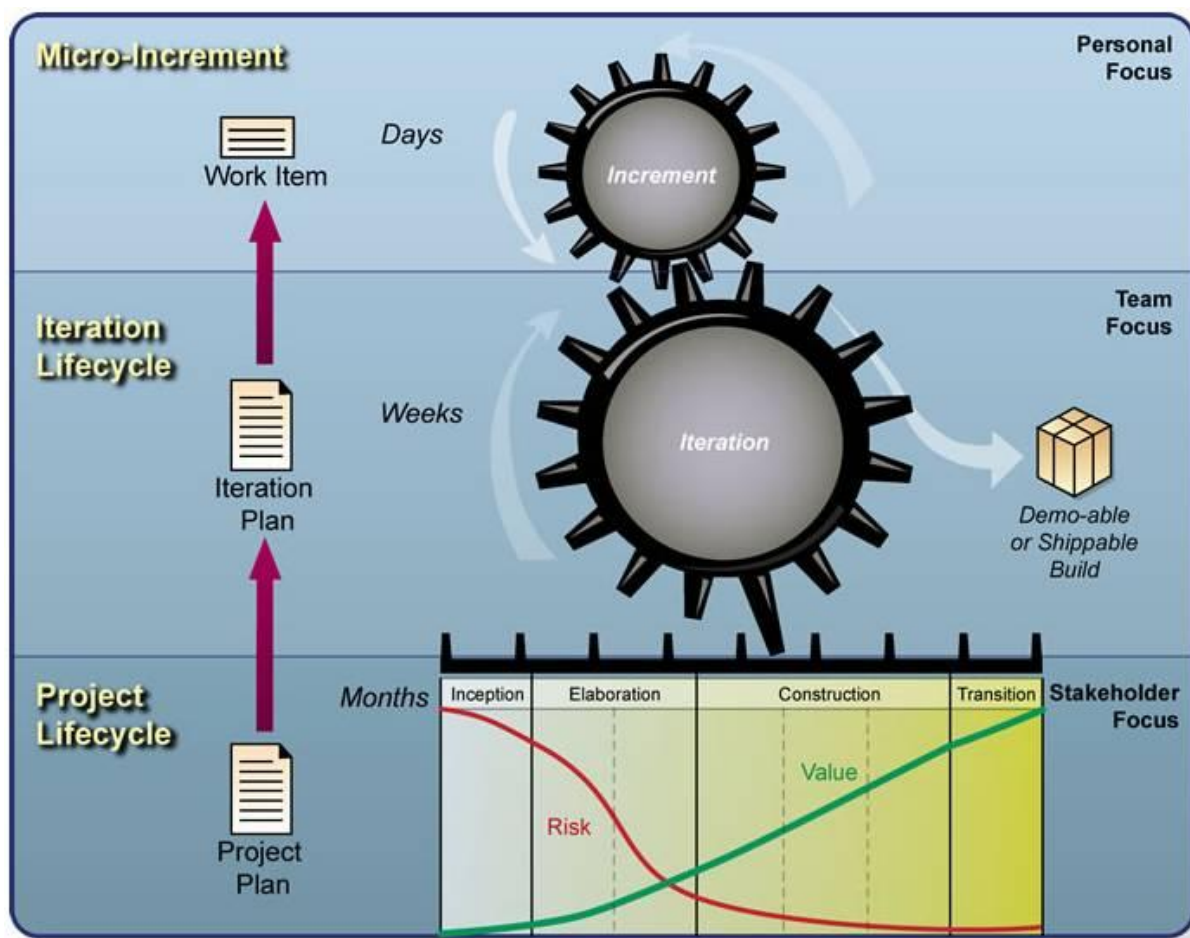


Figura 24 Organização do trabalho e foco do conteúdo no Openup
(retirado de (The Eclipse Foundation, 2012))

No nível pessoal, os membros da equipa no projeto OpenUP contribuem com os seus trabalhos em **micro-incrementos**, que tipicamente representam o resultado de algumas horas a alguns dias de trabalho. A aplicação evolui um micro-incremento por vez e o progresso é efetivamente visto diariamente. Os membros da equipa abertamente partilham o seu progresso diário nos micro-incrementos, o que aumenta a visibilidade do trabalho, a confiança e o trabalho de equipa.

O projeto é dividido em interações: intervalos planeados e com tempos definidos, tipicamente medidos em semanas. O Openup ajuda a equipa a concentrar apropriadamente os seus esforços através de **ciclos de vida de interações** a fim de entregar valor incremental ao cliente de maneira previsível – uma versão completa testada ou um incremento ao produto no final de cada interação.

O Openup estrutura o **ciclo de vida do projeto** em quatro fases: **Início, Elaboração, Construção e Transição**. O ciclo de vida do projeto permite aos *stakeholders* a fiscalização, transparência e mecanismos de direção para controlo financeiro do projeto, âmbito exposição ao risco, valor fornecido e outros aspetos do processo.

3.2.2.11 SCRUM

Inicialmente, o Scrum foi concebido como uma abordagem para desenvolvimentos de produtos comerciais que aumentaria a velocidade e a flexibilidade, baseado em estudos de casos de empresas de produção padronizada e em série nos setores de automóveis e produtos de consumo, por Takeuchi e Nonaka no artigo "The New Product Development Game", publicado em "Harvard Business Review", em Janeiro de 1986. Takeuchi e Nonaka constataram que projetos com equipas pequenas e multidisciplinares produziram melhores resultados, na época, no desenvolvimento de novos produtos em empresas no Japão e nos Estados Unidos, e associaram estas equipas altamente eficazes à jogada Scrum do *Rugby* (Sabbagh, 2010). Assim, Scrum não é um acrónimo como a designação de outros métodos ágeis, mas o seu nome deve-se à mesma designação no desporto *Rugby*, ou seja, à jogada que consiste na reunião de jogadores quando se organizam em círculos para planear a próxima jogada.

Jeff Sutherland, John Scumniotales e Jeff McKenna conceberam, documentaram e implementaram o Scrum, na empresa Easel Corporation em 1993, incorporando os estilos de gestão observados por Takeuchi e Nonaka. E em 1995, Ken Schwaber formalizou a definição de *Scrum* e ajudou a implantá-lo no desenvolvimento de software por todo o mundo (Schwaber & Armstrong, 2012).

O método Scrum junta conceitos de *Lean*, desenvolvimento iterativo e do estudo de Hirotaka Takeuchi e Ikujiro Nonaka.

A função primária do Scrum é ser utilizado para a gestão de projetos de desenvolvimento de software. No entanto, teoricamente pode ser aplicado a qualquer contexto no qual um grupo de pessoas necessitem trabalhar juntas para atingir um objetivo comum.

A seguinte descrição deste método resulta maioritariamente da consulta das páginas web, e materiais, disponibilizados pela Comunidade Portuguesa de utilizadores da framework Scrum (fundado por Tiago Andrade e Silva, Developer Evangelist na Microsoft e um dos gurus nesta área em Portugal), no site oficial do Scrum onde os autores disseminam o estado da arte e da consulta do guia de referência usado nas ações de formação do curso Certified Scrum Master (16h) monitorado em Portugal, por Alexandre Magno da empresa “ambithus”.

3.2.2.11.1 Características

Segundo os autores do método, o Scrum tem como premissa a atuação em projetos de desenvolvimento de software muito complexos e imprevisíveis, em que muito dificilmente o projeto possa ser totalmente planeado desde o início do mesmo.

O Scrum além de seguir o manifesto ágil e os seus princípios, o Scrum possui cinco valores que são muito voltados para a parte humana, são eles: o foco; a coragem; a sinceridade; o comprometimento e o respeito. Além disso o Scrum baseia-se em três pilares:

- **Transparência:** por se tratar de um processo empírico a transparência é considerada como fundamental por isso todo o processo deve estar visível a todos os envolvidos na criação do produto. Como tal, todos os elementos devem estar em sintonia com os aspetos a partilhar e assim padronizar ao máximo a informação para que todos a possam interpretar da mesma forma;
- **Inspecção:** deverão ser efetuadas várias inspeções ao longo do processo para a deteção de eventuais falhas;
- **Adaptação:** caso as inspeções detetem problemas no processo, ou alguma forma de o melhorar, então devem ser feitas adaptações. Estas adaptações devem ser feitas o mais rápido possível para garantir a produtividade da equipa e a qualidade do produto.

Através da correta inspeção e adaptação o processo é gradativamente aprimorado sempre procurando torna-lo o mais perfeito possível.

3.2.2.11.2 Papéis

O Scrum possui poucos papéis mas com responsabilidades bem definidas. Esses papéis são:

- **Product Owner (PO):** pode ser o dono do projeto ou um procurador que represente o cliente. Suas funções principais são: defender os interesses do cliente; ajudar a criar, manter e priorizar a lista de tarefas do projeto; auxiliar na definição das metas da equipa para cada *Sprint*; responder pela parte financeira do produto. Para exercer o papel do PO, são necessárias algumas características como: conhecer bem o negócio, ter capacidade de

se comunicar de forma clara e objetiva, ter tempo e disponibilidade para acompanhar o projeto e se possível, possuir alguma experiência com desenvolvimento de software.

- **Scrum Team (ST):** também designado por *Development Team* ou *Team Member*. Representa as pessoas responsáveis pela execução das tarefas do projeto de software, idealmente pretende-se que seja uma equipa com características multidisciplinares. As suas funções principais são: auxiliar nas definições de estimativas de tempo; ajudar na priorização dos itens do *Sprint*; definir os esforços de implementação dos itens do *Sprint*; determinar o objetivo do *Sprint* e desenvolver o produto propriamente dito.

- **Scrum Master (SM):** pode ser considerado o líder da ST, pois as suas funções principais são: proteger a equipa de influências externas; garantir que a equipa de desenvolvimento seja produtiva; perceber e resolver problemas internos entre a equipa; auxiliar o PO no processo de definição dos requisitos; identificar, priorizar e remover os impedimentos para a conclusão dos trabalhos; conduzir as reuniões diárias (*Daily Scrum Meetings*), de revisão de atividade (*Sprint Reviews*) e de planeamento (*Sprint Planning*); gerir e registar todo o trabalho que foi definido na reunião de *Sprint*; assegurar que o método está a ser corretamente seguido. Para a execução de todas essas funções é importante que o SM seja um profundo conhecedor do Scrum e que consiga contagiar toda a equipa com as boas práticas. Por isso, algumas das características de um bom SM são: ser responsável, humilde, colaborativo, comprometido com a equipa e o trabalho, influente e entendido.

