**Maria João Moreno Semedo**

# Ganhos de produtividade e de sucesso de Metodologias Ágeis VS Metodologias em Cascata no desenvolvimento de projectos de software

Dissertação apresentada para a obtenção do grau de Mestre em Engenharia Informática e Sistemas de Informação no Curso de Mestrado: Engenharia Informática e Sistemas de Informação pela Universidade Lusófona de Humanidades e Tecnologias

Orientador: Professor Doutor Jose Rogado

Co-Orientador: Professor Doutor Rui Ribeiro

**Universidade Lusófona de Humanidades e Tecnologias**

**ECATI**

**Lisboa**

**Março de 2012**

# Dedicatória

Aos meus pais

À minha avó

Aos meus irmãos

Ao Carlos Bruno Levy

Aos meus sobrinhos

# Agradecimentos

Ao Renato Semedo

Ao Carlos Levy

Ao Professor Rui Ribeiro

Obrigada

## Resumo

Hoje em dia muitas empresas procuram cada vez mais inovar os seus sistemas informáticos. É hoje uma realidade que as novas tecnologias, os novos sistemas devem disponibilizar a informação correcta, no formato certo, à pessoa certa e em tempo exacto. "Ontem" tínhamos sistemas inflexíveis, hoje, têm que ser transversais, partilhados e em rede. Inerentes a esta inovação, as Empresas também pretendem que os seus sistemas sejam práticos, funcionais e rentáveis. Assim, tendo como base estas premissas, as mesmas procuram no mercado "parceiros" que possuam estas características e que lhes possam garantir/oferecer o que necessitam para desenvolverem as suas áreas de negócios, de uma forma sustentada, acrescentando valor aos seus próprios Clientes. Neste contexto, as Empresas prestadoras deste tipo de serviço para se poderem diferenciarem de uma forma positiva no mercado, quer nacional quer estrangeiro, devem adoptar sistemas que possam ir ao encontro das necessidades dos seus actuais Clientes e dos seus Clientes futuros.

Várias empresas já tiveram um histórico de problemas relativos a elevados investimentos em que estes não conseguiram atingir os objectivos a que se propuseram atingir, visto que, muitas vezes não efectuam uma escolha adequada dos métodos, do âmbito e dos critérios que necessitam para desenvolverem esses mesmos projectos, originando perda de qualidade e diminuição da sua produtividade. Como reflexo destas escolhas, não conseguem alcançar os objectivos de negócio previamente definidos e propostos.

Neste âmbito, é fundamental perceber a importância da utilização de uma metodologia, a qual se pode definir como: um conjunto de técnicas, procedimentos, ferramentas e documentação que sustentam e auxiliam os gestores/responsáveis no desenvolvimento e implementação de um sistema de informação. Dentro de uma metodologia, existe ainda várias fases que orientam os gestores/responsáveis do projecto para a correcta definição e escolha de técnicas apropriadas de acordo com as etapas que ocorrem durante o desenvolvimento e implementação de um sistema de informação.

O que muitos desconhecem, é que muitas vezes o sucesso de um projecto de software depende de vários factores, tais como o planeamento, a definição dos objectivos a atingir e principalmente na escolha da metodologia mais adequada e compatível com o projecto em curso.

Hoje em dia muitas empresas adoptaram a fundo as metodologias ágeis como sendo as mais eficazes e acessíveis à mudança proporcionando melhorias em diversos aspectos

durante o processo de desenvolvimento de software, tais como: o tempo de entrega, qualidade do produto final e a redução de custos operacionais.

Por outro lado temos a modelo em cascata/tradicional que foi e é a metodologia mais conhecida e a mais antiga. Para desenvolver um projecto nesta metodologia é preciso seguir cada etapa sequencialmente, ela implica que, um grande esforço seja feito em duas das fazes consideradas cruciais no desenvolvimento do projecto: levantamento de requisitos/necessidades e o design.

O presente trabalho tem como objectivo principal o estudo dos ganhos de produtividade e de sucesso de metodologias ágeis vs. metodologias em cascata no desenvolvimento de projectos de software.

**Palavras-chave**: metodologias ágeis, metodologias tradicionais, cascata, scrum, ganhos de produtividade, sucesso.

## Abstract

Nowadays many companies are increasingly looking to innovate their systems. It is now a reality that new technologies, new systems must provide the correct information in the right format, to the right person and exact time. "Yesterday" had inflexible systems today have to be cross-sharing and networking. Inherent in this innovation, the companies also claim that their systems are practical, functional and profitable. Thus, based on these premises, they seek to market "partners" that possess these characteristics and that they can ensure / provide what they need to develop their business area, in a sustained manner, adding value to their own customers. In this context, the companies providing such service to be able to differentiate in a positive way in the market, both domestic and foreign, should adopt systems that can meet the current needs of its clients and its future customers.

Several companies have already had a history of problems relating to high investments in that they failed to achieve the objectives they set out to achieve, since they often do not make a proper choice of methods, scope and the criteria they need to develop these same projects, resulting in quality loss and decreased productivity. As a consequence of these choices, fail to achieve business objectives previously defined and proposed.

In this context, it is essential to realize the importance of using a methodology which can be defined as: a set of techniques, procedures, tools and documentation that support and assist managers / guardians in the development and implementation of an information system. Within a method, there are still several steps that guide managers / project leaders for the correct definition and choice of appropriate techniques according to the steps that occur during the development and implementation of an information system.

What many do not know, is that often the success of a software project depends on several factors, such as planning, defining the objectives to be achieved and especially in choosing the most appropriate methodology and compatible with the ongoing project.

Nowadays many companies have adopted agile methodologies in depth as the most effective and accessible to change by providing improved in several respects during the process of software development, such as delivery time, product quality and reduce operating costs.

On the other hand we have the waterfall model / traditional approach was and is the oldest and best known. To develop a project of this methodology is necessary to follow each step sequentially, it implies that a major effort be made in two of doing considered crucial in the development of the project: requirements gathering / requirements and design.

The present work has as main objective the study of the productivity gains and successful Agile vs. waterfall methodologies in developing software projects.

**Keywords:** Agile, traditional methodologies, waterfall, scrum, productivity gains, success.

# Siglas

ASD - Adaptative Software Development

APs - Áreas de Processos

CMMI - Capability Maturity Model Integration

CMM - Capability Maturity Model

DSDM - Dynamic Systems Development Method

EIA SECM - Electronic Industries Alliances's Systems Engineer Capability Model

FDD - Feature Driven Development

IBM - International Business Machines

IPD-CMM - Desenvolvimento Integrado de Processo e Produto

NPV - Net Present Value (valor presente líquido)

PIP - Process Improvement Proposal

PSP - Personal Software Process

RAD -Rapid Application Development

ROI – Return on Investment

RUP - Rational Unified Process

SECM - Systems Engineer Capability Model

SEI/CMM - Software Engineering Institute/ Capability Maturity Model

SW-CMM – Software Capability Maturity Model

C3 – Nome do projecto da Chrysler Corporation

UML - Unified Modeling Language

XP - Extreme Programming

# Índice Geral

# Índice de Tabelas

# Índice de Figuras

# Introdução

Actualmente, quanto mais rápido a tecnologia evolui, mais intensa se torna a necessidade de mudar os sistemas de informações. Como resultado desta realidade, produtores de software sentem necessidade de uma constante procura de novos serviços, os quais têm como finalidade oferecer aos seus Clientes os seus novos produtos para que possam trabalhar com mais confiança e qualidade.

Hoje em dia, muitas organizações estão dependentes das indústrias de software. Como já verificámos, existem vários problemas relacionados com o processo de desenvolvimento de sistemas. Assim, a título de exemplo, pode-se destacar: alto custo, alta complexidade, dificuldade de manutenção, um grande aumento entre as necessidades dos utilizadores e o produto desenvolvido (Sommerville, 2003).

Muitos acreditam que o problema que têm surgido ao longo dos anos deve-se ao facto da utilização da "primeira" metodologia (cascata). Desenvolveram e implementaram uma nova metodologia (metodologias ágeis), no entanto também existem muitos que, apostam e defendem a sua utilização como sendo um processo mais simplificado.

Durante todo o processo de desenvolvimento deste trabalho, serão abordados alguns conceitos considerados importantes para que se possa compreender o contributo de cada uma das metodologias no desenvolvimento de software. Será igualmente efectuado um enquadramento histórico sobre as mesmas, como surgiram, quais é que existem, qual é a mais procurada e por fim o objectivo principal deste trabalho que consiste em apurar efectivamente quais são os ganhos de produtividade e de sucesso entre as duas metodologias de desenvolvimento de software.

# Capitulo I

## 1.1. Objectivo Geral

Este trabalho tem como objectivo geral a comparação entre as duas metodologias (ágeis e cascata), mencionando qual é a que tem mais sucesso, e em termos de produtividade, qual é a que obtêm mais lucro no desenvolvimento de software.

## 1.2. Objectivo específico

- Com esse trabalho, pretende-se fazer a análise destas duas metodologias, separadamente, efectuando uma pequena comparação entre elas;
- Ficar a conhecer de uma forma mais aprofundada o modelo em cascata e, descrever de uma forma sucinta as suas características, desvantagens e vantagens;
- Investigar as empresas que utilizam o modelo Scrum, para obter informações e opiniões relativas a utilização das metodologias ágeis, principalmente o modelo Scrum;
- Descrever as vantagens, desvantagens e as práticas do método Scrum.
- Identificar e descrever qual das duas metodologias obtêm mais sucesso e é mais lucrativa.

## 1.3. Organização

Este documento está organizado da forma que a seguir se descreve. No capítulo 2 irão ser abordados os conceitos que compreendem o domínio, o enquadramento histórico das duas metodologias, as suas vantagens, desvantagens e as suas falhas. Serão igualmente mencionadas as principais estratégias de cada uma delas, e por fim compará-las. No capítulo 3 serão abordados os ganhos de produtividade, de sucesso de cada uma das metodologias e apresentado um exemplo da metodologia com mais sucesso. Por fim será apresentado a conclusão, as limitações encontradas ao longo do trabalho e os trabalhos futuros.

# Capitulo II

## 2.1. Revisão Bibliográfica

Antigamente quando se pensava em desenvolver um projecto de software, não se fazia qualquer plano, e nem tinham uma condução adequada para projecto. Esses dois factores foram uma das principais causas de grande preocupação e de insucesso dos projectos até aos meados dos anos 70. Existem outros factores que condicionam ou apoiam o insucesso e/ou fracasso dos projectos. Nessa época houve muito desperdício de recursos por partes das empresas, pois empregavam uma boa parte dos seus recursos nesses projectos (fracassado ou de má qualidade) e no fim não obtinham qualquer lucro e viam os seus gastos a aumentarem.

Após todos esses insucessos nessa época, em 1970 Winston Royce criou o primeiro modelo de desenvolvimento de software, no qual denominou de Modelo de Ciclo de Vida Clássico ou Cascata. Naquela época esse modelo foi responsável pela grande revolução no que diz respeito ao desenvolvimento de projectos, alterando a forma como o projecto era desenvolvido e planeado, pois com esse modelo os responsáveis podiam desenvolver os seus projectos de uma forma organizada e com uma sequência programada para que esta pudesse ser seguida.

O método de desenvolvimento do modelo em cascata é sequencial ou por fases, ou seja, cada fase tem de ser cumprida ou finalizada para se avançar para a fase seguinte.

Após esse modelo, começaram a surgir outros modelos em que as suas finalidades são as mesmas do modelo em cascata. A esses métodos deu-se o nome de Metodologias Tradicionais.

No seu recente artigo, Royce chegou à conclusão que tradicionalmente quando se usam processos de desenvolvimento de software baseados no modelo em cascata a gestão do projecto baseia-se nos seguintes princípios (Royce 2000):

- Congelar os requisitos antes da fase de desenho;
- Proibir a criação de código antes de existir um desenho detalhado do sistema;
- Usar uma linguagem de programação de alto nível;
- Completar os testes das unidades constituintes do sistema antes de iniciar a sua integração;
- Manter uma rastreabilidade detalhada entre todos os artefactos do processo;

- Documentar extensivamente cada fase do desenho;
- Verificar a qualidade através de uma equipa independente;
- Inspeccionar tudo;
- Planear tudo previamente e com grande fidelidade;
- Controlar o código fonte.

As etapas do modelo em cascata seguem-se, de uma forma linear (Larman, 2002). E cada fase concluída deste modelo, gera um marco, que é geralmente algum documento, um protótipo de software ou mesmo uma versão do sistema (Neto, 2004).

Sommerville (2004) afirmou que muitas metodologias pesadas são desenvolvidas em cima do modelo em cascata.

De acordo com Pressman (2006) e Paula Filho (2003) o modelo em cascata caracteriza-se pelo seu carácter preditivo, prescritivo, sequencial, burocrático, rigoroso, orientado a processos, dados formais e controlado, que tem, o sucesso alcançado desde que esteja em conformidade com o que foi planeado.

O foco principal das metodologias tradicionais é a previsibilidade dos requisitos do sistema, que traz uma grande vantagem: tornar os projectos completamente planeados, facilitando a gestão dos mesmos, mantendo-os sempre uma linha e caracterizando o processo como bastante rigoroso (Oliveira).