Segundo alguns autores, o recomendado é que a equipa seja pequena e possua de cinco a nove elementos, com características multidisciplinares necessárias para entregar o projeto utilizável a cada ciclo. Outros autores diminuem esse limite e sugerem que a equipa seja composta por no máximo sete elementos. Independente do tamanho da equipa, o importante é destacar que no Scrum não existem rótulos tais como: gestor, coordenador, analista entre outros. Pois, todos os componentes são membros da mesma equipa e trabalham com um objetivo comum, cumprir as metas do *Sprint* e entregar software a funcionar e com qualidade.

3.2.2.11.3 Funcionamento

No Scrum um projeto inicia-se com a visão do produto que será desenvolvido, ao longo do seu desenvolvimento são realizados vários eventos e elaborados diversos artefactos (ver Figura 25).

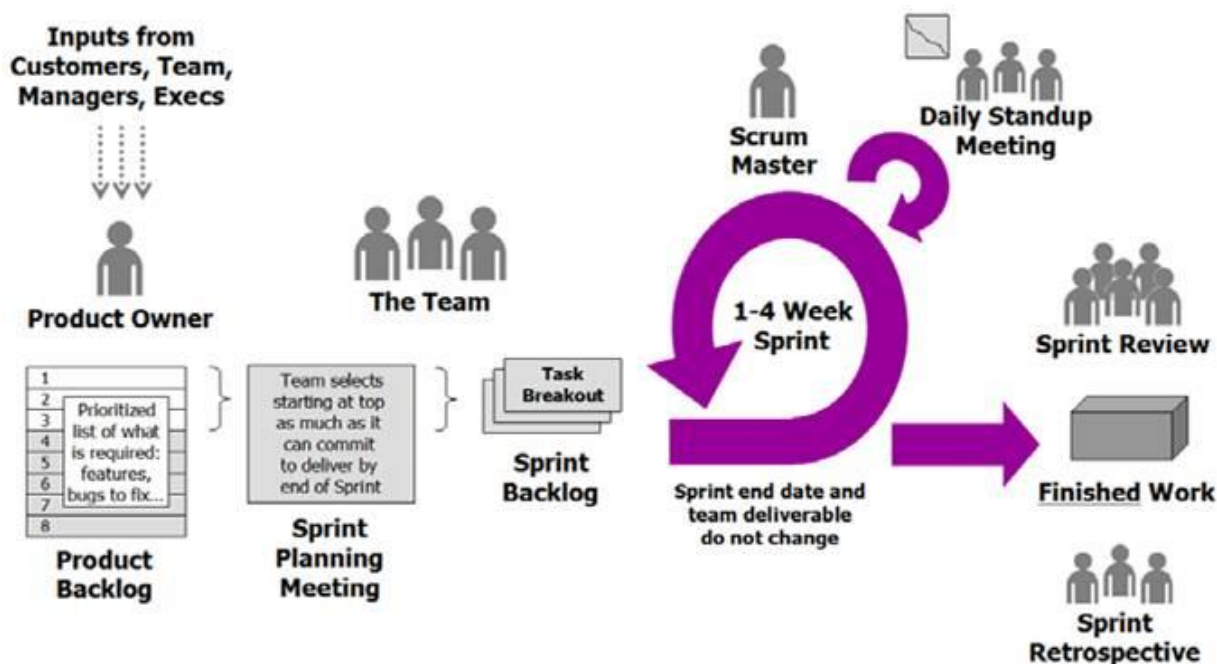


Figura 25 Ciclo de Vida de cada interação no Scrum (retirado de (Sadhan, 2010))

O Scrum possui alguns eventos de duração fixa realizados em intervalos regulares. Cada um destes eventos são oportunidades para inspeção e adaptação. E os eventos mais importantes no Scrum são:

- **Sprint:** é a designação dada a cada iteração do processo de desenvolvimento. A duração recomendada para cada Sprint é entre duas a quatro semanas o que permite *feedbacks* constantes do PO enquanto o produto está a ser desenvolvido.
- **Sprint Planning Meeting:** é a designação da reunião que é feita no início de cada Sprint, nesta reunião devem estar presentes o PO, o SM e ST e durante a mesma o PO descreve para toda a equipa as funcionalidades de maior prioridade. Esta reunião tem uma duração máxima de oito horas no caso de um Sprint de um mês, o tempo é reduzido de forma proporcional.
- **Sprint Planning Meeting – 1ª parte:** na primeira parte da Sprint Planning Meeting em que o ST faz uma previsão das funcionalidades que serão desenvolvidas durante o Sprint. O PO apresenta os itens do topo do Product Backlog ao ST e toda a equipa colabora na definição dos itens. Para a seleção dos itens a constar no Sprint são tidos em conta os incrementos mais recentes, a capacidade projetada do ST e a performance em Sprints passadas.
- **Sprint Planning Meeting – 2ª parte:** na segunda da Sprint Planning Meeting o ST decidirá como transformar os itens selecionados num incremento durante o Sprint. o ST é

auto-organizado e coordena-se para decompor os itens em unidades de um dia ou menos, o suficiente para os primeiros dias do Sprint. Os itens selecionados e o plano para o desenvolvimento do incremento dão origem ao Sprint Backlog.

• **Daily Scrum:** diariamente o ST reúne-se para uma reunião de, no máximo, quinze minutos, onde cada membro por sua vez responde objetivamente para os outros membros às seguintes questões:

- O que fiz desde o último Daily Scrum?
- O pretendo fazer até ao próximo Daily Scrum?
- Existe algo a impedir-me de concluir alguma tarefa?

Como o ST trabalha de uma forma colaborativa permite que os membros da equipa comuniquem e sincronizem o seu trabalho. E, por serem autogeridos não é necessário fazerem qualquer relatório da reunião para PO e SM, apenas para os próprios. Caso exista algum impedimento em primeiro o ST tenta resolvê-lo, e caso não o possa resolver então endereça o impedimento em causa para o SM para que ele o resolva. Estas reuniões diárias permitem analisar melhor o progresso e corrigir de imediato algum problema que possa impactar do desenvolver dos trabalhos do Sprint.

• **Sprint Review:** ocorre no final de cada Sprint, e em que podem participar quem estiver interessado no produto. Esta reunião é utilizada para demonstrar as novas funcionalidades feitas durante o Sprint e o objetivo é inspecionar o que foi produzido e recolher opiniões para, caso necessário, adaptar o plano para o Sprint seguinte, portanto pretende-se aprimorar o produto.

• **Sprint Retrospective:** é o último evento de um Sprint e ocorre imediatamente após o Sprint Review. Participam nesta reunião todos os elementos do ST e o foco é o aprimoramento do processo – a interação entre os membros de equipa, as práticas, as ferramentas utilizadas, o que funcionou e o que precisa ser melhorado na próximo Sprint. Além da identificação dos problemas pretende-se também identificar as medidas a tomar para a melhoria do processo a aplicar já nos próximos Sprints.

• **Release Planning Meeting:** no final de cada Sprint o ST reentrega um incremento do produto, que é algo potencialmente “entregável”. Apesar de poder entrar logo em produção existem situações em que isso é postergado para que seja feito somente na *Release* do produto. Uma *Release* é a representação de uma entrega do produto em produção. Uma *Release* é normalmente planeada através do evento designado *Release Planning Meeting*. As *Releases* são planeadas principalmente em ambientes onde as

entregas precisam de uma data fixa, seja por obrigações contratuais, necessidades de mercado, eventos ou para cumprir com o *roadmap* de um produto.

O Scrum possui alguns artefactos que dão uma visão do andamento do projeto e dos Sprints. Esses artefactos, embora utilizados ao longo do projeto, possuem momentos de criação diferentes. Os artefactos criados e usados no Scrum são:

- **Product Backlog:** é uma lista ordenada criado pelo ST mas onde apenas o PO pode inserir, remover ou reordenar os itens. O formato mais usual para estes itens é sob a forma de *User Stories*, estes deverão ser ordenados de acordo com o critério do PO, em que geralmente os itens mais importantes ficam no topo e serão os primeiros a implementar. Além dos itens funcionais o Backlog contém também itens não funcionais, arquiteturais e de infraestrutura. Podendo ainda conter itens que representem riscos a serem removidos.

- **Sprint Backlog:** é o conjunto de itens selecionados para serem implementados durante o Sprint mais o plano para transformá-lo num incremento. Assim, no final de cada Sprint planning Meeting, um novo Sprint Backlog é criado. Normalmente o plano é composto pelas tarefas técnicas necessárias para transformar o item num incremento do produto. Um dos formatos mais comuns de Sprint Backlog é o Task Board, ou seja, um quadro normalmente com informação em *post-its* e com quatro colunas: referência do item do Product Backlog; “To do”, “Doing” e “Done”, portanto as tarefas a fazer, em curso e prontas respetivamente. O objetivo é tornar visível para toda a equipa o progresso de cada Sprint, com a informação atualizada diariamente, e assim os membros da equipa podem adicionar ou remover tarefas no decorrer do Sprint.

- **Product Increment:** no final de cada Sprint o ST entrega ao PO um incremento do produto, resultante da sua produção durante o Sprint. Isto permite que o PO perceba o valor do investimento e também que vislumbre outras possibilidades para o produto final.

- **Definition of Done:** o Scrum não define quaisquer práticas de engenharia de software para a ST. No entanto recomenda que se utilizem práticas ágeis para este fim e que permitam ao ST garantir a qualidade do que se esteja a produzir durante o Sprint. Uma funcionalidade só é considerada pronta ou terminada quando passar todas as fases definidas pelo ST, caso uma funcionalidade não tenha completado as fases necessárias então deve retornar para o PO para que seja incluída no Sprint seguinte.

• **Burndown Charts:** no Scrum o ST é auto-organizado, e para ajudar nesta autogestão o ST pode fazer uso do Sprint Burndown Chart, este gráfico torna visível diariamente o progresso do trabalho realizado comparativamente com o trabalho estimado inicialmente no Sprint Planning Meeting com a quantidade restante estimada em horas de trabalho. Enquanto o Burndown Chart permite ao ST fazer um melhor acompanhamento do evoluir dos trabalhos, o PO pode fazer uso de um outro gráfico similar para gerir as *Releases*, o Release Burndown Chart.

3.2.3 Fatores Críticos de Sucesso para o Desenvolvimento Ágil

O mercado está cada vez mais recetivo à utilização das abordagens ágeis e como tal são produzidos cada vez mais materiais para facilitar a sua adoção.

No entanto, a utilização efetiva da gestão ágil no processo de desenvolvimento de software não pode ser encarada como trivial. Ao analisarem-se as descrições anteriores dos métodos considerados ágeis pode deduzir-se que a sua implementação não está pendente apenas da adoção das novas práticas mas entre os pré-requisitos mais importantes está a **mudança cultural da organização**. Esta mudança cultural, para adotar a agilidade, implica a privação do gestor de projetos na tomada de decisões pela equipa e do lado dos elementos da equipa pressupõe o assumir das responsabilidades pelas decisões tomadas o que é imprescindível além do maior comprometimento com a equipa e o projeto. A estrutura organizacional passa a ser determinante e estruturas mecanicistas tornam-se inadequadas.

Outro fator importante a ter em conta é que o cliente passa a integrar a equipa, portanto deixam de fazer sentido os contratos fechados para cada projeto, pois o cliente não vai considerar o âmbito, custos e prazos fechados mas terá que ser agilizado ao longo do projeto. Para tal é recomendada uma abordagem evolutiva iterativa do projeto com a participação do cliente em detrimento das longas fases baseadas em especificações exaustivamente detalhadas (como na abordagem tradicional).

3.3 Sistemas Integrado de orientação do PDS

Existem algumas soluções no mercado, na maioria proprietárias de empresas de TI, que disponibilizam ferramentas e orientações para a sua utilização e também serviços de consultoria para a sua melhor adoção, como por exemplo a Microsoft com a “Microsoft Solutions Framework”.

3.3.1 Microsoft Solutions Framework (MSF)

A Microsoft Solutions Framework (MSF) surgiu em 1994 como um conjunto de boas práticas recolhidas pela Microsoft a partir de sua experiência na produção de software e serviços de consultoria. Desde então, o MSF evoluiu, tornando-se uma *framework* flexível para guiar o desenvolvimento de projetos de software. Como principais características, temos o estabelecimento de papéis bem definidos, a definição e implantação das boas práticas em fluxos de trabalho e atividades.

A presente descrição do MSF resulta maioritariamente da consulta e análise da página web e materiais, disponibilizados pela Microsoft.

A Microsoft não classifica o MSF como uma metodologia ou um método, mas sim como uma disciplina. O que significa que, basicamente o MSF serve como um grande guia e uma coleção de boas práticas, porém o MSF não se aprofunda em detalhes. Por exemplo, num dado momento do projeto, o MSF refere a necessidade de fazer uma especificação funcional. No entanto, não define a utilização de UML, análise essencial ou outras técnicas, pois isso fica ao critério do utilizador.

A falta de detalhes do MSF pode parecer uma deficiência a princípio, mas essa característica permitiu uma abordagem simples e direta das técnicas apresentadas. Ou seja, o MSF permite uma fácil compreensão tanto por parte da equipa de desenvolvimento como do cliente, além de ser bastante flexível na sua aplicação.

O MSF conta atualmente com a versão 4.0, publicada em 2005, é uma combinação de um meta-modelo, que pode ser usado como base para processos de engenharia de software prescritivo, e depois personalizável e escalável a processos de engenharia de software. O MSF 4.0 fornece uma ferramenta de alto nível como *framework* de orientações e princípios que

podem ser mapeados para uma variedade de modelos de processo prescritivos. O programa está estruturado em dois métodos descritivos e prescritivos.

A Microsoft entende que as organizações têm diferentes dinâmicas e contrariam as suas prioridades durante o desenvolvimento de software, assim algumas organizações necessitam de processos e sistemas adaptáveis ao ambiente de desenvolvimento, enquanto outros precisam de algo padronizado, repetitivo e de um ambiente controlado. Para atender estas necessidades, a *Microsoft* representa o meta-modelo do MSF 4.0 em dois modelos, um chamado ***Microsoft Solutions Framework para Agile Software Development*** (MSF4ASD) e o outro ***Microsoft Solutions Framework para Capability Maturity Model Integration***, ou seja, para melhoria de processos (MSF4CMMI). Note-se que estes processos de engenharia de software podem ser modificados e personalizados de acordo com as preferências da organização, clientes e equipa do projeto.

O MSF4ASD usa os princípios do desenvolvimento ágil e prevê um processo de orientação que incide sobre as pessoas e as mudanças. Inclui aprendizagem de oportunidades e de avaliações ao usar iterações em cada iteração.

O MSF4CMMI dispõe de mais artefactos, mais processos, mais aprovações, mais planeamento e destina-se a projetos que requerem um maior grau de formalidade, o MSF4CMMI é portanto um método formal para a engenharia de software.

O MSF4CMMI tem mais documentos obrigatórios e relatórios do que a versão ágil, mais formal e desenvolvimento deste processo reduz o risco em grandes projetos de software e fornece um estatuto mais mensurável. Uma das vantagens de utilizar o CMMI é o processo pelo qual a utilização de uma avaliação padrão poderá comparar a capacidade de desenvolver software comparando com outras organizações.

A filosofia MSF afirma que não existe uma única estrutura ou processo que se aplica otimamente aos requisitos e ambientes para todos os tipos de projetos. Portanto, o MSF suporta múltiplas abordagens ao processo, para que possa ser adaptado para apoiar qualquer projeto, independentemente da sua dimensão ou complexidade. Esta flexibilidade significa que o MSF pode suportar uma ampla variação no grau de execução dos processos de engenharia de software e manter porém um conjunto de princípios fundamentais e mentalidades.

O MSF consiste na execução de séries curtas de ciclos de desenvolvimento e iterações. Este modelo abraça o desenvolvimento rápido e iterativo com a aprendizagem contínua e o refinamento, devido ao progressivo conhecimento da empresa e do projeto de todos os agentes envolvidos.

Seguem-se os oito princípios fundamentais, que constituem a estrutura principal para os outros modelos e disciplinas do MSF:

- **Comunicação aberta** - o MSF aconselha a comunicação aberta entre toda a equipa, clientes e outros envolvidos na equipa;
- **Visão partilhada do projeto** - o processo de partilha da visão do projeto é definida logo no início do projeto. Com a criação desta visão a equipa comunica com o intuito de identificar e resolver conflitos e falsas visões, o que permite definir a direção do projeto;
- **Mais autoridade aos membros da equipa** - o MSF delega autoridade (*empowerment*) aos elementos da equipa para atingirem as metas e entregas, essa aceitação da autoridade permite a partilha de responsabilidades na tomada de decisões e direções quando necessário;
- **Estabelecer responsabilidades claras e partilhadas** - toda a equipa pode partilhar várias responsabilidades, as quais devem ser adequadas aos elementos da equipa e permitir o claro relacionamento com os respetivos *stakeholders*;
- **Foco no negócio e prestação valor** - entender porque o projeto existe da perspetiva do negócio e como este é medido;
- **Agilidade** - as iterações do ciclo de vida promovem mudanças no projeto, o que origina pedidos de alterações que são incluídos no projeto mesmo tendo que fazer ajustes nas metas e entregas do projeto;
- **Investimento na qualidade** - o MSF tem como premissa que toda a equipa é responsável por ponderar os custos e funcionalidades para preservar a solução em qualidade e assegurar a qualidade do produto. Os elementos da equipa devem obedecer aos critérios de qualidade em todas as fases do projeto, e por sua vez a organização deve investir na sua equipa, em competências, formação e experiência.
- **Aprender com a experiência**: o crescimento contínuo é baseado na experiência individual e da equipa. Em particular com as más experiências, como fracassos de projetos, deve aprender-se de forma a não repetir os mesmos erros que os cometidos em projetos anteriores.

O MSF consiste em dois modelos: o modelo de equipa e o modelo de governação (ou de processo). O modelo de equipa prevê a escalabilidade do projeto, identifica quem vai trabalhar no projeto e interliga cada equipa com um responsável do modelo de processo que aprovisiona a alta qualidade através do ciclo de vida do projeto. O modelo de processo trabalha em conjunto com o modelo de equipa ao organizar o projeto em fases distintas.

O **MSF modelo de equipa** define uma estrutura de papéis e as suas responsabilidades num projeto. O MSF define seis papéis centrais numa estrutura não hierárquica (conforme Figura 26). Os papéis podem ser assumidos pelos vários membros da equipa de projeto e de desenvolvimento de software.



Figura 26 Modelo de equipa com os papéis mais importantes (extraído de (Microsoft, 2006))

Os papéis a desempenhar pelos membros dessa equipa são:

- **Gestor de Produto** (*Product Management*): lida principalmente com os clientes na definição dos requisitos do projeto e garante que as expectativas do cliente são cumpridas;
- **Gestor de Projeto** (*Program Management*): mantém o desenvolvimento do projeto e a entrega ao cliente;
- **Programador** (*Development*): desenvolve a solução de acordo com as especificações;
- **Testador** (*Test*): valida e garante a qualidade dos produtos;
- **Instalador/ Operador** (*Release Management*): garante a instalação e o bom funcionamento do software;
- **Apoio ao Utilizador** (*User experience*): apoia o utilizador em questões funcionais e de usabilidade do sistema.

Um membro da equipa pode assumir vários papéis, o MSF tem sugestões de combinações de papéis, mas no entanto sugere que o programador não assuma qualquer outro papel.

O **MSF modelo de governação ou de processo** descreve a sequência de alto nível das atividades para a construção e implantação de soluções de TI. Em vez de prescrever uma série específica de procedimentos, é flexível o suficiente para acomodar uma ampla gama de projetos de TI. O MSF combina dois modelos de processo de desenvolvimento de software: o modelo em cascata, que enfatiza o cumprimento de metas, e o modelo espiral, que incide sobre a contínua necessidade de refinar os requisitos e as estimativas para um projeto. Um aspecto inovador do modelo de processo MSF é que cobre o ciclo de vida de uma solução desde o início do projeto à operacionalização do mesmo. Isso ajuda as equipas de projeto a se focarem no valor de negócio para o cliente, porque nenhum valor é percebido até que a solução esteja instalada e em operação.

Este modelo assume cinco faixas com sobreposição de atividade (conforme Figura 27), cada uma das fases com uma meta de qualidade definida e atividades a cumprir nessa fase também definidas. As faixas podem ser pequenas no âmbito mas realizadas rapidamente para serem coerentes com o método ágil ou pode ser serializada e alongada para ser coerente com o método de cascata.



Figura 27 Modelo de processo do MSF com as fases e marcos importantes (extraído de (Microsoft, 2006))

As cinco fases do modelo de processo são:

- **Visão** (*Envision*): define o âmbito do projeto, ou seja, o que vai ser realizado e identifica os constrangimentos;
- **Plano** (*Plan*): planeia e projeta uma solução para atender às necessidades e expectativas do cliente de acordo com as limitações identificadas;
- **Construção** (*Build*): a construção propriamente dita da solução;

- **Validação** (*Stabilize*) – validação da solução para garantir que está de acordo com as necessidades e expectativas identificadas;
- **Instalação** (*Deploy*) – implantação da solução.

O MSF faz uso de três disciplinas clássicas, Gestão de Riscos, Gestão de Disponibilidade e Gestão de Projetos, que adaptou para se adequarem a esta *framework*. Estes conceitos estão refletidos tanto no modelo de processo e as responsabilidades definidas nos papéis do modelo de equipa. Tanto

Tanto a estrutura de equipas no MSF, como o modelo de processo e as suas disciplinas são descritos nos documentos oficiais da Microsoft sempre como um conjunto de boas práticas, guias de orientação e aplicação e não como um processo prescritivo e dogmático de software.