Alguns autores apontam as metodologias tradicionais, principalmente o modelo em cascata, como metodologias pesadas e inadequadas para o desenvolvimento de software.
Pressman (1995) aponta alguns problemas identificados no modelo em cascata:

- Os projectos reais raramente seguem o fluxo sequencial que o modelo propõe;
- Alguma iteração sempre ocorre e traz problemas na aplicação do paradigma;
- Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente;
- O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projectos;
- O cliente deve ter paciência. Uma versão de trabalho dos programas não estará disponível até um ponto tardio do cronograma do projecto. Um erro crasso, se não for detectado até que o programa de trabalho seja revisto, pode ser "catastrófico".

Sete anos mais tarde criticou pela segunda vez o modelo em cascata, em que ele afirma que, muitas "metodologias" pesadas são desenvolvidas no modelo em cascata o que dificulta o controlo do projecto, pois a cada alteração em determinado ponto do projecto, como os requisitos, será necessário uma volta ao início do mesmo para alteração de documentação ou outro marco (PRESSMAN 02).

O pesquisador Tom Gilb (1999), desencoraja o uso do modelo clássico (cascata) em grandes softwares, estimulando o desenvolvimento incremental como um modelo que apresenta menores riscos e maiores possibilidades de sucesso.

Mais tarde surgiram um grupo de modelos de desenvolvimento de software, no qual foram-lhes atribuídos o nome de Metodologias Ágeis. Essas metodologias são tidas como metodologias fáceis de trabalhar porque nelas não é obrigatório o modo sequencial, ou seja, pode-se concluir a fase seguinte sem que a fase anterior esteja concluída e, com isso não é preciso aguardar que uma fase seja concluída para dar continuidade à fase seguinte do projecto.

Essas metodologias foram bem aceites pelos engenheiros, e cada vez mais está a ganhar o seu espaço mercado.

Segundo Scott W. Ambler, Metodologia Ágil "é uma metodologia baseada na prática para modelagem eficaz de software".

Beck (1999) citou os valores que considera fundamentais dentro do manifesto ágil, que foi a inspiração para o aparecimento dos doze princípios básicos, que são características que diferenciam as metodologias ágeis de outras metodologias chamadas tradicionais.

Cockburn em 2000 definiu o desenvolvimento ágil de software como uma abordagem de desenvolvimento que trata problemas das mudanças rápidas.

Segundo Fowler (2001) as metodologias ágeis são uma reacção às metodologias tradicionais e conceituais.

Highsmith (2002) define a agilidade como a habilidade de criar e responder as mudanças com respeito ao resultado financeiro do projecto num ambiente de negócios turbulento. Agilidade é a habilidade de balancear flexibilidade com estabilidade.

Métodos, práticas e técnicas para o desenvolvimento ágil de projectos que prometem aumentar a satisfação do cliente (BOEHM, 2003) para produzir alta qualidade de software e para acelerar os prazos de desenvolvimento de projectos (ANDERSON, 2003).

Xispe (2004), diz que para circular esse problema de constantes alterações, assumindo que essas mudanças fazem parte do processo de desenvolvimento, surgiram os métodos ágeis, aqueles com o foco no código e optimizados para alterações de requisitos, como a XP-Extreme Programming (WELLS 04), esses métodos também prezam pela qualidade do software, mas a sua filosofia de desenvolvimento é diferente, dando ênfase principalmente no código, sendo que as alterações necessárias não devem levar em tanto tempo.

Segundo Teles (2004), os processos ágeis de desenvolvimento compartilham a premissa de que o cliente aprende sobre as suas necessidades, na medida em que é capaz de manipular o sistema que está sendo produzido e, com base no feedback do sistema, ele reavalia as suas necessidades e prioridades, criando mudanças que devem ser incorporadas ao software.

Cockburn et al. (2001) concluíram que a maioria dos métodos ágeis não possuem nada de novo. O que as diferencia das metodologias tradicionais são os enfoques e os valores.

Hoje em dia existem dois tipos de metodologias:

- Metodologias Tradicionais ou pesadas (Cascata, RUP, CMMI, PSP, etc.);
- Metodologias Ágeis ou leves (XP, SCRUM, Agile Modeling, ASD, FDD e LEAN).

## 2.2. Metodologias de desenvolvimento de software - Enquadramento

Segundo Pressman (2001), o processo de desenvolvimento de software é um conjunto de tarefas requeridas para a produção de um software com alta qualidade. O resultado do processo é um produto (software) que reflecte a forma como o processo foi realizado. Embora existam vários processos de desenvolvimento de software, existem também actividades genéricas comuns a todos eles (SOMMERVILLE, 2004):

- *Especificação do software:* definição do que o sistema deve fazer; definição dos requisitos e das restrições do software;
- *Desenvolvimento do software:* projecto e implementação do software, o sistema é desenvolvido conforme sua especificação;
- *Validação do software:* o software é validado para garantir que as funcionalidades implementadas estejam de acordo com o que foi especificado;
- *Evolução do software:* o software evolui conforme a necessidade do cliente.

O processo de desenvolvimento de software fornece uma interacção entre utilizadores e engenheiros de sistemas, entre utilizadores e tecnologia, entre engenheiros de sistemas e a tecnologia.

Desenvolver um software é na realidade um processo de aprendizagem interactivo e o resultado é uma corporização de conhecimento recolhido, transformado, organizado à medida que o processo é conduzido.

Ficou comprovado que um projecto de software tem duas dimensões:

- *Gestão de projectos:* ocupa-se do modo como planear e controlar adequadamente as actividades de engenharia, de modo a cumprir todos os objectivos do projecto em termos de custo, qualidade, prazo, etc.;
- *Engenharia de software:* ocupa-se da construção do sistema de software e, centra-se nas questões técnicas (desenhar, codificar, testar, etc.).

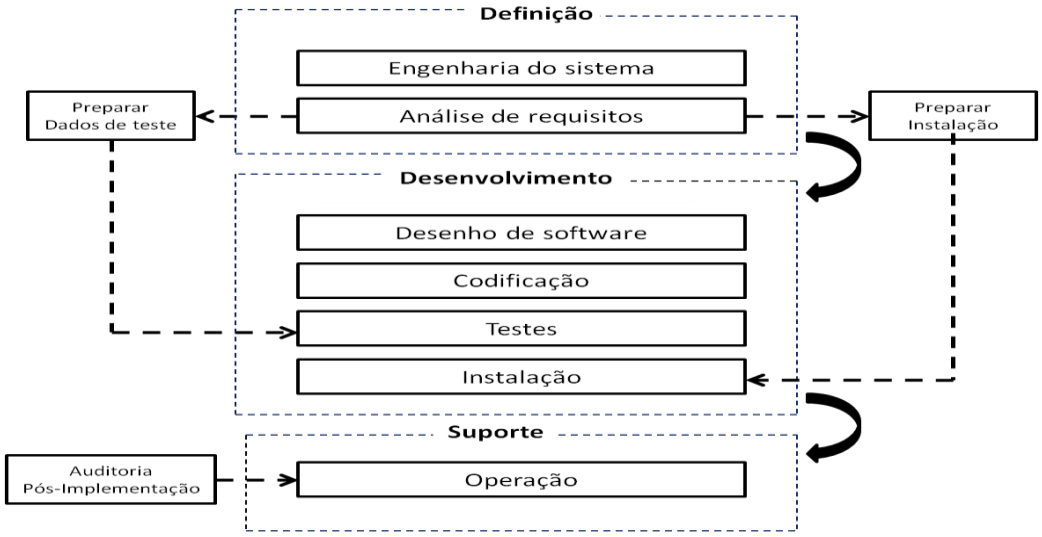



Figura 1- Fases genéricas do desenvolvimento de software do ponto de vista da engenharia de software
Fonte: Adaptado por Miguel (2003)

Existe um modelo ***genérico do processo de desenvolvimento de software***, onde o processo de software é caracterizado por fases: Quando se define um pequeno número de actividades que são aplicáveis a todos os projectos de software, independentemente da respectiva dimensão e complexidade, a isso dá-se o nome de ***estrutura comum do processo de desenvolvimento***. A fase seguinte é o ***conjunto de tarefas*** que permitem a adaptação das ***actividades de estrutura*** as características do projecto de software e aos requisitos da equipa de projecto.

As ***actividades envolventes*** (garantia de qualidade do software, gestão de configuração do software e as medidas) que são independentes de qualquer das actividades da estrutura e ocorrem ao longo de todo o processo.

A figura abaixo ilustra como o processo de desenvolvimento de software pode ser caracterizado.

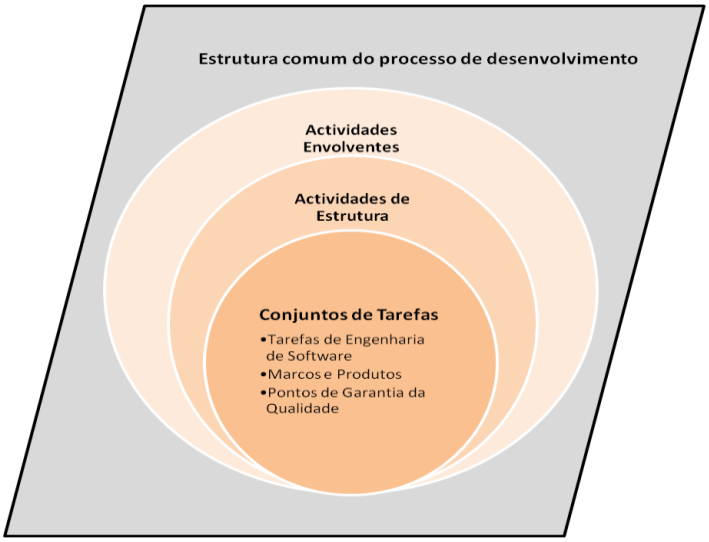



Figura 2 – o processo de desenvolvimento de software
Fonte: Pressman (2000)

Após uma descrição breve sobre o desenvolvimento de software, a sua estrutura, desde o seu desenvolvimento até a sua concepção, agora chegamos ao ponto mais importante do trabalho em que, se vai falar das metodologias que são utilizadas para o desenvolvimento de software.

A primeira metodologia surgiu no princípio da década de 70, sendo esta, utilizada durante anos, até que, anos mais tarde surgiu no mercado um novo tipo de metodologia e até agora, é uma das mais utilizadas.

Pode-se definir uma metodologia de desenvolvimento de software como sendo, um conjunto de actividades que servem de auxílio à produção de software. Com isto, estas actividades resultam num produto que demonstra como foi conduzido todo o processo de desenvolvimento.

Uma metodologia de desenvolvimento de software é a representação simplificada do processo (SOMMERVILLE, 2004) que combina a descrição do trabalho, procedimentos e convenções utilizadas pelos seus membros (COCKBURN, 2000).

Pode-se afirmar que dentro do desenvolvimento do software, existe um dos desafios mais importantes e comuns que é, a entrega de um produto que, satisfaça as verdadeiras necessidades dos clientes, dentro de um prazo previsto e com um orçamento previsto.

A engenharia de software tem um papel importante no desenvolvimento do software, pois oferece as metodologias que servem de grande auxílio para os gestores/desenvolvedores durante todo o processo de desenvolvimento: metodologias tradicionais e metodologias ágeis.

## 2.3. Metodologias Tradicionais

Nos meados da década de 60 até ao princípio da década de 70 todas as actividades que diziam respeito ao desenvolvimento de software eram executadas de uma forma desorganizada, não eram estruturadas e sem qualquer planeamento, o que no fim originavam um produto final de má qualidade porque, não tinham qualquer documentação de sustentação do seu desenvolvimento para que pudesse ser seguida pelos seus desenvolvedores e, para além disso o produto, na maioria das vezes era entregue fora de prazo definido, não era do agrado do cliente e nem sempre era entregue ao cliente o que tinha sido pedido. Quase 70% dos produtos desenvolvidos não correspondiam às verdadeiras necessidades dos utilizadores e o custo do produto era muito elevado.

Muitos clientes não ficavam satisfeitos com o produto final, pois empregavam uma boa parte dos seus recursos e, com isso os gastos aumentavam ainda mais. Através destes factos essa época ficou conhecida como a época da crise de software (Pressman 02).

Com todas essas necessidades sentiu-se a obrigação de tornar o desenvolvimento de software num processo bem organizado, estruturado e bem planeado para que atende-se a todas as necessidades dos utilizadores e dos clientes. Com isso, nos meados da década de 1970 Winston W. Royce apresentou o primeiro modelo de desenvolvimento de software denominada como modelo em cascata ou modelo de ciclo de vida clássico.

Mas tarde apareceram outros tipos de metodologias tradicionais mas, a mais utilizada actualmente é o modelo em cascata.

As metodologias tradicionais, muitas vezes são denominadas como metodologias pesadas e tem como característica a sua divisão por fazes e/ou etapas.