3.4 Desenvolvimento Tradicional *versus* Desenvolvimento Ágil

Os métodos ágeis para o desenvolvimento de software estão a ganhar cada vez mais adeptos nas organizações de TI, mas nem sempre são bem aceites ou entendidos pelos executivos das organizações, que estão acostumados com a gestão de projetos de forma tradicional, nomeadamente a utilização do modelo de cascata (Sousa, 2011).

Muitas vezes são discutidas as vantagens e desvantagens de cada uma das abordagens mas nem sempre por conhecimento de causa, mas apenas por ideias formadas do que são sem que sejam profundamente estudadas.

A maioria dos métodos ágeis nada possui de novo em relação à abordagem tradicional, mas o que as difere são os princípios e valores. De realçar que o foco na abordagem ágil são as pessoas e não os processos, como é defendido nos métodos tradicionais, e despende-se assim menos tempo com documentação e mais tempo com implementação. Dessa forma, os métodos ágeis são adaptativos, em vez de serem preditivos. Com isso, adaptam-se a novos fatores decorrentes do desenvolvimento do projeto, ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento.

Portanto, a diferença primordial está em evitar os problemas já conhecidos de tentar estimar e fazer o planeamento total do software em detalhes, atividades que comprovadamente falham (ver Figura 28). Para contornar isso, a abordagem ágil permite o envolvimento ao longo do projeto do cliente e da equipa nas tomadas de decisão o que pressupõe maior interesse e motivação da equipa por se sentir parte do projeto, ao longo de todas as etapas do decorrer do mesmo.

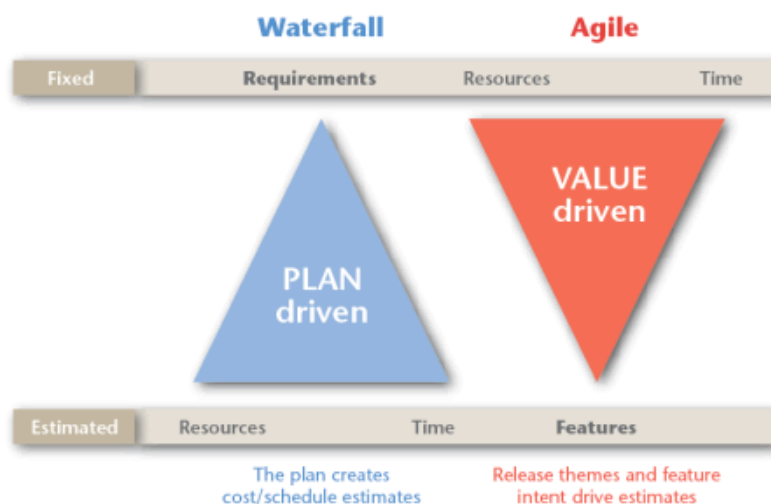


Figura 28 Abordagem tradicional e ágil (extraído de (Sousa, 2011))

De acordo com os factos descritos neste trabalho são de destacar as seguintes grandes vantagens entre as abordagens tradicionais e ágeis:

- Com o detalhe pormenorizado das funcionalidades que o sistema vai ter, na abordagem tradicional, assim será possível fazer uma **melhor estimativa dos recursos e custos** do projeto logo no início do mesmo;
- A especificação dos requisitos do sistema ao terem que ser definidos no início do projeto, na abordagem tradicional, não permitirá adaptação através de alterações que possam ser identificadas ao longo do projeto, ao contrário da abordagem ágil que prevê entregas periódicas e assim o cliente pode introduzir **mudanças ao longo do projeto** permitindo uma maior flexibilidade e agilidade;
- A **vantagem competitiva** que se possa ter com o lançamento em primeiro no mercado e a obtenção da liderança no mercado será muito mais provável com a utilização da abordagem ágil, pois conseguem-se resultados muito mais rápidos. O mesmo se aplica para alterações e correções nos produtos para que possam ser disponibilizados em períodos menores que os da abordagem tradicional;
- **Melhoria da qualidade** do produto na abordagem ágil, quando a fase de testes está presente durante todo o desenvolvimento do produto, o que permite ao utilizador fazer ajustes, e se necessário, fornecer à equipa de desenvolvimento uma visão sobre os problemas a ultrapassar ainda relativamente cedo e sem comprometer o produto final;
- A abordagem ágil recomenda que o utilizador deve ter um papel ativo no decorrer do projeto, o que permite ao **cliente acompanhar o progresso** do projeto e certificar-se que

este não fica abaixo das suas expectativas e assim aumenta a satisfação do cliente com o projeto;

- Como a abordagem ágil as entregas periódicas de pequenos incrementos possibilitam um melhor **controle dos riscos** do projeto, através da sua mitigação periódica ao longo do projeto associado a tomadas de decisão mais corretas e o mais cedo possível;
- Na abordagem ágil consegue-se um melhor **controle dos custos** ao possibilitar que se disponha de um orçamento fixo e são as características do produto e a duração da fase de desenvolvimento que aumentam se necessário;
- Com base nos pontos anteriores pode dizer-se que a ênfase do desenvolvimento ágil está em criar **o produto certo**, ao passo que o desenvolvimento tradicional se preocupa mais em completar o projeto com sucesso, mesmo que mais tarde venham a perceber que o produto não é bem aquilo que se esperava.

Da mesma forma, é de destacar, como base no estudo apresentado no presente trabalho, as grandes desvantagens que diferenciam as abordagens tradicionais das ágeis:

- A abordagem ágil requer um grande envolvimento dos utilizadores durante o desenvolvimento do produto, apesar da vantagem na participação ativa ao longo do projeto, pressupõe também grande **responsabilidade e compromisso** o que pode exigir muito tempo. De destacar também o tempo gasto com os testes de aceitação, em que entregas frequentes pressupõem maior dedicação e tempo para execução de todas as validações necessárias frequentemente;
- Com as constantes mudanças no decorrer do projeto, na abordagem ágil, todas as mudanças podem ser adotadas, o que pode provocar o **adiamento constante do fim** do projeto. Para tal deve existir maturidade para definir um preço fixo para o projeto para fazer-se os reajustes necessários no limite do projeto;
- A abordagem ágil pressupõe a diminuição de documentos escritos, pelo que pode ocorrer existir **pouca informação disponível** para novos elementos da equipa, o que dificulta a integração no projeto, podendo criar desentendimentos no grupo e dificuldades para toda a equipa que se podem traduzir no não cumprimento dos objetivos em cada ciclo de desenvolvimento;
- Ao incluir a fase de testes ao longo de todo o desenvolvimento na abordagem ágil, apesar de possibilitar a qualidade do produto provoca também um **aumento de custos** do projeto;

- Na abordagem ágil o facto do incentivo ao constantemente envio de feedback pelo cliente faz do desenvolvimento algo bastante intenso para quem nele participa, conjuntamente com o ter de implementar a totalidade dos componentes em cada iteração, e ainda a quantidade de iterações, pode ser **mentalmente cansativo** e tornar difícil para a equipa de desenvolvimento encontrar equilíbrio.

A análise de vantagens e desvantagens apresentada traduz de uma forma geral a comparação entre as duas abordagens ágeis e tradicionais, poder-se-ia ir mais além através da análise detalhada de cada um dos métodos apresentados, nomeadamente os métodos ágeis, no entanto a revisão bibliográfica revelou que não existem os mesmos critérios a caracterizar cada um dos métodos não possibilitando assim uma correta e completa comparação entre os métodos.

4. Análise dos Métodos e de Benchmarking

Depois de apresentados os vários métodos de PDS apresenta-se agora uma análise dos métodos ágeis. Faz-se também uma análise de Benchmarking tendo em conta um trabalho importante na área.

4.1 Análise dos Métodos Ágeis

Esta secção pretende complementar o estudo dos métodos, ou seja, surge como resultado da secção 3.2.2, na qual são apresentados e descritos os métodos ágeis: ASD, AUP, Crystal, DSDM, YP, ESSUP, XP, FDD, LSD, Openup e Scrum.

Todos os métodos têm as suas particularidades o que permite a sua diferenciação e são as suas características próprias que facilitam o processo de tomada de decisão para a escolha do melhor método a usar para um projeto.

Muitos dos métodos não possuem uma estrutura idêntica, nem tão pouco um comportamento similar no decorrer de cada projeto, pelo que se evidencia a necessidade de elaborar uma matriz comparativa. Portanto, para uma melhor análise dos métodos ágeis apresentados anteriormente é, de seguida, produzida uma análise comparativa a partir de interpretação de dois quadros elaborados tendo em conta alguns critérios e os respetivos comportamentos dos métodos ágeis enumerados em epígrafe.

A análise comparativa é feita em função de vários critérios, nomeadamente:

- **Planeamento:** tendo em conta a existência desta atividade, se esta atividade implica a criação de um plano para todo o projeto ou se o planeamento é realizado repetidamente no decorrer do projeto e também quando esta atividade ocorre;
- **Recolha dos requisitos:** qual a atividade em que esta tarefa ocorre e se é dado destaque a esta tarefa;
- **Análise dos requisitos:** entende-se como a definição propriamente dita dos requisitos e se são usadas técnicas para facilitar a análise dos requisitos do sistema;
- **Regras:** entendidas como características particulares de cada método;
- **Equipa:** características particulares a ter em conta na constituição de cada equipa do projeto;
- **Papéis:** recomendação de papéis a serem assumidos pelos elementos da equipa do projeto;
- **Codificação:** quando ocorre a tarefa de implementação do código do sistema,
- **Tipos de estimativas:** tendo em conta a definição de requisitos como é feita a previsão de tempo/ esforço de cada requisito a implementar;
- **Reuniões:** tipo de cerimónias previstas para ocorrer ao longo do projeto;
- **Monitoria:** na existência de monitoria quais os artefactos recomendados para a execução do controlo e acompanhamento do projeto;
- **Testes:** tipo de testes recomendados para execução em cada projeto;
- **Versionamento:** a ter e conta o versionamento dos produtos ou sistemas a entregar no decorrer de cada projeto;
- **Duração de cada ciclo:** assumindo que a abordagem pressupõe a produção de incrementos ao longo do projeto então qual a duração estimada para cada ciclo;

Critérios de análise	ASD	AUP	Crystal	DSDM	YP	ESSUP *
Planeamento	Fase de especulação subfases: Início do Projeto e Planeamento do Ciclo Adaptável	Fase de conceção	Fase de estágio	Estudo do objetivo do negócio	Fase de planeamento (da release e da iteração)	-
Recolha de requisitos	Definidos em JAD	Fase de conceção	Fase de estágio	Fase de pré-projeto	Conversa com cliente resulta documento de visão	-
Análise de requisitos	Definidos e priorizados em JAD - declaração de missão	Requisitos analisados na Fase de Elaboração	Documento de visão na fase de estágio	<i>User stories</i> na fase de projeto	<i>User stories</i> na fase de inicialização	Casos de uso
Regras	-	Evita documentação extensa, dois ambientes (Estágio e Produção)	Iterações fixas	Pareto (80%/20%); Mudanças reversíveis	-	-
Equipa	Multidisciplinar	-	Várias equipas a trabalhar em paralelo	Equipas pequenas	-	-
Papéis	Patrocinador; facilitador; escritor; programador; gestor do projeto e cliente	-	Patrocinador, utilizador, programador/analista e programador/analista sénior	Gestor executivo, visionário, gestor do projeto, coordenador técnico, líder de equipa, programador, testador, facilitador e outros papéis específicos	Cliente, utilizador, gestor de projeto, programador, testador	-
Codificação	Fase de Colaboração, ASD orientado a componentes e revisões técnicas	Fase de implementação	-	-	Fase de implementação	-
Estimativas	Estimativa por missão	Estimativa alto nível com base da modelação	Estimativa por funcionalidade	Estimativa por funcionalidade	Estimativas por <i>user story</i>	-
Reuniões	Análise em grupo com foco no cliente (JAD)	-	<i>Workshops</i> de análise	Revisão do negócio e <i>workshops</i>	Reuniões de acompanhamento (face a face)	-
Monitoria	Relatórios de progresso	Relatórios de progresso	Relatórios de progresso	Relatórios de progresso	Relatórios de progresso	-
Testes	Testes integrados	Testes unitários e testes de aceitação	Desenvolvimento por casos de teste	Testes integrados	Testes unitários, de integração, usabilidade e aceitação	-
Versionamento	Frequente	Frequente	Frequente	Frequente	Frequente	-
Duração de cada ciclo	4-8 Semanas	-	1-4 Meses	-	1-4 Meses	-

*pouca informação pública que permita tirar conclusões

Quadro 4 Análise comparativa dos métodos ágeis

Critérios de análise	XP	FDD	LSD	Openup	SCRUM
Planeamento	<i>Release planning e Iteration plan</i>	Processo de desenvolvimento do modelo abrangente	Etapa início	Fase início do ciclo de vida do projeto e plano de cada iteração	<i>Sprint Planning</i>
Recolha de requisitos	No planeamento de cada release	Requisitos são listados, definidos e documentados através de casos de uso; lista de funcionalidades	Etapa início	Fase início do ciclo de vida do projeto	<i>Product Backlog</i>
Análise de requisitos	Cartões de classes e funcionalidades; <i>User stories</i>	Requisitos agrupados e priorizados conforme importância e dependência	Etapa crescimento regular	Plano de cada iteração	<i>Sprint Backlog User stories</i>
Regras	Valores fundamentais: comunicação, simplicidade, feedback e coragem E princípios focados no processo de codificação	Desenvolvimento por funcionalidade	Eliminar Desperdícios, Incluir a Qualidade no Processo, Criar Conhecimento, Adiar Comprometimentos, Entregar Rápido, Respeitar as Pessoas e Otimizar o Todo	3 Camadas: micro-incremento, ciclo de vida da iteração e ciclo de vida do projeto	
Equipa	Duplas	Equipas de funcionalidades		-	Multidisciplinar
Papéis	Cliente, gestor do projeto, treinador, programador, analista de testes, testador, redator técnico, fiscal e consultor	Arquiteto líder, programador líder, especialistas do domínio, programador, gestor do projeto, gestor do desenvolvimento	Patrocinador, proprietário do negócio, proprietário técnico, Lean-Agile PM, restante equipa	Analista, arquiteto, programador, gestor de projeto, patrocinador e testador	Product Owner, Scrum Team e Scrum Master
Codificação	Refatoração; integração continua; programação em pares; propriedade coletiva	Propriedade do código individual; Inspeção de código	Etapa crescimento regular	Em cada incremento	<i>Sprint</i>
Estimativas	Jogos de planeamento	Estimativa por funcionalidade	Cartão tema (esforço por funcionalidade) para cada iteração	-	<i>Sprint planning; planning poker</i>
Reuniões	<i>Stand Up Meeting</i>	Desenvolvimento do modelo abrangente	Reuniões diárias	Reuniões diárias	Reuniões diárias; <i>sprint review</i>
Monitoria	Relatórios de progresso	Relatórios de progresso	Relatórios de progresso e gráficos de progresso	Relatórios de progresso	<i>Burndown chart</i>
Testes	Testes primários; Triagem de erros	Testes integrados	Testes unitários e de integração	Testes unitários e de integração	-
Versionamento	Frequente	Frequente	Frequente	Frequente	Frequente
Duração de cada ciclo	1-4 Semanas	1-2 Semanas	1-3 Meses	1-4 Semanas	2-4 Semanas

Quadro 5 Continuação da análise comparativa dos métodos ágeis

Os quadros apresentados permitem a extração de algumas constatações relativas ao comportamento dos métodos ágeis, seguidamente são expostas algumas afirmações com base no estudo dos métodos e nos quadros apresentados anteriormente. No entanto, fica de fora desta análise o método ESSUP por não dispor publicamente de informação suficiente que permita comparação com os restantes métodos.

- Todos os métodos preveem planeamento do projeto podendo não ser um planeamento do projeto como um todo, o planeamento total tem maior notoriedade na abordagem tradicional, na abordagem ágil o planeamento ocorre na preparação de cada um dos ciclos dos incrementos, desta forma o planeamento é focado em cada ciclo e não se despende esforço com algo que não vai ocorrer de imediato. Porém nem todos os métodos incluem no seu processo uma etapa para a definição do plano, como o AUP e o FDD, podendo este ocorrer em simultâneo com outras atividades.
- A recolha de requisitos pode ocorrer numa cerimónia própria para o efeito como acontece nos métodos AD e YP, pode ser incluída no planeamento, ou noutra etapa de arranque, de cada iteração como acontece nos métodos XP, LSD, Openup, AUP, Crystal e DSDM mas pode ocorrer em paralelo com outras atividades ficando registadas em local próprio esperando por discussão na tarefa de análise de requisitos ou outra tarefa similar, como é o caso dos métodos FDD e Scrum.
- A análise de requisitos é uma tarefa que permite a compreensão das funcionalidades que se pretendem implementar com o projeto, nesta tarefa pretende-se decompor o requisito do cliente para um melhor entendimento de quem o vai implementar para isso pode-se recorrer a algumas técnicas como *user stories* no caso dos métodos XP, Scrum, DSDM e YP. Na análise de requisitos pretende-se priorizar as funcionalidades a implementar o que afetará o plano de cada iteração.
- Alguns métodos possuem regras específicas, com o método XP que dita algumas regras que afetam a forma como é feita a codificação do sistema, como a programação em pares e a refactorização, estas são particularidades que não se encontram em outros métodos, no entanto o método LSD também tem valores muito próprios. É a implementação das regras particulares que distancia uns métodos dos outros e fazem a diferença no momento de escolha para a adoção dos métodos.
- As dimensões das equipas podem variar, sendo o DSDM recomendado para equipas pequenas e já os métodos da família Crystal permitem o funcionamento de várias equipas em paralelo, assim como o FDD que pode ter tantas equipas quantas funcionalidades tiver

o sistema. Mas na maioria dos métodos é recomendado que as equipas sejam multidisciplinares.

- Alguns métodos recomendam existência de vários papéis, uns métodos como DSDM, FDD e XP enumeram mais de seis papéis e por outro lado o Scrum enumera apenas três. No entanto, a maioria dos métodos recomenda pelo menos o patrocinador, o cliente e o programador.

- Apenas alguns métodos, como o XP e o FDD, refere práticas a seguir no desenvolvimento propriamente dito de um sistema. Em outros métodos é omissa a forma como a codificação deverá ser feita, sendo que apenas o XP aborda de forma mais profunda esta atividade.

- A forma como as estimativas são feitas depende das técnicas usadas na decomposição dos requisitos, estas estimativas permitem definir o tamanho de cada iteração. A maior parte dos métodos faz as estimativas por funcionalidade como os métodos FDD, LSD, Crystal e DSDM. Outros métodos usam técnicas mais elaboradas como *planning poker* no Scrum.

- Com uma duração curta para cada iteração é importante a comunicação entre os elementos de equipa de modo a que os obstáculos encontrados ao longo do projeto possam ser prontamente ultrapassados sem por em causa o cumprimento dos prazos, custos e âmbito definidos para cada iteração. Assim, a maioria dos métodos recomenda reuniões diárias, já o Scrum refere que essa reunião não deve durar mais de quinze minutos e o XP reforça que as reuniões devem acontecer com todos de pé. Além das reuniões diárias alguns métodos recomendam a realização de *workshops*, reuniões de revisão no final de cada iteração e reuniões de acompanhamento com o cliente.

- Relativamente à monitoria do projeto a maioria recomenda relatórios de progresso para acompanhamento do projeto, o LSD e o Scrum referem o uso de gráficos atualizados diariamente para um melhor acompanhamento do projeto.

- Todos os métodos recomendam a necessidade de validar o sistema através de testes, ou unitários, de integração ou aceitação.

- Quanto ao versionamento do sistema, dado que se trata da implementação de vários incrementos em cada projeto então todos os métodos recomendam a gestão de versões do sistema, este versionamento facilita por exemplo a realização de correções pois ficam contidas numa versão do sistema.

- A abordagem ágil prevê a duração curta dos ciclos de desenvolvimento de modo a que o cliente possa mais cedo ver o sistema a funcionar, desta forma a maioria dos métodos recomenda que cada ciclo de desenvolvimento dure em média de uma a quatro semanas. No entanto, o ASD recomenda um período maior entre quatro a oito semanas e já o LSD e o YP recomendam entre quatro a dezasseis semanas.

4.2 Análise Benchmarking

A análise de *Benchmarking* pode ser implementada nas diversas áreas, nomeadamente na gestão de projetos, de entre outros benefícios é de destacar: o estímulo à mudança, a procura por melhorias nos processos e práticas, a reflexão das organizações com base em análise de indicadores de desempenho e das suas práticas de trabalho, o auxílio na definição de metas estratégicas e operacionais.

Como não se pretende implementar o processo de *Benchmarking* propriamente dito, então seguidamente apenas são descritos os conceitos associados ao mesmo e sem apresentar detalhadamente as fases de implementação do mesmo por não ser alvo neste trabalho.