Segundo SOARES (2004), *essas metodologias surgiram em um contexto de desenvolvimento de software muito diferente do actual, baseado apenas em um mainframe e terminais burros. Na época, o custo de fazer alterações era muito alto, uma vez que o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do software, como analisadores de código. Por isso, o software era todo planeado e documentado antes de ser implementado.*

### 2.3.1. RUP - Rational Unified Process

O RUP foi criado pela Rational Software Corporation que mais tarde foi comprada pela IBM, onde passou a ser chamada de IRUP, tornando-se numa marca na área de software, em que fornece técnicas a serem seguidas pelos membros da equipa de desenvolvimento de software com o objectivo a aumentar a sua produtividade no processo de desenvolvimento.

Também ele é considerado como a principal framework utilizada comercialmente no mercado e mantém um controlo efectivo sobre a gestão de projectos e da elevada qualidade de produção de software.

Pode-se aplicar o RUP a vários tipos de projectos, pois sendo ele considerado uma framework genérica para os processos de desenvolvimento, atendendo a todas as necessidades e ao tamanho de cada projecto. Pois, a mesma adapta-se a qualquer projecto, seja ele grande ou pequeno.

Para sua concepção, esta metodologia usa uma abordagem de orientação a objectos, em que a sua documentação e projecção é feita utilizando a UML para ilustrar os processos em acção.

O RUP é, por si só, um produto de software. É modular e electrónico, e toda a sua metodologia é apoiada por diversas ferramentas de desenvolvimento integradas. É uma metodologia completa para viabilizar o sucesso de grandes projectos de software. Sendo ele uma metodologia de desenvolvimento iterativo, concentra-se na redução de riscos do projecto.

Esta metodologia apresenta algumas características “negativas” que são: os aumentos dos requisitos da documentação, a necessidades que vem por parte dos clientes (pois eles têm de ter alguns conhecimentos em UML) e ao facto de estar ligado ao proprietário do software IBM Rational.

As suas principais características são: um desenvolvimento iterativo e incremental, orientado a objectos, com foco na criação de uma arquitectura robusta, análise de risco.

Segundo Kruchten (KRUCHTEN, 1996), é um processo de construção de sistemas de software feito em pequenos passos, e é apresentado como um modelo mais detalhado.

O RUP é uma metodologia iterativa, ou seja, trabalha em ciclos de desenvolvimento. Com isso há vários ***benefícios*** como (KRUCHTEN, 2000):

- *Gestão de Requisitos:* após do começo do desenvolvimento, é comum que os requisitos mudem, fazendo com que haja alterações na documentação. Desenvolvendo iterativamente, essas alterações são facilmente de gerir;
- *Integração dos Elementos:* quando cada módulo é desenvolvido este é integrado ao sistema como um todo, evitando que este seja um problema no final do projecto;
- *Gestão de Riscos:* a cada iteração é possível analisar pontos críticos e planear estratégias para não perder tempo durante o desenvolvimento;
- *Testes:* os testes são realizados no final de cada módulo, permitindo que erros e não-conformidades possam ser tratados ainda dentro do mesmo ciclo.

***Arquitectura do RUP***

A metodologia RUP identifica cada ciclo de desenvolvimento do projecto em quatro fases, que são (KRUCHTEN 2000):

- **Concepção**: nesta fase define-se o objectivo total do projecto;
- **Elaboração**: visa eliminar os principais riscos e principalmente definir a arquitectura que o sistema será desenvolvido. Sendo que esta deverá ser a mais estável e que permita o sistema a evoluir;
- **Construção**: nesta fase o sistema começa a ser desenvolvido (codificado), sendo que este desenvolvimento, assim como todo o projecto, ocorre de maneira iterativa. Ao final de cada ciclo, pode ser gerada uma versão utilizável que crescerá a cada iteração;
- **Transição**: esta fase é responsável pela implantação do sistema. O Software é entregue aos utilizadores para que estes possam fazer suas considerações, ou seja, fazer a avaliação do produto.

O RUP tem também nove disciplinas que servem para organizar logicamente as actividades da engenharia de software de acordo com a sua natureza, ou seja, elas descrevem o que deve ser feito em cada iteração a respeito de actividades responsáveis e documentos. Essas disciplinas são chamadas de workflows. Dessas disciplinas existem seis que são da engenharia de software e três que são de suporte.

*As noves disciplinas são:*

- *Modelagem de Negócios:* Visa entender a estrutura e a dinâmica da organização para qual se vai desenvolver o projecto;
- *Requisitos:* Definir, documentar e organizar as funcionalidades do sistema, bem como gerir o objectivo e as mudanças destas funcionalidades;
- *Análise e Projecto:* Utiliza a linguagem UML. Traduz os requisitos em uma linguagem que descreve como o sistema deve ser implementado;
- *Implementação:* Fluxo muito importante. É nesta etapa que o sistema é codificado na linguagem de programação especificada;
- *Testes:* Faz teste em todo o sistema, verificando se este está de acordo com o que foi especificado e, também valida a integração entre objectos já implementados. Tem como objectivo, encontrar defeitos antes da implementação do produto;
- *Implantação:* Implanta o sistema para o utilizador final, dando ênfase no treinamento, instalação e suporte;
- *Gestão de Projecto:* Engloba a gestão de riscos, planeamento e o acompanhamento do projecto;
- *Gestão de Configuração e Mudanças:* Faz a gestão de documentos gerados ao longo do projecto, para que estes estejam sempre em conformidade com o que já foi desenvolvido;
- *Ambiente:* Define como o RUP foi configurado para ser utilizado no projecto, além da organização do ambiente de trabalho para toda a equipa de desenvolvimento.

A figura abaixo indica a relação existente entre as disciplinas do RUP e as suas fases.

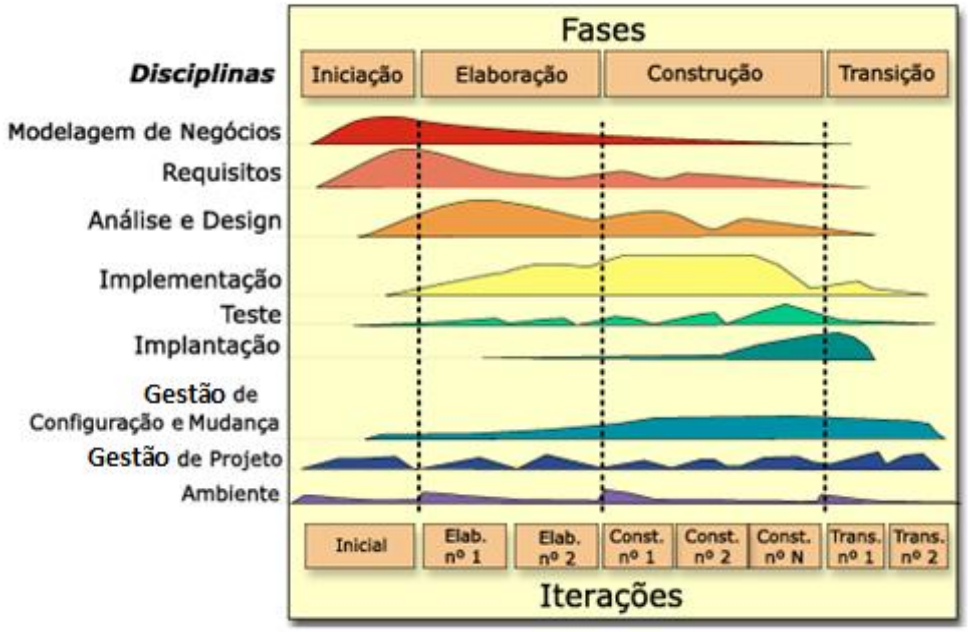



Figura 3- Relação entre as fases x disciplinas do RUP
Fonte: http://www.wthreex.com/rup/portugues/index.htm

### 2.3.2. CMMI - Capability Maturity Model Integration

Em 1987 apareceu um novo modelo como o auxílio no desenvolvimento de processo de software, no qual foi-lhe atribuído o nome de Capability Maturity Model (CMM) ou CMM Software. Mais tarde, em 1997 surgiu uma nova versão do modelo CMM, a CMMI (Capability Maturity Model Integration) em que a sua publicação foi feita três anos mais tarde (2000). Este modelo surgiu como uma forma de integrar os modelos existentes (CMM para software (SW-CMM), Electronic Industries Alliances's Systems Engineer Capability Model *(*EIA SECM*)* e o Desenvolvimento Integrado de Processo e Produto (IPD/CMM)). Por outras palavras o CMMI é o resultado da evolução de SW-CMM, do SECM e do IPD-CMM.

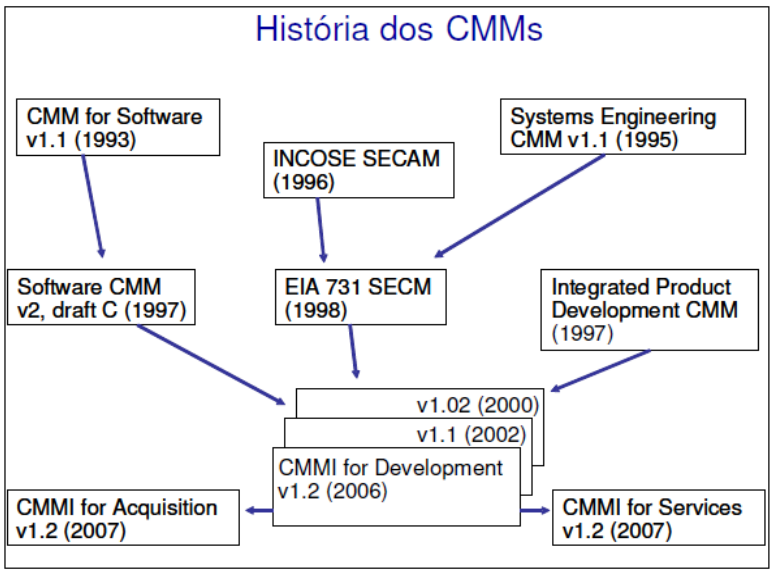



Figura 4 – História do CMMI
Fonte: http://www.sei.cmu.edu/library/assets/whitepapers/cmmi-dev_1-2_portuguese.pdf

CMMI é, um processo de melhoria de abordagem que ajuda as organizações a melhorarem o seu desempenho. CMMI pode ser utilizado para orientar a melhoria do processo através de um projecto, uma divisão, ou uma organização inteira (wikipedia).

O CMMI é um modelo de maturidade para melhoria de processo, destinado ao desenvolvimento de produtos e serviços, composto pelas melhores práticas associadas as actividades de desenvolvimento e de manutenção que cobre o ciclo de vida de um produto desde a sua concepção até a sua entrega e manutenção. Existem cinco níveis de maturidades incorporados dentro do CMMI.

Na figura abaixo são apresentados as características dos cinco níveis de maturidade.

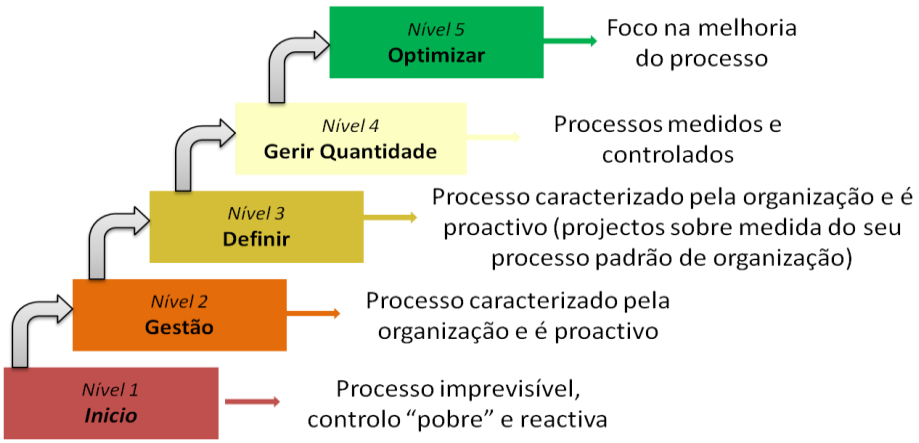



Figura 5 – Características dos níveis de maturidade
Fonte: http://software.gsfc.nasa.gov/docs/What%20is%20CMMI.ppt

O objectivo deste modelo é auxiliar as organizações na melhoria dos seus processos de desenvolvimento e manutenção de produtos e serviços.

Hoje em dia existem vários modelos de maturidade, padrões, metodologias e directrizes que são de grande auxílio para as organizações no melhoramento dos seus negócios. Mas, no que diz respeito aos problemas que as organizações vêm enfrentando, que é fazer a gestão eficaz dos seus activos, as abordagens disponíveis não focam sobre este problema e, este é um ponto crítico para o sucesso do negócio.

Esta metodologia oferece uma grande oportunidade de evitar ou eliminar essas barreiras por meio de modelos integrados que ultrapassam as disciplinas. O CMMI para desenvolvimento consiste nas melhores práticas relativas às actividades de desenvolvimento e manutenções aplicadas aos produtos e serviços.

Existem três versões do CMMI, em que a versão mais recente é o CMMI V1.3 que foi lançada em Novembro de 2010, nela existem 22 áreas de processos (APs). A versão V1.3 agrega três constelações:

- *CMMI para o desenvolvimento:* Aborda os processos de desenvolvimento de serviços e produtos;
- *CMMI para Aquisição:* Faz a gestão de cadeia de fornecimento, aquisição e processos de terceirização no governo e na indústria;
- *CMMI para serviços:* Aborda as directrizes para a prestação de serviços dentro de uma organização e para clientes externos.