Serão abordados estudos de *Benchmarking* elaborados pelo PMI, no entanto em Portugal o PMI não possui o mesmo nível de envolvimento nas práticas e conhecimento do estado da gestão de projetos no país, além de que a página oficial do PMI Portugal se restringe à divulgação de eventos e grupos de trabalho de entre os associados do PMI Portugal. Por esse motivo foi alvo do presente estudo a análise parcial de estudos fora de Portugal. Pelas políticas de privacidade impostas nas páginas oficiais do PMI e PMI Brasil apenas serão abordados os resultados obtidos pelo PMI Brasil, que apesar de não refletirem de todo a realidade de Portugal permitem extrapolar o estado da arte na prática de gestão de projetos.

4.2.1 Conceito de Benchmarking

De acordo como PMBOK o *Benchmarking* é um processo contínuo de comparação de produtos, serviços e práticas empresarias entre os mais fortes concorrentes ou empresas reconhecidas como líderes. É um processo de pesquisa que permite realizar comparações de processos e práticas entre organizações para identificar o melhor do melhor e alcançar um nível de superioridade ou vantagem competitiva.

O *Benchmarking* surgiu como uma necessidade de informações e desejo de aprender depressa, como corrigir um problema empresarial. Pois, a competitividade mundial tem aumentado, acentuadamente nas últimas décadas, obrigando as empresas a um contínuo aprimoramento dos seus processos, produtos e serviços, visando oferecer alta qualidade com baixo custo e assumir uma posição de liderança no mercado onde atua. Na maioria das vezes o aprimoramento exigido, sobretudo pelos clientes dos processos, produtos e serviços, ultrapassa a capacidade das pessoas envolvidas, por estarem elas presas aos seus próprios paradigmas.

Na aplicação do *Benchmarking*, como todo o processo, é preciso respeitar e seguir algumas regras e procedimentos para que os objetivos sejam alcançados e exista uma constante melhoria do mesmo. Neste processo existe um controlo constante desde sua implantação (plano do processo) até a sua implementação (ação do processo).

Dependendo do *Benchmarking*, o ponto de referência com o que se mede a informação pode diferenciar-se em quatro tipos (Pessoa, 2008):

- **Interno:** é o método pelo qual se descobrem os conhecimentos sobre os processos através de outros departamentos, sedes, grupos, etc. É o tipo de benchmarking mais utilizado. Trata-se de identificar internamente o modelo a seguir;
- **Competitivo:** com o foco nas organizações que disputam o mesmo mercado através da análise de aspetos como produtos, procedimentos, estratégias, etc. É, na prática, o menos usual uma vez que é quase impossível que as empresas se prestem a facilitar dados que tem que ver diretamente com a sua atividade à concorrência;
- **Funcional:** consiste em identificar as melhores práticas de qualquer tipo de organização, considerada excelente na área funcional sujeita a *Benchmarking*. Estuda uma função específica dentro da indústria.
- **Genérico ou multi-sectorial:** É aquele em que as empresas de diferentes sectores se comparam para identificar e determinar as melhores práticas para determinada área. É neste tipo de benchmarking onde encontramos a maior parte de exemplos práticos e nas quais as empresas estão dispostas a trocar informação e a ser mais verdadeiras.

Pela mesma fonte o *Benchmarking* baseia-se nos seguintes princípios fundamentais:

- **Reciprocidade:** o *Benchmarking* é uma prática baseada em relações recíprocas, na qual todos os participantes beneficiam da partilha de informação;

- **Analogia:** consideram-se processos semelhantes ou análogos, quando há uma transferência de conhecimentos entre os parceiros do estudo;
- **Medição:** o *Benchmarking* é uma comparação de desempenho entre empresas, que passa pela compreensão das razões que justificam os melhores resultados e as diferenças em relação às empresas líderes. Os sistemas de medição e as ferramentas utilizadas nessa análise dependem dos indicadores selecionados;
- **Validade:** a fiabilidade dos estudos pode ser validada por métodos estatísticos. Os participantes não devem confiar na sua intuição ou em suposições que poderão pôr em causa as conclusões do *Benchmarking*.

4.2.2 Benchmarking em Gestão de Projetos

A utilização de métodos ágeis tem vindo a crescer, pelo menos nos mercados internacionais, contudo dada a mentalidade reservada das organizações portuguesas juntamente com a restrita informação publicada sobre este assunto, é difícil apurar, pelo menos sem a aplicação de uma análise de mercado, se o mesmo acontece nos projetos de desenvolvimento de software implementados em Portugal.

Atualmente é fácil encontrar-se, por exemplo através de pesquisas pela internet, informação acerca dos vários métodos de gestão de projetos, mas muito dificilmente se encontra informação acerca de empresas que utilizam efetivamente estes métodos. Existem alguns estudos de mercado relacionados com gestão de projetos mas não propriamente a descrever os métodos adotados. Das pesquisas efetuadas verifica-se que esta é uma preocupação já identificada, nomeadamente no Brasil.

Por isso, esta seção centra-se no estudo de Benchmarking do PMI, que é a associação de gestão de projetos mais reconhecida a nível internacional. Segundo o PMI (*Chapters Brasileiros*), o Benchmarking é um processo contínuo e sistemático para medir e comparar processos organizacionais (PMI Rio, 2012).

As organizações que participam deste processo têm a oportunidade de partilhar informações e experiências que podem ajudá-las a agir para melhorar o seu desempenho ao responderem a questões como: estamos a adotar as melhores práticas? Os processos que usamos são eficientes em relação aos das organizações do mesmo setor?

O estudo de *Benchmarking* em gestão de projetos no Brasil tem por objetivo apresentar um perfil de importantes setores da economia, no que diz respeito à gestão de projetos, ao obter:

- Estatísticas sobre as práticas mais utilizadas em gestão de projetos;
- Nível de adequação dos setores da economia às melhores práticas;
- Ferramentas, práticas e técnicas mais utilizadas;
- Perspetivas e tendências em gestão de Projetos.

O *estudo de Benchmarking* em gestão de projetos no Brasil apresenta o posicionamento de importantes segmentos empresariais brasileiros, servindo como importante ponto de referência para todos que tenham interesse em alcançar a excelência e o sucesso na gestão de projetos.

Seguidamente serão analisados, de uma forma geral, os resultados dos relatórios de Benchmarking em gestão de projetos no Brasil, que é o resultado do trabalho voluntário de vários profissionais de todo esse país, que representa as respectivas seções regionais no Brasil do Project Management Institute - PMI. É, no entanto, de destacar a fonte do estudo:

“Estudo de Benchmarking em Gerenciamento de Projetos Brasil, Project Management Institute – Chapters Brasileiros (<http://www.pmi.org.br>)” (PMI Rio, 2012).

Os relatórios analisados são uma contribuição para toda a comunidade de gestão de projetos do Brasil por meio de sua divulgação livre, gratuita e irrestrita através da disponibilização na página oficial na internet do PMI Brasil.

Estes estudos foram iniciados em 2003 com a parceria da empresa “Project Builder” através da publicação do primeiro questionário, nesse primeiro ano apenas participaram 40 organizações, no entanto houve uma evolução gradual ao longo das oito edições já realizadas, chegamos a 2010 com 460 organizações participantes. O último relatório disponível corresponde à avaliação de 2010 e quanto à avaliação de 2011 a página oficial informa que se encontra ainda a decorrer, isto na data da consulta. Não foi apenas o número de participantes que mudou ao longo da disponibilização dos relatórios, mas o próprio questionário foi aumentando de profundidade, permitindo atualmente uma melhor e mais completa análise dos resultados.

No estudo de 2010 as organizações participantes responderam a um questionário online com pouco mais de uma centena de questões.

São diversos os aspetos abordados no questionário, mas o foco neste trabalho são apenas quatro entre os oito aspetos presentes no questionário (ver Figura 29), ou seja:

- **“Project Management Office”(PMO):** pretende examinar se as organizações estão a contemplar na estrutura da empresa uma área funcional com os gestores de projetos da organização, pretende-se saber quais as tendências observadas.

- **“Processos e Metodologias”**: esta secção examina o grau de adequação das organizações às mais importantes práticas em gestão de projetos, através da avaliação dos processos e métodos existentes.
- **“Tecnologias de Suporte”**: esta secção examina quais as tecnologias de suporte que estão a ser utilizadas pelos profissionais em gestão de projetos e as organizações.
- **“Desempenho e Resultados”**: esta secção examina o desempenho e os resultados auferidos pelas organizações relativamente à gestão de projetos, ao identificar os principais problemas, necessidades e fatores críticos de sucesso.

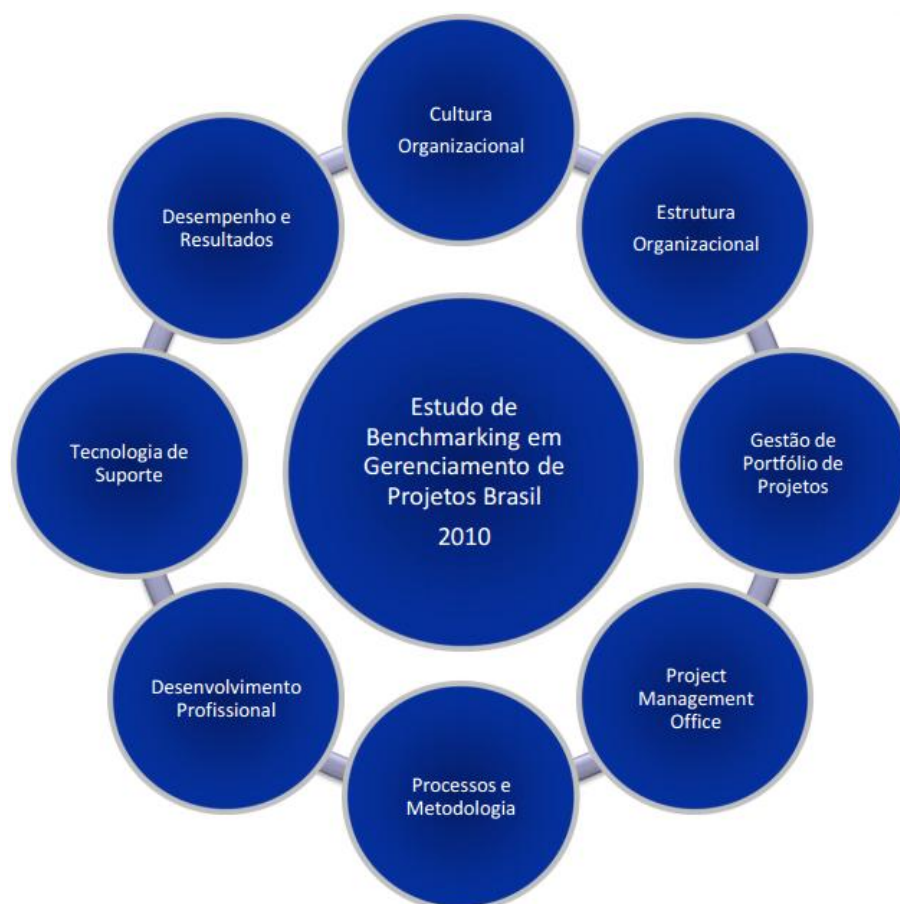


Figura 29 Aspectos abordado no Estudo de *Benchmarking* em análise (extraído de (PMI Rio, 2012))

Apesar do estudo não estar direcionado apenas para organizações com âmbito o desenvolvimento de software, pois apenas 18% das organizações disseram que pertenciam ao sector das TI e 4% de Telecomunicações, mas estão presentes organizações de todos os setores de economia e são interessantes os resultados relacionados com os aspetos descritos e analisados.