### 2.3.3. Personal Software Process (PSP)

O CMM foi criado como um sistema de gestão, de apoio e de assistência aos grupos de engenheiros de software mas, houve uma altura em que se constatou que era necessário desenvolver um projecto que apoia-se individualmente um engenheiro de software. Watts S. Humphrey ao pesquisar, começou a aplicar e a estudar cada vez mais a fundo os princípios do SEI/CMM para poder desenvolver as práticas de software para um único engenheiro.

Humphrey, começou a desenvolver programas utilizando os princípios de CMM para desenvolver pequenos programas de modo a ajudar no controlo, gestão e na melhoria da forma pessoal (engenheiros de software) de desenvolvimento de software, possibilitando a

apropriação dos dados históricos para o atendimento de requisitos e, tornando-os elementos de rotina do trabalho, mais previsíveis e eficientes.

Mais tarde em 1993 Humphrey desenvolveu a Personal Software Process com o intuito de ajudar qualquer engenheiro a melhorar o seu trabalho em qualquer área técnica, utilizando os conceitos e métodos do PSP visto que este pode melhorar as estimativas e a habilidade de planeamento do projecto, assumir compromissos que podem surgir ao longo do projecto, gerir a qualidade do trabalho e reduzir o numero de falhas nos produtos.

PSP é um processo de desenvolvimento de software projectado para ser utilizado pelos engenheiros de software para a elaboração de projectos individuais (wikipedia).

Humphrey define o PSP como sendo um processo de auto melhoria projectado para ajudar o engenheiro de software a controlar, administrar e melhorar o modo como ele trabalha.

Hoje em dia muitas empresas que produzem software partilham um conjunto de necessidades (melhor custo e gestão de cronograma, melhor de qualidade de software e, reduzir o tempo de ciclo de desenvolvimento de software) que muitas vezes gera-se um produto final não desejado pelos clientes. O PSP responde directamente a todas estas necessidades, melhorando as práticas, técnicas e habilidades individuais dos engenheiros de software, fornecendo-lhes uma base quantitativa para a gestão de desenvolvimento dos processos.

O PSP é considerado uma ferramenta poderosa, pois pode ser utilizada de várias formas: administrar o trabalho, avaliar o próprio talento e construir habilidades.

**Objectivos do PSP**

- Ajudar as pessoas a serem melhores engenheiros de software;
- Estabelecer um mecanismo para melhorar, ao nível pessoal, a capacidade de planeamento, acompanhamento e a qualidade dos resultados;
- Conceitos básicos do PSP podem ser usados como ferramenta de uso geral para gerir as actividades pessoais particulares ou profissionais;
- *Benefícios:*
  - ✓ Melhoria da produtividade: melhor conhecimento e controlo dos mecanismos e do tempo de produção;

- ✓ Qualidade dos produtos: resultado do conhecimento das causas dos erros e do seu controlo estatístico.

***O PSP pode melhorar o negócio de desenvolvimento de software de uma organização de várias formas:***

- ❖ Os dados do PSP melhoram o planeamento e o acompanhamento de projectos de software;

  PSP ajuda os engenheiros e os seus gestores a aprenderem a praticar quantitativa de gestão de processos. Eles aprendem a usar processos definidos e recolher dados para gerir, controlar e melhorar o trabalho;
- ❖ Logo no início fazem a remoção de produtos com maior qualidade de defeitos.

***O projecto PSP é baseado no planeamento a seguir e nos seus princípios de qualidade:***

- ❖ Cada engenheiro é diferente, para ser mais eficaz, os engenheiros devem planear o seu trabalho e devem basear seus planos em seus próprios dados pessoais;
- ❖ Para produzir produtos de qualidade, os engenheiros devem-se sentir responsáveis pela qualidade dos seus produtos;
- ❖ É mais eficiente prevenir defeitos do que encontra-los e corrigi-los;
- ❖ O caminho certo é sempre o caminho mais rápido e barato para fazer um trabalho.

## Os níveis do processo do PSP

O processo do PSP é dividida em sete níveis, a figura abaixo indica os níveis do PSP.

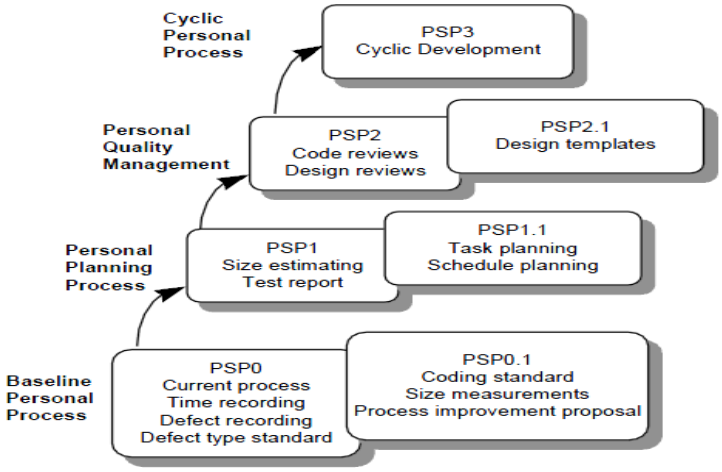



Figura 6 – Níveis do PSP
Fonte: http://www.ic.unicamp.br/~cortes/mc726/cap6.pdf

***PSP0***

Nesta primeira etapa é importante estabelecer uma base (*base-line*) de dimensão histórica. O principal objectivo deste nível é fazer o processo actual, controlo, registar o tempo e os defeitos/erros. Depois de ser concluído o primeiro nível, passa-se automaticamente para o nível seguinte, o ***PSP0.1*** que faz a codificação padrão, medição do tamanho do produto e a proposta de melhoria de processo (PIP- Process Improvement Proposal).

***PSP1***: *O processo de planeamento pessoal*

Neste nível (PSP1) faz-se a recolha dos dados inicias feito na primeira etapa (PSP0) e, com isso vai-se fazer o planeamento do PSP0. No início são feitos relatórios dos testes e as práticas da estimativa do tamanho e de recursos a utilizar. Logo a seguir no ***PSP1.1*** faz-se o planeamento das tarefas e planear a agenda.

***PSP2***: *Gestão de qualidade pessoal*

Neste nível faz-se a revisão do código e do desenho. O PSP2 introduz a técnica de inspecção e revisão para a detecção antecipada dos defeitos. O PSP2.1 aborda os modelos de design.

***PSP3***: *Ciclo de desenvolvimento*

Nesta ultima fase, introduz-se os conceitos para utilização do PSP nos grandes projectos, sem prejudicar a qualidade e a produtividade, inserindo limites de tamanho e uma estratégia de desenvolvimento cíclico no qual os programas de grandes portes são divididas em pequenas partes para o sua posterior integração e desenvolvimento.

### 2.3.4. Cascata

#### 2.3.4.1. Histórico

No ponto anterior (Metodologias tradicionais), fez-se uma pequena abordagem, especificando como era desenvolvida os softwares antes da década de 70, neste ponto vai-se aperceber de uma forma mais profunda as causas do modelo em cascata.

Antes da década de 70, a construção de um software era tida como um "bicho-de-sete-cabeças", pois os desenvolvedores de software (os engenheiros), na maioria dos casos tinham receio de que o produto final não corresponde-se às espectativas do cliente. Na maioria das vezes o produto final não era de boa qualidade, não tinha qualquer estrutura, não era planeado, era desorganizada, o produto era entregue fora do prazo, ou seja, quase 70% dos produtos desenvolvidos não correspondiam as verdadeiras necessidades dos utilizadores, muitos clientes ficavam insatisfeitos com o produto final visto que empregavam uma boa parte dos seus recursos no desenvolvimento desses produtos e, com isso os gastos aumentavam cada vez mais, podendo afirmar-se que custo / benefício nao correspondiam ao desejado. Através destes factores essa época ficou conhecida com a época da crise de software (Pressman 02). A crise de software contribui cada vez mais para o aparecimento do modelo em cascata.

Em 1970 Royce apresentou o primeiro modelo de desenvolvimento de software, em que este passou a dar um novo rumo no que dizia respeito ao desenvolvimento de software: o processo de desenvolvimento de software passou a ser estruturado, organizado, planeado, etc.

O modelo em cascata é um framework para o sequenciamento das actividades associadas ao desenvolvimento de software (Royce, 1970).

O modelo original proposto pelo Royce era o "ciclo de retroalimentação". Quando houver uma interacção entre as etapas posteriores e os anteriores, esta é uma das

circunstâncias ideais para confirmar que naquele momento ocorre o ciclo de retroalimentação. Ou seja, em qualquer ponto no processo de concepção após a conclusão da análise de requisitos, existe uma base close-up que retorna a etapa anterior, caso de houver imprevistos ou dificuldades de concepção.

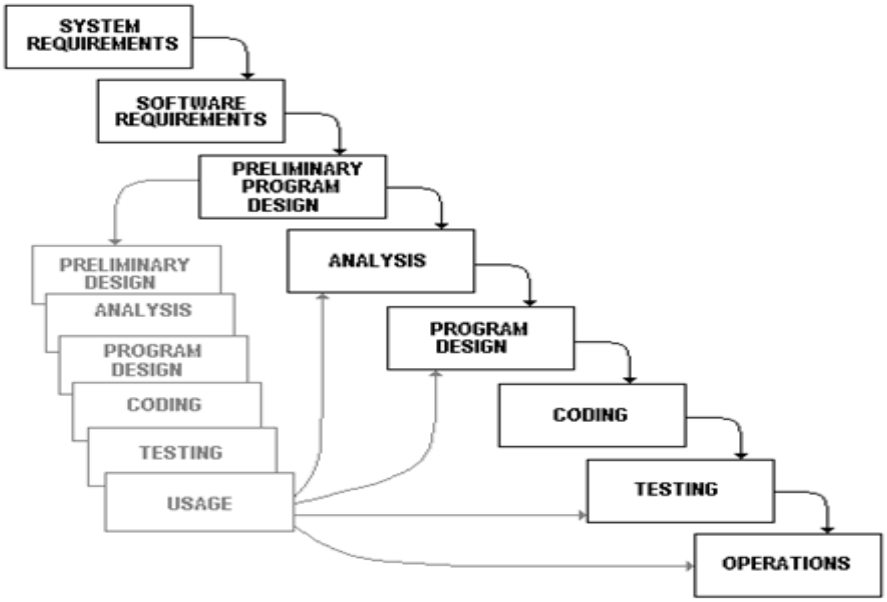



Figura 7 – Modelo em cascata original apresentado por Royce (1970)
Fonte: http://www.informatik.uni-bremen.de/gdpa/def/def_w/WATERFALL.htm

O Modelo em Cascata trouxe contribuições importantes para o processo de desenvolvimento de software:

- Imposição de disciplina, planeamento e gestão;
- A implementação do produto deve ser feita até que os objectivos tenham sido completamente entendidos;
- Permite gestão da base-line, que identifica um conjunto fixo de documentos produzidos ao longo do processo de desenvolvimento.

O modelo em cascata até aos meados dos anos 80 foi o único modelo com uma grande aceitação por partes dos desenvolvedores.

Este modelo prepõe uma abordagem sistemática, linear e sequencial no que diz respeito ao desenvolvimento de software.

Muitos autores e/ou pesquisadores deram as suas contribuições para o aperfeiçoamento deste modelo e com isso alguns acabaram por definir o modelo consoante o

seu ponto de vista. Mas em todas as definições nota-se que há uma característica comum entre elas: o fluxo linear e sequencial das actividades.

- O modelo em cascata, também conhecido com ciclo de vida clássico é classificado como um modelo linear sequencial sugere uma sequência sistemática para o desenvolvimento de software (PRESSMAN, 2001);
- O modelo em cascata é um modelo de desenvolvimento de software sequencial no qual o desenvolvimento é visto como um fluir constante para frente (como uma cascata) através das fases de análise de requisitos, projecto, implementação, testes (validação), integração, e manutenção de software (wikipedia).

A figura abaixo indica o desenho das sete actividades do ciclo de vida do modelo em cascata que actualmente é a mais utilizada.

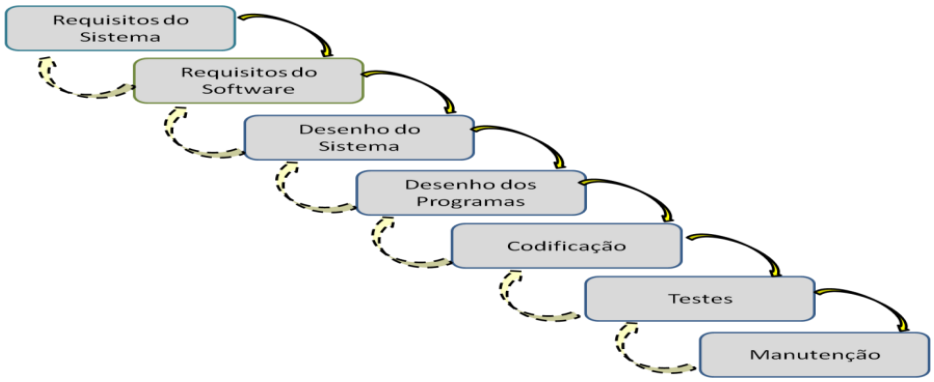



Figura 8 – Modelo do processo em cascata
Fonte: Adaptado por Miguel (Gestão de projectos, 2003)

- A fase de definição de requisitos é uma das fases mais importante para se obter um bom produto final, porque nela todas as eventuais exigências do sistema a ser desenvolvido são capturadas quando se define todos os requisitos do sistema. Nesta fase normalmente os requisitos são divididos em duas partes:
  - ✓ Requisitos do sistema: nesta subfase são recolhidas todas as informações empresariais, em que com isso o trabalho inicial consiste em estabelecer todas as necessidades e os requisitos globais que contem as informações da organização;
  - ✓ Requisitos do software: esta subfase retracta e foca somente no software a construir. Nesta etapa todos os serviços, restrições e objectivos do sistema são definidos por meio de consulta aos utilizadores finais. Por isso é necessário que

os engenheiros do sistema compreendam bem os domínios aplicacionais nas suas várias vertentes (informações, funções, desempenho, interfaces, etc.);

- ❖ Desenho do sistema: esta fase foca-se no desenho lógico do produto, em que visa modelizar uma solução de um modo independente da tecnologia;
- ❖ Desenho dos programas: ela é designada como desenho físico, visa modelar a solução tecnológica para o problema e é um processo com múltiplos passos que se concentra em quatro atributos de um programa de software:
  - ✓ Estrutura dos dados;
  - ✓ Arquitectura do software;
  - ✓ Interfaces;
  - ✓ Algoritmos.
- ❖ Codificação: após ser feito o desenho físico, este tem de ser posto em prática, por isso nesta fase o desenho físico vai ser traduzido em linguagem própria da máquina;
- ❖ Testes: após a codificação, são realizados os testes. Os testes concentram-se na lógica interna do software e nas funções externas. Estes testes são realizados para garantir que os requisitos dos softwares foram atendidos. Após estes testes o software é entregue ao cliente;
- ❖ Manutenção: após o produto ser entregue, instalado e colocado em operação para que possa ser utilizado, são detectados alguns erros que não foram encontrados ou que passaram despercebidos durante a fase de teste, também pode haver algumas alterações que o cliente pretende fazer no produto (alteração do ambiente de negocio, etc.), esses factores (erros e alterações) são efectuadas no âmbito do suporte e manutenção ao sistema após a instalação. Geralmente esta fase é tida como a fase mais longa do ciclo de vida.

Cada fase concluída gera um marco, ou seja, gera-se um documento, protótipo do software ou mesmo uma versão do sistema para cada uma dessas fases do ciclo de vida deste modelo. É neste facto que se baseia a grande parte do sucesso do modelo em cascata.

### 2.3.4.2. Abordagem

O modelo em cascata é uma referência para muitos outros modelos, pois ele serve como base para muitos projectos modernos. Ele é um dos modelos de ciclo de vida mais citado actualmente, é tido como modelo de ciclo de vida clássico. Os outros modelos que fazem parte do ciclo de vida serão abordados neste ponto. Todos estes modelos de ciclo de vida seguem o mesmo padrão de desenvolvimento, ou seja, todos eles tiveram como referencia o modelo em cascata mas, cada um segue os seus próprios princípios e as suas próprias regras.

***Prototipagem***

O protótipo é considerado como o "primeiro sistema", ou seja, desenvolve-se um modelo de software que vai ser implementado. Quando se desenvolve um protótipo, este tem tendência de tornar o desenvolvimento de software num processo mais rápido e sem muitas complicações, visto que facilita a definição dos objectivos do software, retracta a interacção entre o utilizador e o software e, é útil para a verificação e validação em diferentes etapas do processo de desenvolvimento de software.

Quando um cliente não tem uma noção clara ou detalhadamente do produto, o primeiro passo a dar é desenvolver um protótipo, ou seja, quando um cliente descreve o produto que ele quer mas, ao mesmo tempo não indica detalhadamente quais os requisitos de entrada, processamento ou saída, a prototipagem é a melhor solução nesses casos.

*Vantagens da prototipagem:*

- Aumento da visibilidade de um sistema por parte dos clientes e dos desenvolvedores;
- Aumento da compreensão do software;
- Proporciona a execução de mudanças que por sua vez ficam menos dispendiosas, visto que o processo não está finalizado.

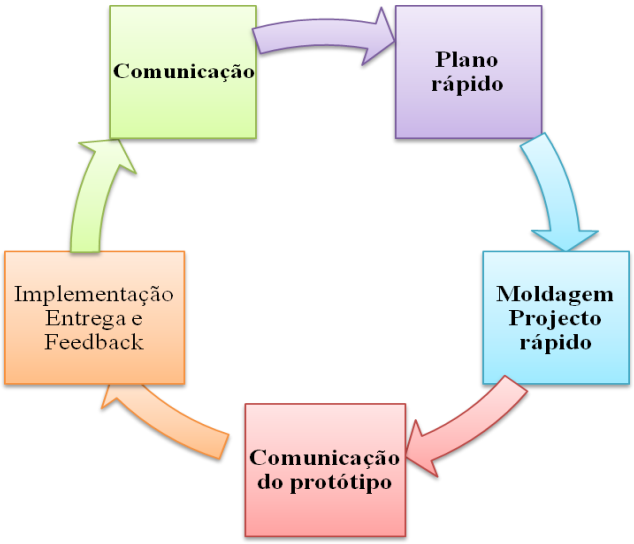



Figura 9 – Modelo de prototipagem
Fonte: Adaptado por Pressman 2000

***Modelo Espiral***

O modelo em cascata foi durante muitos anos a base pura do desenvolvimento de software mas, em 1988 um senhor chamado Barry Boehm (BOE, 88) apresentou um novo modelo de desenvolvimento de software no qual deu o nome de Modelo Espiral. Este modelo é evolucionário, o seu processo de software combina a natureza iterativa de prototipagem com os aspectos controlados e sistematizados no modelo em cascata.

Boehm (BOE, 01) descreveu o modelo espiral de desenvolvimento como um gerador de modelo de processo guiado por risco, usado para guiar a engenharia de sistemas intensivos em software com vários interessados concorrentes. Ele tem duas principais características distintas: a *primeira* é uma abordagem cíclica, para aumentar incrementalmente o grau de definição e implementação de um sistema enquanto diminui seu grau risco. A *segunda* é um conjunto de marcos de ancoragem, para garantir o comprometimento dos interessados com soluções exequíveis e mutuamente satisfatórias para o sistema.

Utiliza-se muito este modelo no desenvolvimento de softwares em conjunto com o paradigma da orientação a objectos, onde o desenvolvimento em módulos, adicionado ao processo de integração, se encaixa nos conceitos do paradigma (Pressman 2002).

Normalmente este modelo pode ser dividido em 3 ou 6 regiões. Pressman apresentou o modelo dividido em 6 regiões:

- *Análise de riscos:* análise de alternativas e identificação/resolução de riscos;
- *Engenharia:* desenvolvimento do produto (criação do produto);
- *Construção e adaptação:* responsável pela implementação, teste, instalação e a avaliação realizada pelo cliente;
- *Avaliação do cliente:* avaliação do produto e planeamento das novas fases;
- *Planeamento:* determinação dos objectivos, alternativas e restrições;
- *Comunicação com o cliente:* a fim de manter contacto com o cliente, adquirindo informações úteis ao sistema.

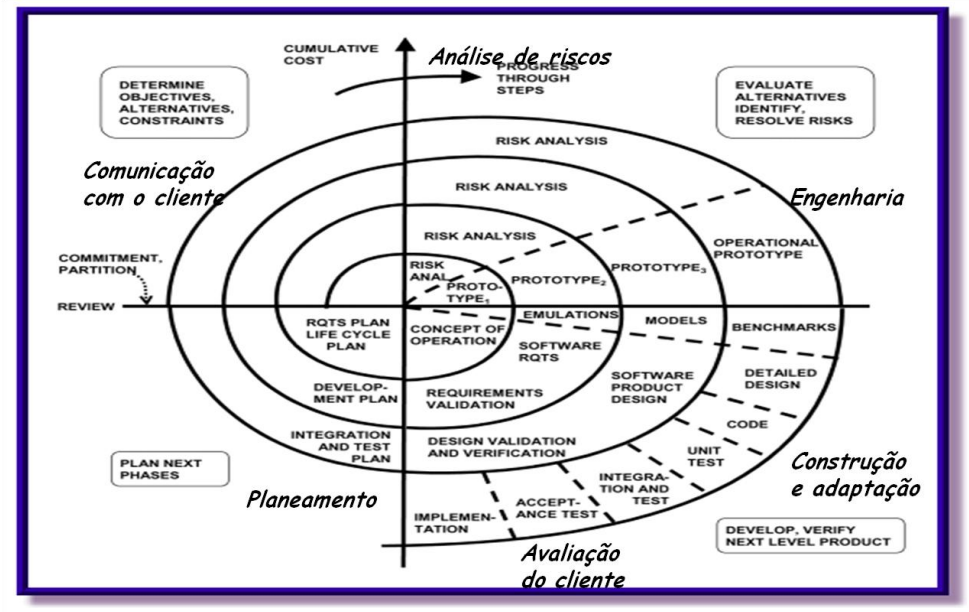



Figura 10 – Modelo espiral[1]

O modelo pode ser adaptado conforme as características do projecto e, com isso varia também o número de tarefas por regiões.

A principal diferença entre o modelo espiral e os outros modelos do processo de software é o reconhecimento explícito de riscos. Por exemplo: os programadores

[1] Fonte: http://www.google.pt/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CFkQFjAE&url=http%3A%2F%2Fw3.ualg.pt%2F~rnascimento%2FaulasES-2006%2Faula6.ppt&ei=TRwHUJW7C4iQiQfgruSwCA&usg=AFQjCNFHD84V1VGdrcbOLjvW7CAhauAwkQ&sig2=uQPAyJPgk7nYH8xDRyR7kg

(engenheiros) "diariamente" estão a lidar com novas linguagens de programação e, se eles tiverem a intenção de utilizar uma dessas linguagens, normalmente correm o risco de não produzirem código - objecto suficientemente eficaz ou que os compiladores disponíveis não sejam confiáveis (Sommerville, 2007).

Os riscos são um dos factores que podem causar vários problemas aos projectos em desenvolvimento. Por isso a gestão de risco é uma das actividades muito importante.

### ***Modelo incremental***

O modelo incremental surgiu através de uma combinação entre o modelo linear e a prototipagem (Mazola, 2001). Este modelo de ciclo de vida é similar ao modelo em cascata.

Todo o desenvolvimento feito com este modelo é dividido em etapas e, elas são denominadas por "incrementos", ou seja, essas etapas por sua vez reproduzirá incrementalmente o sistema até a sua versão final.

Segundo Pressman (2006), o modelo incremental combina elementos do modelo em cascata, sendo estes aplicados de maneira interactiva.

Este modelo aplica sequências lineares de uma forma racional à medida que o tempo passa visto que cada sequência linear produz "incrementos" do software passíveis de serem entregues.

Em cada incremento é realizado todo o ciclo de desenvolvimento de software, do planeamento aos testes do sistema já em funcionamento. Quando se gera o primeiro incremento dá-se o nome de núcleo de produto, sendo que nesta primeira etapa os requisitos básicos são satisfeitos ou realizados mas, muitas características suplementares (algumas conhecidas, outras desconhecidas) deixam de ser efectuados (elaborados).

O modelo incremental diferencia-se da prototipagem da seguinte forma: ela tem como objectivo a apresentar um produto operacional a cada incremento. O produto final a ser entregue é, a versão simplificada do primeiro incremento, oferecendo capacidades que servem aos utilizadores, além de uma plataforma para avaliação dos utilizadores.

Normalmente este modelo é muito útil e indispensável quando, não houver mão-de-obra disponível dentro de uma empresa, para se fazer uma implementação completa, dentro do prazo estabelecido para entrega do produto.