O relatório com os resultados do estudo de Benchmarking 2010, disponível para consulta, em questão traz algumas constatações interessantes, nomeadamente (PMI Rio, 2012):

- 46% dos projetos são internos e 54% dos projetos são externos na maioria junto a clientes;
- Quanto à cultura de gestão de projetos nas organizações apenas 2% refere não haver; 30% refere que existe apesar de embrionária; 33% refere existir uma cultura estabelecida porém em apenas algumas áreas da empresa; 10% refere que existe uma cultura estabelecida em toda a empresa mas limitada aos níveis hierárquicos mais técnicos e 25% refere existir uma cultura estabelecida em toda a organização, tanto nos níveis hierárquicos mais técnicos como nos níveis de gestão;
- Relativamente ao nível de resistência das organizações em relação à gestão de projetos 81% refere existir já um baixo nível de resistência e 19% assume um alto nível de resistência;
- Quanto à dedicação dos profissionais à gestão de projetos 60% refere que na maior parte das vezes as atividades de gestão de projetos são executadas em tempo parcial, já 40% refere que na maioria das vezes a gestão de projetos é uma atividade a tempo inteiro;
- Relativamente ao apoio das áreas de chefias das organizações ao desenvolvimento da cultura de gestão de projetos 80% refere haver esse apoio e 20% refere não existir;
- A distribuição quanto à estrutura organizacional da empresa conta com 22% que referem uma estrutura organizacional orientada a projetos; 5% refere uma estrutura matricial forte com maior influência dos gestores de projetos face aos gestores das áreas funcionais; 42% refere uma estrutura matricial balanceada e 30% refere uma estrutura funcional ou departamentalizada sem a designação formal dos responsáveis por projetos;
- Quanto à existência de dedicação exclusiva a projetos 78% confirmam existência dessa dedicação e 22% refere a não existência;
- A representação da dedicação exclusiva dos profissionais a projetos refere 51% como fundamental e com claros benefícios para o sucesso dos projetos; 33% considera algo importante mas cujos benefícios nem sempre são claros; 15% refere como algo que gostaria de implementar mas ainda sem o apoio interno e 1% considera não ser útil ou aplicável à organização;
- A existência oficial do cargo de gestor de projetos na estrutura organizacional 44% refere não existir oficialmente esse cargo; 37% considera que existe mas ainda sem um plano de carreiras específico definido e 19% considera que existe conjuntamente com um plano de carreira específico;

- Relativamente ao equilíbrio entre projetos e tarefas do dia-a-dia 44% refere que as tarefas de rotina do dia-a-dia são prioridade em relação aos projetos, por vezes prejudicando o acompanhamento dos mesmos; 30% refere que a prioridade são os projetos e 25% considera existir um equilíbrio claro, definido e justo entre os esforços dedicados às duas atividades;
- Quanto à existência de bonificações financeiras para o gestor e a equipa do projeto em função dos resultados 43% refere não ter e pretende também vir a ter; 32% refere não ter mas pretende vir a utilizar; 14% refere ter para todos os envolvidos no projeto e 11% considera existir mas apenas para o gestor do projeto;
- Relativamente à existência de um PMO corporativo 61% refere não existir e 39% refere que sim;
- Quanto às áreas da organização que possuem um PMO são de destacar 50% são do sector das TI; 34,7% são de Engenharia; 6,5% das Telecomunicações entre outras áreas;
- Relativamente ao nível de utilização de metodologias de gestão de projetos 46% refere que possui uma metodologia única para a gestão de projetos; 41% refere que a organização possui metodologias, desenvolvidas em algumas áreas específicas e nem todas usam a mesma metodologia e 13% refere não possuir uma metodologia formal;
- Quanto às áreas que utilizam metodologia de gestão de projetos é de destacar as TI com 64,8%, seguida da Engenharia com 44,8% e, entre outros, as Telecomunicações com 10,4%;
- Quanto à utilização efetiva da metodologia de gestão de projetos 52% refere que a metodologia é usada na maioria das vezes; 28% refere que é utilizada de facto; 18% refere utilizar apenas em alguns poucos casos e 1% refere que não é utilizada de facto;
- Quanto aos aspetos considerados na metodologia de gestão de projetos são de destacar os 98,4% o prazo; 96,5% o âmbito e 85,5% o custo, entre outros aspetos;
- Quanto ao nível de utilização de modelos de maturidade em gestão de projetos 52% refere que conhece e utiliza ou pretende utilizar; 31% refere que a organização não conhece nenhum modelo disponível e 17% refere que conhece mas não pretende utilizar;
- Relativamente à utilização de softwares de gestão de projetos 81% refere usar e 19% refere não utilizar;
- Quanto ao grau de utilização dos softwares de gestão de projetos 45% refere usar em 100% dos projetos que realiza; 34% refere usar em pelo menos 50% dos projetos realizados e 21% informa usar em menos de 50% dos projetos realizados;

- Relativamente à frequência na qual os projetos têm alcançado o Sucesso, em termos de prazo, Custo, Qualidade e satisfação do cliente (interno ou externo) 66% refere que sempre ou na maioria das vezes e 34% refere poucas vezes ou nunca;
- Se a organização costuma ter problemas no cumprimento dos prazos estabelecidos para os projetos 78% refere que sim e apenas 22% refere que não;
- Se a organização costuma ter problemas no cumprimento dos custos estabelecidos para os projetos 61% refere que sim e apenas 39% refere que não;
- Se a organização costuma ter problemas de qualidade nos projetos 56% refere que sim e apenas 44% refere que não.

Apesar dos resultados apresentados serem referentes apenas a informação recolhida no Brasil serve como uma referência, apesar de não ser possível extrapolar estes dados para a gestão de projetos em Portugal. Uma vez que são culturas muito díspares, principalmente nesta área de gestão de projetos em que há muito se atribui importância em países como o Brasil mas só aos poucos, e poucos, parece ser dada a devida importância em Portugal.

5. Considerações finais

Este capítulo sintetiza as conclusões deste trabalho, as constatações do mesmo e enumera alguns aspetos relevantes para o trabalho futuro.

5.1 Análise Crítica ao Trabalho

Para cumprir com o objetivo do trabalho, a apresentação de um estudo dos métodos de apoio à gestão de projetos no PDS, recorreu-se em primeira instância à análise da bibliografia disponível. Relativamente aos métodos ágeis a maioria da informação reside ainda na internet, apesar de existirem já alguns livros publicados sobre este assunto, infelizmente não publicados ainda em Portugal, a maior fonte disponível para aprofundar o conhecimento foi mesmo a internet. Acontece que nem sempre é possível garantir a integridade da informação e a tarefa de validar as informações encontradas teve uma dificuldade acrescida porque foram encontrados níveis diferentes de informação. No entanto, sempre que possível foi usada bibliografia disponibilizada pelos criadores dos métodos e assim ultrapassada em parte esta dificuldade.

Com a exposição da informação nos capítulos 2 e 3, relativos ao enquadramento do assunto, conceitos associados e a apresentação do estudo dos métodos propriamente dito, estes capítulos mais descritivos pretendem apresentar os principais conceitos para o entendimento de cada método não esgotando contudo os temas, podendo cada um dos métodos dar lugar a um relatório próprio.

A apresentação do capítulo 4, no qual foram apresentados resultados como a análise comparativa dos métodos ágeis tendo em conta as descrições no capítulo anterior, este capítulo pretende resumir de uma forma simples o estudo dos métodos. Quanto à análise de *Benchmarking* apresentada serviu como solução de contingência pois considero que o ideal teria sido a elaboração e análise de um estudo de mercado contemplando a atual realidade da gestão de projetos em Portugal, no entanto a alternativa não foi possível de concretizar.

5.2 Trabalho Futuro

Nos últimos anos tem surgido muita informação sobre os métodos ágeis, muito se escreve mas ainda existem espaços a serem preenchidos nessa linha de pesquisa, como uma abordagem de métricas para avaliar o desempenho dos métodos e mais conhecimento não tanto sobre a teoria dos métodos mas sobre a sua adoção prática.

O desafio futuro dos métodos ágeis é encontrar maneiras de eliminar alguns dos seus pontos fracos, como a falta de análise de riscos, sem torná-los métodos pesados.

Relativamente a este trabalho, propriamente dito, este poderia ser complementado com:

- Estudo das ferramentas de apoio à utilização dos métodos;
- Compilar as melhores práticas entre os vários métodos ágeis, aplica-las a um caso de estudo e analisar o seu desempenho;
- Análise de mercado sobre o estado da gestão de projetos e utilização de métodos ágeis em Portugal, através da distribuição de questionários e a sua respetiva análise.

5.3 Conclusões

Com a elaboração deste trabalho conclui-se que os métodos ágeis procuram estabelecer novos paradigmas para o desenvolvimento de software.

Os métodos ágeis visam a simplicidade na implementação de um projeto, pois as suas regras são simples de serem seguidas. No entanto, o sucesso da adoção de um método ágil está diretamente relacionado com o comprometimento dos elementos da equipa, dado o elevado grau de informalidade pressuposto. Com a abordagem ágil é mais rápida a entrega da primeira versão do software pedido e com entregas mais rápidas, o cliente não precisa esperar muito para ver o software a funcionar, o que agrega transparência sobre o progresso e o valor entregue, reduzindo também as possibilidades de que grandes falhas aconteçam no projeto.

Os métodos ágeis comportam-se como um suplemento aos modelos existentes e não um método completo.

O desenvolvimento ágil de software tem vantagens e desvantagens, que devem ser pesadas calma e conscientemente antes de se iniciar um projeto. No entanto, considero que as vantagens superam de longe as desvantagens, mas é preciso saber muito bem o que se vai fazer quando se envereda por este caminho e saber como contornar as desvantagens e atingir o sucesso.

Embora não seja a solução para todos os problemas, a abordagem ágil mostra-se uma maneira de trabalhar bastante organizada e iterativa, podendo inclusive contribuir para um ambiente de trabalho mais amigável, portanto é uma boa opção para se obter os diferenciais desejados.