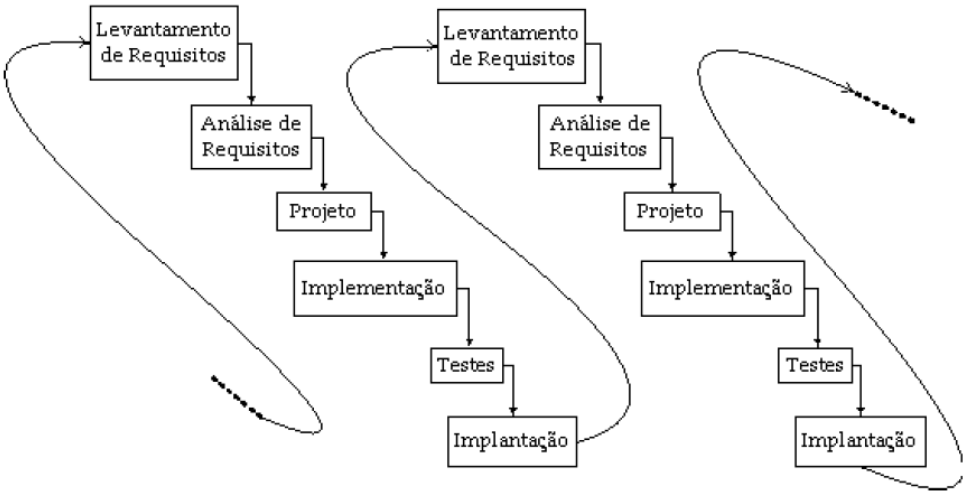



Figura 11 – Modelo do processo incremental
Fonte: http://www.devmedia.com.br/1-introducao-ao-desenvolvimento-agil/5916

### *2.3.4.3. Princípios do modelo em cascata*

O modelo em cascata tem alguns princípios básicos como:

- O projecto é dividido em fases sequenciais com alguma sobreposição e splashback aceitável entre as fases de desenvolvimento;
- O modelo do produto é definido e desenhado no início do projecto e "não muda". Além disso, o tempo e os custos dos projectos também são definidos desde o princípio;
- A codificação é feita sobre as especificações técnicas e funcionais escritas nas fases anteriores do projecto;
- É mantido um controlo apertado ao longo da vida do projecto através de extensa documentação escrita, revisões formais e de aprovação / assinatura do utilizador e a gestão de tecnologia de informação que ocorre na maioria das fases finais, antes de começar a próxima fase.

#### 2.3.4.4. Características

Este modelo apresenta várias características no processo de desenvolvimento de software, que são:

- Divisão inflexível do projecto em estágios distintos. A fase seguinte só deve iniciar quando a anterior tiver sido concluída e aprovada pelas partes envolvidas. Exemplo: o design apenas deve começar quando os requisitos estiverem totalmente definidos e aprovados;
- O modelo em cascata é apropriado quando se tem um entendimento claro dos requisitos;
- Oferece maior previsibilidade de prazos e custo: melhor planeamento e gestão;
- Dificuldades em realizar mudanças com o processo em andamento – requisitos sempre mudam;
- Seu enfoque está nos documentos e artefactos (requisitos, projectos, códigos);
- É um modelo amplamente usado para aengenharia de software;
- Abordagem sequencial e sistemática.

#### 2.3.4.5. Qualidade do modelo em cascata

O modelo em cascata ao ser implementado, introduziu algumas qualidades consideradas importantes para o processo de desenvolvimento de software:

- Chama a tenção de que o processo de desenvolvimento deve ser conduzido de uma forma disciplinada, com actividades claramente definidas, determinado a partir de um planeamento e sujeito a uma gestão durante a realização de todo o projecto;
- Define de maneira clara quais são estas actividades e quais os requisitos para desempenha-las. Por fim, o modelo introduz a separação das actividades de definição e design da actividade de programação que era o centro das atenções no desenvolvimento de software.

### 2.3.4.6. Problemas com modelo em cascata

Pressman (1995) aponta alguns problemas identificados neste ciclo de vida:

- Os projectos reais raramente seguem o fluxo sequencial que o modelo propõe. Alguma interacção sempre ocorre e traz problemas na aplicação do paradigma;
- Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente. O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no inicio de muitos projectos;
- O cliente deve ter paciência. Uma versão de trabalho dos programas não estará disponível até um ponto final do cronograma do projecto. Um erro crasso, se não for detectado até que o programa de trabalho seja revisto, pode causar muitos problemas.

### 2.3.4.8. Criticas ao modelo

Apesar de ser o mais utilizado na engenharia de software, este modelo é alvo de muitas críticas:

- Não reconhece explicitamente a necessidade de *revisão*, isto é, retornar a fase anterior e refazer as coisas, à luz da informação obtida durante o desenvolvimento:
  - ✓ Informação acerca daquilo que o utilizador quer, como o desenho se deve comportar, como o Sistema de Gestão de Bases de Dados (SGBD) deve ser usado, etc.
- Durante o desenvolvimento, têm de ser tomadas muitas decisões, que podem conduzir a um certo número de caminhos e resultados possíveis. O modelo simples não reconhece isto. Por exemplo, dependendo da análise de requisitos do cliente, pode-se decidir por uma das três alternativas:
  - ✓ Desenvolver o sistema de raiz;
  - ✓ Refazer um sistema existente;
  - ✓ Adquirir um pacote de software que forneça a maioria das funcionalidades exigidas.

Esta simples decisão pode ter três resultados substancialmente diferentes.

- É muitas vezes difícil o cliente/utilizador expressar, de forma explícita, todos os seus requisitos. O modelo em cascata, por ser sequencial e linear, mas tem dificuldade em acomodar a incerteza existente no início da maioria dos projectos;
- O utilizador tem de esperar, por vezes bastante tempo, até lhe ser entregue uma versão operacional do sistema de software. O actual ritmo de mudança de negócios não é compatível com esse compasso de espera (quando o sistema é disponibilizado, existe um sério risco de os requisitos iniciais se terem alterado substancialmente).

#### 2.3.4.8. Conclusões

*Vantagens*

As vantagens do modelo em cascata fazem com que este seja o mais utilizado. Assim, e no seguimento do que se menciona, abaixo apresenta-se exemplos das respectivas vantagens:

- Uma grande vantagem do modelo em cascata é que, a documentação é produzida em cada etapa do desenvolvimento do modelo. Por conseguinte, origina que a compreensão da concepção do produto seja mais simplificada;
- Fixa pontos específicos para a escrita de relatórios;
- É um modelo linear e claro, os modelos lineares são os mais simples de ser implementado;
- Após cada etapa importante da codificação do software, é feito o teste para a verificação dos códigos (a execução correcta dos códigos);
- Todas as actividades identificadas nas fases deste modelo são fundamentais e estão na ordem correcta;
- Esta abordagem é actualmente a norma e provavelmente permanecerá como tal nos próximos tempos.

*Desvantagens*

- Ironicamente, a maior desvantagem do modelo cascata é uma das suas maiores vantagens:
  - ✓ Não se pode voltar atrás, se a fase de desenho não for efectuada correctamente, ou seja, será problemática a fase de implementação se existirem erros de sistema;
- Muitas vezes acontece que, o cliente não tem definido o âmbito do seu projecto, não sabe de uma forma clara o que exactamente pretende com o software, não sabe quais as funcionalidades que o mesmo deve executar e, isso gera muita confusão/problemas no processo de desenvolvimento de software. Esses problemas também ocorrem após a conclusão do software, o Cliente precisa ou quer fazer algumas alterações no software (mudanças de negocio, etc.)
- Não prevê a adjudicação de um contrato de manutenção;
- Não permite a reutilização;
- É excessivamente sincronizado;
- Se ocorrer um atraso todo o processo é afectado.

## *2.4. Falhas no desenvolvimento tradicional*

A grande falha no desenvolvimento tradicional é que ele se baseia em premissas, que na realidade não se aplicam no trabalho ao desenvolver o software.

Segundo Drucker (DRUCKER, 1999), existem duas categorias de trabalhadores: o ***trabalhador manual*** e o ***trabalhador do conhecimento***. Um trabalhador manual executa actividades que dependem basicamente das habilidades manuais e que não se baseia no uso intensivo do conhecimento. São actividades relativamente fáceis de automatizar por serem altamente determinísticas e repetitivas. O trabalhador do conhecimento é aquele que produz com base no uso intensivo do conhecimento e da criatividade (Teles, 2004).

O desenvolvimento tradicional basicamente é conhecido como um trabalho que é executado por trabalhadores do conhecimento, mas, utiliza premissas que só são válidas para o trabalhador manual.

Vários autores como: Drucker, DeMarco, Lister, Toffler e DeMarsi através dos seus estudos, têm demonstrado que a produtividade do trabalho do conhecimento deriva de factores completamente diferentes daqueles usados para o desenvolvimento tradicional. De facto, eles mostram que as premissas da produtividade do trabalho do conhecimento são praticamente opostas as do trabalho manual e, o grande erro cometido na maioria dos projectos de software, é desconsiderar este facto e adoptar as mesmas práticas do ambiente industrial.

Um exemplo da grande diferença em relação ao trabalhador manual e do trabalhador do conhecimento está na escolha da melhor política para tratar a ocorrência de erros. Para actividades que envolvem trabalhadores manuais, os erros podem e devem sempre ser evitados, e os processos rígidos e determinísticos ajudam a alcançar este objectivo. Já para as actividades que envolvem trabalhadores do conhecimento, os erros devem ser vistos como acontecimentos naturais, saudáveis e até inevitáveis.

## 2.5. Metodologias Ágeis - Enquadramento

Como foi dito no capítulo anterior na década de 70 houve muitos fracassos em relação ao desenvolvimento de software, com isso foi desenvolvido a primeira metodologia de desenvolvimento de software (modelo em Cascata) para que o software fosse desenvolvido de uma forma estruturada e organizada de maneira que não houvesse fracassos. Mas, logo depois foram surgindo outros tipos de metodologias no qual lhes foram atribuídas o nome de metodologias tradicionais. Todas essas metodologias foram desenvolvidas para combater os fracassos que eram frequentes no desenvolvimento do software.

Apesar de terem apresentados várias metodologias (tradicionais) para combater os fracassos, mesmo assim continuou-se a verificar que o problema não estava resolvido por completo, visto que alguns fracassos ainda estavam presentes e, a partir dali surgiu a necessidade de desenvolver uma nova metodologia que combatesse esses fracassos, em que esta seria como uma alternativa as metodologias tradicionais. A partir daquele momento, vários metodologistas começaram a desenvolver novas metodologias de acordo com o trabalho que estavam a desenvolver. A essas metodologias deram o nome de Metodologias Ágeis.

As metodologias ágeis surgiram na década de 90, mas, só tiveram o seu verdadeiro impacto em diversos segmentos da indústria de software nos últimos anos, principalmente em 2001 quando surgiram os princípios do manifesto ágil.

A agilidade é a habilidade de criar e responder às mudanças com respeito ao resultado financeiro do projecto num ambiente de negócios turbulento.

Segundo Highsmith, a agilidade é a habilidade de equilibrar a flexibilidade com a estabilidade (Highsmith, 2002).

Actualmente a agilidade tornou-se numa palavra mágica quando se descreve um processo moderno de software. Tudo é ágil. Uma equipa ágil é uma equipa capaz de responder adequadamente as modificações (Ivar Jacobson). O acolhimento das modificações é o principal guia para a agilidade. Os engenheiros de software devem reagir rapidamente se tiverem de acomodar as rápidas modificações (Jac, 2002). Todo o processo de software pode utilizar a agilidade.

Larman (2002) afirmou que um processo ágil implica um processo adaptativo e leve, que responde facilmente a mudanças.

Por essas e por outras características que as metodologias ágeis são tidas como metodologias leves.

Segundo Scott W. Amber, metodologia ágil é uma metodologia baseada na prática para modelagem eficaz de software.

Fowler (2001) define as metodologias ágeis como uma reacção as metodologias tradicionais e conceituais.

As metodologias ágeis estão estruturadas de modo a atender a natureza mutável e dinâmica do processo de concepção do sistema, ou seja, esta metodologia foi desenvolvida para atender qualquer tipo de mudanças ao longo do ciclo de vida de um sistema.

Elas têm como objectivo reduzir o ciclo de vida do software, mas, acelerando o seu desenvolvimento em pequenas versões (versões mínimas).

Essas metodologias adoptaram duas características das metodologias tradicionais, em que as utiliza para que os projectos sejam conduzidos de forma adaptativa: adoptando um processo espiral e iterativo de forma que no final, cada uma das iterações entregue um produto pronto a ser utilizado e completo. Com essas iterações (curtas), o cliente e o utilizador podem a ter contacto com o produto (software) logo que a primeira versão deste esteja finalizada e colocada em funcionamento e, é nesta etapa que é decidida tanto pelos clientes como pelos desenvolvedores, quais as características que deverão ser implementadas e quais é que serão retiradas do sistema. Isso faz com que o sistema seja desenvolvido da forma mais iterativa possível.

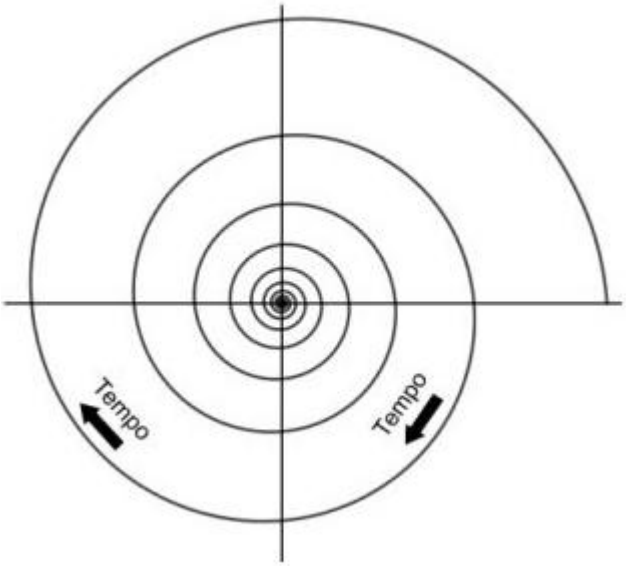



Figura 12 - Desenvolvimento iterativo em espiral
Fonte: http://devagil.wordpress.com/2007/07/07/introducao-ao-desenvolvimento-agil/

Dentro das metodologias ágeis existem vários métodos de aplicações para o desenvolvimento de software, mas só serão abordados alguns desses métodos nesse trabalho: XP, ASD, FDD, LEAN, DSDM, CRYSTAL e SCRUM.

Vários autores e pesquisadores deram as suas opiniões sobre essas metodologias, mas um autor chamado Scott W. Ambler (2008) resolveu fazer uma pesquisa onde fez várias perguntas para os desenvolvedores que ainda não tinham utilizado as metodologias ágeis, colocando a seguinte questão: “Quando é que eles iam começar a utilizar as metodologias ágeis?” e, ele chegou à seguinte conclusão conforme nos indica o gráfico abaixo (ambysoft).

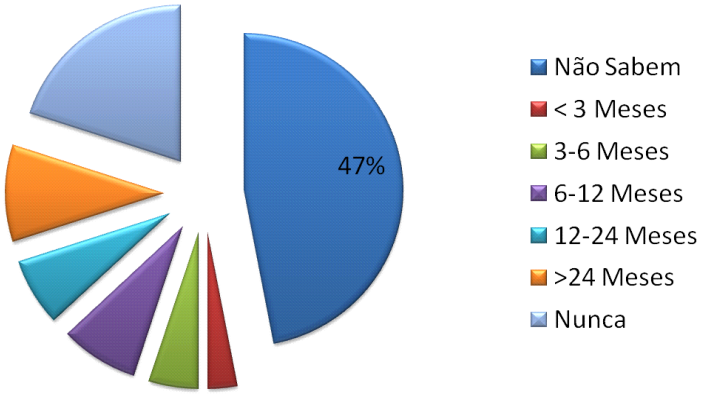



Figura 13 – "A utilização das metodologias ágeis pelos engenheiros de software."
Fonte: Scott Ambler (2008)

Em 2010 a empresa Versionone, patrocinou uma pesquisa onde procurou saber qual o método mais utilizado dentro das metodologias ágeis e chegou a seguinte conclusão:

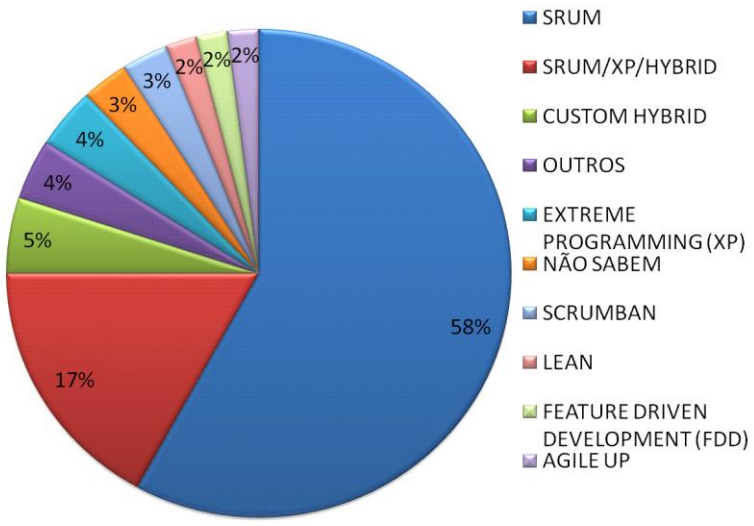



Figura 14 – Métodos ágeis mais utilizados
Fonte: Versionone (2010)

De acordo com a figura apresentada acima, podemos verificar que os métodos mais utilizados dentro das metodologias ágeis são SCRUM e XP.

### 2.5.1. Manifesto Ágil – M.A

O Manifesto Ágil (M.A) é uma declaração de princípios que fundamentam o desenvolvimento ágil de software (wikipedia).

O M.A surgiu durante uma conferência realizada em 2001 onde participaram 17 metodologistas. Esta conferência teve como objectivo principal a discussão e a comparação dos diferentes resultados que cada um tinha obtido com os seus projectos, tentando desta forma criar uma metodologia que unificasse todas as práticas que estavam a ser discutidas entre eles e, com isso criou-se um conjunto de princípios e valores que deverão ser seguidos para se ter um bom desenvolvimento ágil, no qual deram o nome de Manifesto Ágil.

Os metodologistas antes de apresentarem os princípios ágeis, definiram os valores fundamentais do manifesto:

- Os indivíduos e as suas interacções estão acima dos procedimentos e ferramentas;
- O funcionamento do software acima da documentação abrangente;
- A colaboração com o cliente é mais do que a negociação de contractos;
- A capacidade de resposta a mudanças acima de um plano pré-estabelecido.

**Os 12 princípios do Manifesto Ágil:**

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor;
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam as mudanças, para que o cliente possa tirar vantagens competitivas;
3. Entregar o software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos;
4. Pessoas relacionadas aos negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projecto;
5. Construir projectos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seus trabalhos;
6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara;

7. Software funcional é a medida primária de progresso;
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e utilizadores, devem ser capazes de manter indefinidamente, passos constantes;
9. Atenção contínua a uma técnica excelente e bom design aumentam a agilidade;
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser efectuado;
11. As melhores arquitecturas, requisitos e designs surgem de times auto-organizáveis;
12. Em intervalos regulares, o time reflecte em como ficar mais efectivo, então, se ajustam e optimizam de acordo com o seu comportamento.

Os princípios do manifesto ágil estão presentes em qualquer uma das metodologias ágeis, definindo práticas a serem seguidas para que o desenvolvimento do sistema torne num processo mais eficaz.

### 2.5.2. Extreme Programming – XP

Com o aparecimento das metodologias ágeis, durante alguns anos o XP foi o que obteve mais atenção por parte dos desenvolvedores (engenheiros).

Este método surgiu na década de 90, mas só foi publicado em 1999 quando o autor da metodologia (Kent Beck) foi convidado para trabalhar no projecto C3, onde acabou por aplicar este método no projecto. Ele deu o nome XP a este método porque a abordagem foi desenvolvida pelo avanço reconhecida como uma boa pratica, tal como o desenvolvimento iterativo e o envolvimento do cliente em níveis “extremos” (Beck, 2000). A C3 (da Chrysler) foi o primeiro projecto a ser desenvolvido com o XP.

O objectivo principal do XP é ajudar a criar sistemas de melhor qualidade que deverá ser produzido em menor tempo e de uma forma económica. Segundo Teles, os objectivos são alcançados através de um pequeno conjunto de valores, princípios e práticas, que diferem substancialmente da forma tradicional de se desenvolver um software (Teles 2008). Beck definiu o XP como um método ágil para equipas pequenas e médias que desenvolvem

softwares baseados em requisitos vagos e em constante mudança (Bec,99). Mas há quem afirma que o XP pode ser utilizado em qualquer tipo de empresas (todo o tamanho) e indústria do mundo inteiro (Don Wells).

Segundo Fowler, o XP é o método ágil mais popular actualmente, a sua utilização visa responder aos sistemas onde as mudanças de requisitos são constantes e, com base em seus valores e práticas busca fazer isso da forma mais simples e eficiente (Fowler, 2000).

A base do XP é composta por quatro partes: valores, princípios, actividades e práticas.

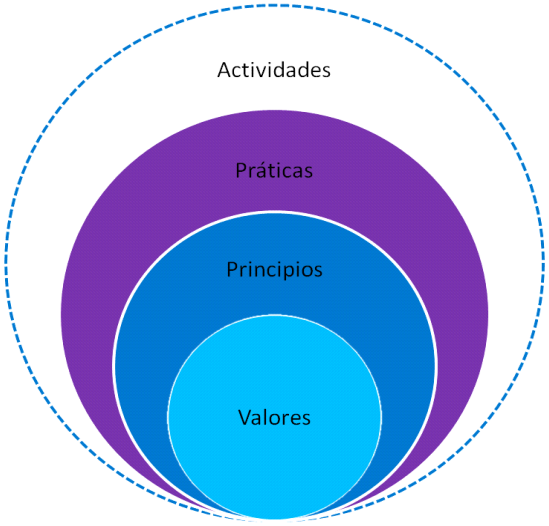



Figura 15 – Base do XP (Beck e Andres, 2004)
Fonte: Chagas (2008)

*Existem seis valores dentro do XP que o diferencia das outras metodologias:*

- Feedback constante;
- Abordagem incremental;
- A comunicação entre as pessoas é encorajada;
- Simplicidade;
- Respeito;
- Coragem.

O XP segue um conjunto de princípios ou práticas que se apoiam uma nas outras para que haja sucesso no processo de desenvolvimento de um projecto. A tabela abaixo indica as práticas ou princípios segundo Sommerville (Sommerville, 2007).

| Princípio ou prática | Descrição |
|---|---|
| Planeamento incremental | Os requisitos são registrados em cartões de histórias a serem incluidas em um release são determinadas pelo tempo disponivel e a sua prioridade relativa. Os desenvolvidores dividem essas histórias em 'tarefas'. |
| Pequenos releases | O conjunto mínimo útil de funcionalidade que agrega valor ao negócio é desenvolvido primeiro.<br>Releases do sistema são drequentes e adicionam duncionalidades incrementalmente ao primeiro release. |
| Projectos simples | É realizado um projecto suficiente para atender aos riquisitos actuais e nada mais. |
| Desenvolviemnto test-first | Um framework automatizado do teste unitário é usado para escrever os testes para uma nova parte da funcionalidade antes que seja implementada. |
| Refactoring | Espera-se que todos os desenvolvedores recriam o código continuamente tão logo, os aprimoramentos do código foram encontrados. Isso torna o código simples e facil de manter. |
| Programação em pares | Os desenvolvedores trabalham em pares, verificando o trabalho do outro e fornecendo apoio para realizar sempre um bom trabalho. |
| Propriedade colectiva | Os pares de desenvolvedores trabalham em todas as áreas do sistema, de tal modo que não se formem ilhas de conhecimento, com todos os desenvolvedores de posse de todo o código.<br>Qualquer um pode mudar qualquer coisa. |
| Integração continua | Tão logo o trabalho em uma tarefa seja concluída, este é integrado ao sistema como um todo. Depois de qualquer integração, todos os testes unitários do sistema devem ser realizados. |
| Ritmo sustentável | Grandes quantidades de horas extras não são consideradas aceitáveis, pois, no médio prázo, há uma redução na qualidade do código e na produtividade. |
| Cliente on-site | Um representante do utilizador final do sistema (o cliente) deve estar disponível em tempo integral para apoiar a equipa de XP. No processo da extreme programming, o cliente é um membro da equipa de desenvolvimento e é responsável por trazer os requisitos do sistema à equipa para implementação. |

Figura 16 – Princípios ou práticas da XP
Fonte: Sommerville (2007)

*As actividades do XP estão divididas em quatro vertentes:*

- Escutar – o XP baseia-se na comunicação entre os membros da equipa. Para que a comunicação chegue a todos é necessário saber escutar;
- Testar – Teste não é algo que se faz antes de entregar a solução final ao cliente, mas sim um passo de integração dentro de todo o processo;
- Escrever código – o XP está assente na codificação através de práticas, como programação em pares, refactoring e revisão de código;
- Design – Uma das ideias mais radicais do XP é a de que o design evolui e aumenta durante o projecto. O design não é estático nem atribuído a um só elemento, mas sim baseado em toda a equipa.

No XP, todos os requisitos são expressos como cenários (chamados históricos do utilizador) que são implementados directamente como uma serie de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada tarefa antes da escrita do código. Todos

os testes devem ser executados com sucesso quando um novo código é integrado no sistema (Sommerville, 2007).

Normalmente os desenvolvedores que trabalham com o XP têm preferência pelos projectos que iniciam por uma faze de desenvolvimento curto e, em seguida de uma longa fase de produção e refinamento, visto que este método tem como a sua principal preocupação a interacção entre o cliente e o desenvolvedor, ou seja, ele visa juntar esses dois lados para que possam trabalhar juntos e assim faz com que seja produzido um software com um valor real. O cliente estará envolvido em todo o projecto. A figura 19 mostra-nos como funciona o ciclo de vida do XP.

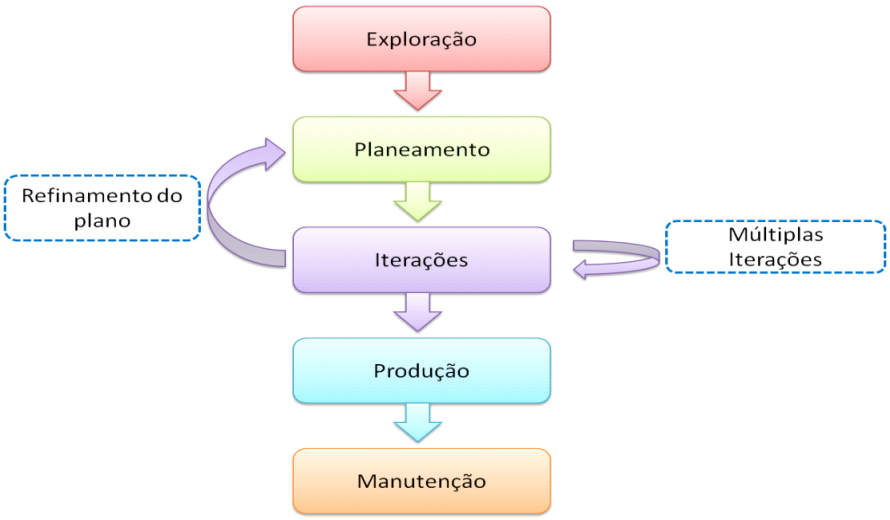



Figura 17 – Ciclo de vida do XP
Fonte: Adaptado de Beck e Andres, 2004

### 2.5.3. Desenvolvimento adaptativo de software – ASD (Adaptative Software Development)

O método RAD (Rapid Application Development) foi a principal causa de inspiração para a criação do ASD.