Quanto aos métodos tradicionais estes devem ser aplicados apenas em situações em que os requisitos do sistema são estáveis e os requisitos futuros são previsíveis.

A intensiva competitividade na área de desenvolvimento de software faz com que as empresas procurem sempre o aperfeiçoamento de seus serviços para poder vencer a concorrência, por isso é usado o Benchmarking. O estudo apresentado comprova, pelo menos no Brasil, que está a dar-se mais importância à função da gestão de projetos nas empresas.

Prazo e qualidade além, é claro, de melhor aceitação e adaptação a mudanças são importantes diferenciais que podem ser atingidos utilizando-se os métodos ágeis de desenvolvimento.

Após reflexão sobre o assunto chega-se ainda à conclusão de que o ponto fulcral está nas mudanças que surgem perante as necessidades e, na capacidade ou agilidade de adaptação e de aceitação das mesmas.

REFERÊNCIAS

Bibliografia Citada

Teixeira, R. (2009). *Caracterização da Prática da Gestão de Projectos de Desenvolvimento de Software – Perspectiva das Empresas vs. Perspectiva dos Prestadores de Serviços*. UNIVERSIDADE DE TRÁS-OS-MONTES E ALTO DOURO, Vila Real. Retrieved from http://repositorio.utad.pt/bitstream/10348/293/1/msc_rmcteixeira.pdf

Serghei Floricel, & et al. (2011). *Increasing Project Flexibility: The Response Capacity of Complex Projects*. Project Management Institute, 2011.

Gonçalves, D. (2009). PJMG quality Modelo de actividades da gestão de projectos de desenvolvimento de software no âmbito da qualidade. Retrieved July 1, 2012, from <http://repositorio.utad.pt/handle/10348/338>

Infopédia. (2012). Infopédia -Dicionário da Língua Portuguesa - com Acordo Ortográfico - Projeto. Retrieved July 15, 2012, from <http://www.infopedia.pt/pesquisa.jsp?qsFiltro=0&qsExpr=projecto>

António Miguel. (2006). *Gestão de Projectos de Software* (2ª ed.). Lisboa: FCA.

Warrillow, S. (2011). Project Management Maturity Model. Retrieved July 18, 2012, from <http://www.projectsmart.co.uk/pdf/project-management-maturity-model.pdf>

A guide to the Project Management Body of Knowledge (PMBOK Guide). (2008). (Fourth ed.). Atlanta, USA: Project Management Institute, Inc.

The Standish Group. (2012). The Standish Group. Retrieved September 21, 2012, from <http://blog.standishgroup.com/>

Wikipédia. (2012, April 1). Metodologia (engenharia de software) – Wikipédia, a enciclopédia livre. Retrieved July 21, 2012, from [http://pt.wikipedia.org/wiki/Metodologia_\(engenharia_de_software\)](http://pt.wikipedia.org/wiki/Metodologia_(engenharia_de_software))

Roger Pressman. (2011). *Engenharia de Software: Uma Abordagem Profissional* (7ª ed.). Brasil: ARTMED.

Donato, R. (2010, July 27). Cascata | Caverna do Software. Retrieved July 24, 2012, from <http://voat.com.br/rdal/?tag=cascata>

Leite, J. C. (2000, March 8). O processo de desenvolvimento de software. Retrieved July 24, 2012, from <http://www.dimap.ufrn.br/~jair/ES/c2.html>

Martins, P. V. (2002, October 21). Modelo Iterativo e Incremental. Retrieved from http://w3.ualg.pt/~pventura/ep/aulas_tp/t1_g2.pdf

Jenny, J. (2012, January 2). O Modelo RAD « Juliana Jenny. Retrieved August 15, 2012, from <http://julianakolb.com/2012/02/01/o-modelo-rad/>

Modelos ciclo de vida – Wikipédia, a enciclopédia livre. (2012, May 14). Retrieved July 20, 2012, from http://pt.wikipedia.org/wiki/Modelos_ciclo_de_vida#Modelo_emEspiral

Tomás, P. P. (2010, December 28). Historia de la Informática » Blog Archive » Ingeniería del Software. Retrieved July 30, 2012, from <http://histinf.blogs.upv.es/2010/12/28/ingenieria-del-software/>

Zadrozny, B. (2006, April 29). Engenharia de Software II. Retrieved July 31, 2012, from http://www.ic.uff.br/~bianca/engsoft2/index_arquivos/Aula3-EngSoft2.pdf

Macoratti, J. C. (2005). O processo de Software. Retrieved August 11, 2012, from http://www.macoratti.net/proc_sw1.htm

IBM. (2012). IBM - Rational Unified Process (RUP). Retrieved October 1, 2012, from <http://www-01.ibm.com/software/awdtools/rup/>

Portier, B. (2007, April 5). SOA terminology overview, Part 2: Development processes, models, and assets. Retrieved August 15, 2012, from <http://www.ibm.com/developerworks/library/ws-soa-term2/index.html>

ATDM. (2009). Análise de Sistemas. Retrieved from http://sim2008.files.wordpress.com/2009/09/as_tag_23-11-2009.pdf

Manifesto for Agile Software Development. (2001). Manifesto for Agile Software Development. Retrieved August 15, 2012, from <http://agilemanifesto.org/>

Rodrigo. (2011, June 21). Engenheiro de Software - Artigos, Tutoriais, Livros e Dicas Atuais sobre o mundo gerenciamento de Software! Retrieved August 2, 2012, from <http://rtiweb.net/engenheiro-software/2011/06/21/introducao-adaptive-software-development/#axzz24V86Oh82>

Scott, Ambler. 2009. "Agile Modeling and the Rational Unified Process (RUP)." <http://www.agilemodeling.com/essays/agileModelingRUP.htm>.

Cockburn, A. (2008). Alistair.Cockburn.us | Crystal. Retrieved September 13, 2012, from <http://alistair.cockburn.us/Crystal>

Brito, R. (2011). Família de Metodologia Crystal. Retrieved from <http://pt.scribd.com/doc/92021041/Trabalho-Metodologia-Agil-Crystal>

DSDM Consortium, D. C. (2012). Atern E-Book – DSDM Atern. Retrieved September 15, 2012, from <http://www.dsdm.org/dsdm-atern/atern>

Garcia, F. P., & et al. (2006, Agosto). easYProcess - Processo de desenvolvimento de software para o DSC. Retrieved September 16, 2012, from <http://www.dsc.ufcg.edu.br/~yp/>

Thomas, D. (2006, December 27). The Essential Unified Process: New Life for the Unified Process | Dr Dobbs's. Retrieved September 16, 2012, from <http://www.drdobbs.com/architecture-and-design/the-essential-unified-process-new-life-f/196702101?pgno=2>

Jeffries, L. R. (2012). What is Extreme Programming? | xProgramming.com. Retrieved August 2, 2012, from <http://xprogramming.com/what-is-extreme-programming>

Don Wells. (2009, September 18). XP flow Chart. Retrieved August 2, 2012, from <http://www.extremeprogramming.org/map/project.html>

Warden, S. (2007). *The Art of Agile Development* (1st ed.). Sebastopol, USA: O'Reilly Media, Inc, USA.

Audy, J. H. N. (2012, July 12). Metodologia | Jorge Kotick Audy | Página 2. Retrieved July 25, 2012, from <http://jorgekotickaudy.wordpress.com/category/metodologia/page/2/>

De Luca, J. (2002). Feature Driven Development | The portal for all things FDD. Retrieved September 16, 2012, from <http://www.featuredrivendevelopment.com/>

Poppendieck, M., & Poppendieck, T. (2011). Poppendieck.LLC. Retrieved September 16, 2012, from <http://www.poppendieck.com/>

The Eclipse Foundation. (2012, June 1). OpenUP. Retrieved September 17, 2012, from <http://epf.eclipse.org/wikis/openup/>

Sabbagh, R. (2010, August 14). Scrum em Ação. Retrieved September 19, 2012, from <http://scrumemacao.com.br/web/>

Schwaber, K., & Armstrong, A. (2012). Scrum.org > Home. Retrieved September 19, 2012, from <http://www.scrum.org/>

ambithus. (2012). Certified Scrum Master 2012 (SCRUM - Guia de referência).

Sadhan. (2010, August 6). Agile Methodology | Digital Thought. Retrieved September 20, 2012, from <http://www.sadhanbiswas.com/myblog/techsavvy/agile-methodology/>

Guia da Equipe de Recursos de Migração de Perfil do Usuário. (2006, November 30). Retrieved July 25, 2012, from <http://technet.microsoft.com/pt-br/library/bb490196.aspx>

Sousa, E. (2011, October 3). Scrum «Gerenciamento Estratégico. Retrieved September 22, 2012, from <http://gerenciamentoestrategico.wordpress.com/tag/scrum/>

Microsoft. (2006, November 30). Guia da Equipe de Recursos de Migração de Perfil do Usuário. Retrieved July 25, 2012, from <http://technet.microsoft.com/pt-br/library/bb490196.aspx>

Pessoa, G. A. (2008). Benchmarking: Noções básicas. Retrieved September 20, 2012, from <http://pt.scribd.com/doc/6046970/Benchmarking-Noco-es-basicas>

PMI Rio. (2012). Benchmarking - PMI Rio. Retrieved September 20, 2012, from <http://pmi-rio.ning.com/page/benchmarking-1>

benchmarking_gp_2010_geral.pdf. (n.d.). Retrieved from http://www.mp.go.gov.br/portalweb/hp/33/docs/benchmarking_gp_2010_geral.pdf

Bibliografia Consultada

A guide to the Project Management Body of Knowledge (PMBOK Guide). (2008). (Fourth ed.). Atlanta, USA: Project Management Institute, Inc.

Chrissis, M. B., Konrad, M., & Shrum, S. (2011). *CMMI for Development - Guidelines for Process Integration and Product Improvement* (Third ed.). Boston: Addison Wesley.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum* (1st ed.). New Jersey, USA: Addison-Wesley Educational Publishers Inc.

Victor Sequeira Rodão. (2007). *Gestão de Projectos - Abordagem Instrumental ao Planeamento, Organização e Controlo* (2ª Edição.). Lousã: Monitor.

Warden, S. (2007). *The Art of Agile Development* (1st ed.). Sebastopol, USA: O'Reilly Media, Inc, USA.

António Miguel. (2006). *Gestão de Projectos de Software* (2ª ed.). Lisboa: FCA.

ANEXO 1 – MANIFESTO PARA O DESENVOLVIMENTO ÁGIL DE SOFTWARE

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

ANEXO 2 – PRÍNCÍPIOS SUBJACENTES AO MANIFESTO ÁGIL

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams. 24

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