Em 1992, o metodologista James Highsmith criou o método ASD com base na experiência que ele obteve trabalhando com o RAD. Highsmith com ajuda de um colega (Sam Bayer) conseguiu incorporar aos métodos do RAD conceitos da teoria de sistemas adaptativos complexos, originando o ASD. Os dois metodologistas ao longo das suas parcerias

conseguiram obter muitos sucessos em vários projectos, colocando em prática esse método (Bayer e Highsimth, 1994).

Segundo Highsmith, ASD é um número crescente de alternativas sobre a gestão dos métodos tradicionais, processo acêntrico do software com o foco em pessoas, resultados aos métodos mínimos e a colaboração máxima. É gerada para a alta mudança e alta velocidade sobre os projectos de e-business do dia-dia.

Dentro do ASD existe um "ciclo de vida" que incorpora três fases (Pressman, 2000): especulação, colaboração e aprendizagem.

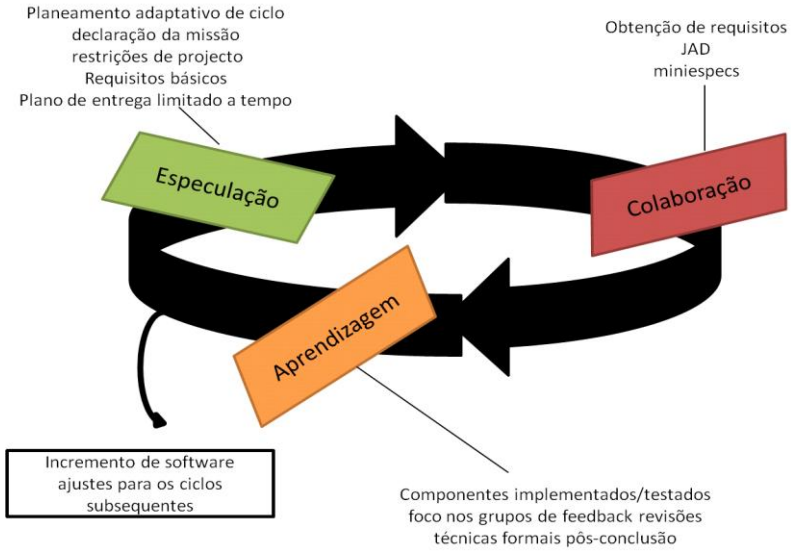



Figura 18 - Desenvolvimento adaptativo de software
Fonte: Adaptado por Pressman

- *Especulação:* durante esta fase, o projecto é iniciado, tomando o lugar do planeamento e, este usa as informações de iniciação do projecto, conduzindo ao planeamento do ciclo adaptativo que são indispensáveis:
  - ✓ Calendário do projecto (data de entrega ou descrições dos utilizadores);
  - ✓ Declaração da missão (feita pelo cliente);
  - ✓ Definir o objectivo de cada iteração;
  - ✓ Atribuição da funcionalidade de cada iteração.

- *Colaboração:* equipa de trabalho motivado, em que estas equipas criam um ambiente estável de modo a comunicarem e colaborarem entre si para que possam resolver os problemas técnicos e, de requisitos de negócios de uma forma eficiente. Mas as colaborações nem sempre são fáceis de se obter, porque nem sempre é fácil de haver comunicação entre os membros da mesma equipa, e para que haja uma boa colaboração tem de haver uma boa comunicação;
- *Aprendizagem:* à medida que o ciclo adaptativo vai-se desenvolvendo, a "ênfase" está tanto na aprendizagem como na direcção do caminho para a conclusão do ciclo (ciclo completo).

Highsimth é um dos 17 metodologistas que participaram na reunião que originou o Manifesto Ágil e ele afirmou que um dos princípios do ASD que se pode citar é o humanismo. Esse sugere que a comunicação (um dos valores da metodologia) seja feita através de contactos pessoais de preferência cara a cara, deixando de lado as documentações pesadas e complexas (Highsimth, 1997).

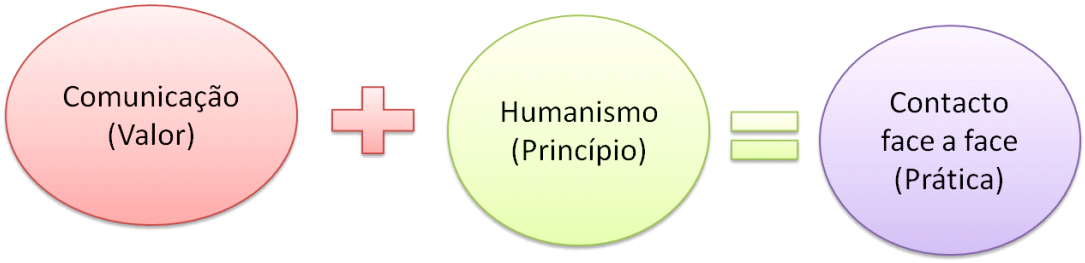



Figura 19 – Princípios do ASD de acordo com os conceitos de Highsimth
Fonte: Chagas (2008)

### 2.5.4. Feature Driven Development – FDD

FDD é um método ágil para a gestão e de desenvolvimento de software. Ele combina as melhores práticas da gestão ágil de projecto com uma abordagem completa para a engenharia de software orientada a objectos, conquistando os três principais públicos de um projecto de software: os clientes, os gestores e os desenvolvedores (Heptagon).

Teve a sua origem através de um projecto que estava ser desenvolvido inicialmente por Jeff De Luca, que depois obteve a ajuda do "colega" Peter Coad. Os dois desenvolvedores acabaram por juntar os seus conhecimentos, visto que De Luca desenvolvia sistemas como

base em metodologias lineares e frameworks leves e Coad desenvolvia modelos orientados a abjectos, com isso os dois acabaram por juntar os conceitos e os processos, dando ao surgimento de uma nova metodologia ágil: Feature Driven Development.

FDD surgiu em 1997, através de uma experiência de análise e de modelação orientada a objectos e da gestão de projectos (Coad, 1999).

Esta metodologia tem como **lema**: *"Resultados frequentes, tangíveis e funcionais".*

*Segundo Highsmith (2002), o FDD tem algumas características particulares como:*

- Resultados úteis a cada duas semanas ou menos;
- Blocos pequenos de funcionalidade de grande valorização pelo cliente, chamados features;
- Planeamento detalhado;
- Rastreabilidade e relatórios com precisão;
- Monitoramento detalhado dentro do projecto, com resumos de alto nível para clientes e gestores, tudo em termos de negócio;
- Fornece uma forma de saber, no início do projecto, se o plano e a estimativa são sólidos.

O FDD vê uma característica como uma função valorizada pelo cliente que pode ser implementada em duas semanas ou menos (Coad, 1999). As características do FDD fornecem alguns benefícios (Pressman, 2000):

- As características são tidas como pequenos blocos de funcionalidade passiveis de entrega, onde os utilizadores podem descrevê-las mais facilmente, entender como elas se relacionam uma com as outras e revisá-las melhor quanto a ambiguidades, erros ou omissões;
- As características podem ser organizadas em um agrupamento hierárquico relacionado ao negócio;
- É um incremento de software passível de entrega, a equipa desenvolve características operacionais a cada duas semanas. Sendo características pequenas,

as suas representações de projecto e de código são mais fáceis de inspeccionar efectivamente;

- ❖ Planeamento de projecto, cronograma e monitoração são guiados pela hierarquia de características em vez de colocar um conjunto de tarefas de engenharia de software arbitrariamente adoptada.

Coad et al (Coad, 99) sugerem a seguinte combinação para definir uma característica (Pressman, 2000):

<acção> o < resultado > < por | para| de | a > um < objecto>

Em que <objecto> é "uma pessoa, lugar ou coisa (inclusive papeis, momentos no tempo ou intervalos de tempo, ou descrições como entradas de catalogo) ".

Sozinho, Coad (Coad, 99) apresentou um conjunto de características que agrupou em categorias relacionadas ao negócio definindo-o como:

<verbo (acção) > um <objecto>

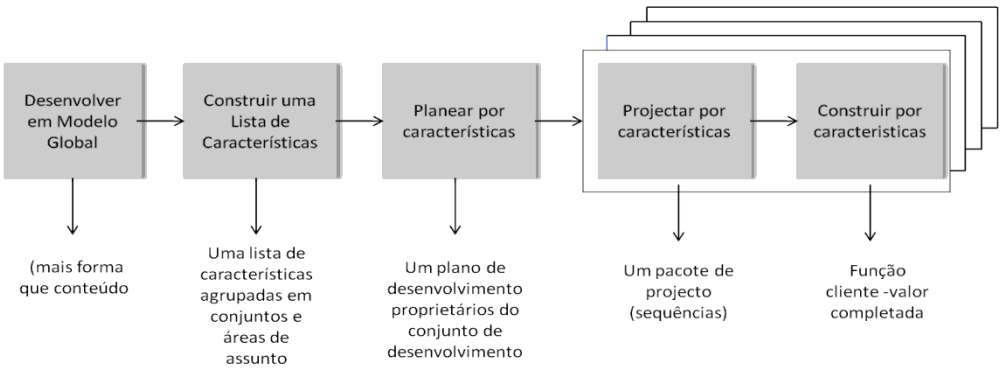



Figura 20 - Desenvolvimento guiado por características (Coad,99)
Fonte: Adaptado por Miguel -Gestão de Projectos de Software

### 2.5.5. Lean Development

Em 1950, surgiu a filosofia de gestão de Lean, em que a indústria Toyota foi a primeira a aplicar essa filosofia. Na década de 90, a indústria de Toyota já estava num patamar bem elevado no que dizia respeito a gestão dos seus negócios e, com isso muitas empresas começaram a adoptar essa filosofia.

Lean é uma tradução do Lean Manufacturing[2] e, princípios e práticas do Lean IT para o domínio do desenvolvimento de software (Wikipédia).

O Lean normalmente é considerado como uma metodologia de negócios, que resulta do aumento do crescimento rentável, do ritmo de inovação acelerado e no aumento da satisfação dos clientes. Ele foca o seu ponto de vista na melhoria de qualidade dos produtos, reduzindo os custos e, melhorando a capacidade de resposta aos clientes.

Saleem (Sall,2008) definiu a filosofia do Lean como sendo uma filosofia de produção que enfatiza a minimização do montante de todos os recursos (incluindo o tempo) utilizados nas várias actividades da empresa.

Muitas empresas utilizam Lean como a filosofia principal para os seus negócios como, por exemplo: FEDEX, MICROSOFT, DELL, WIPRO, BOEING, etc.

Segundo Poppendieck (Poppendieck, 99) existem sete princípios do Lean:

1) Eliminar desperdício.

   "Desperdício é tudo aquilo que não agrega valor ao cliente." (Taiichi Ohno).

   *Existem vários factores que geram o desperdício*:

   - ✓ Códigos incompletos;
   - ✓ Excessos de processos;
   - ✓ Criação de documentos: levam tempo para sua construção e, nem sempre se tem a garantia de que alguém irá lê-los. Ficam desactualizados e podem ser perdidos.

[2] **Wikipédia** - **Lean manufacturing** muitas vezes, " **Lean** ", é uma prática de produção que considera as despesas dos recursos para qualquer outro objectivo que não a criação de valor para o cliente final a ser um desperdício, e, portanto, um alvo para a eliminação.

2) Ampliar a aprendizagem: O processo de aprendizagem é feito através de ciclos de iterações curtas;
3) Atrasar o compromisso: adiar as suas decisões por maior tempo possível, para que possa dar uma resposta com base em factos;
4) Entregar rapidamente.

   "O moral da história é que devemos encontrar uma maneira de entregar o software tão rápido possível para que os nossos clientes não tenham tempo de mudar de ideias." (Mary Poppendieck) As entregas devem ser alinhadas com as perspectivas dos clientes em um determinado momento do tempo;

5) Dar poder a equipa (Team): toda equipa de desenvolvimento tem que ter a capacidade de tomar decisão, pois é algo necessário, visto que todos sabem fazer os seus trabalhos e, com isso são capazes de tomarem a decisão de fazer uma alteração rápida ou até mesmo uma implementação;
6) Construção com Integridade.

   *Existem dois tipos de integridade*:

   - ✓ **Integridade percebida**, quando o cliente precisa ter uma experiência sobre o sistema global (em que o cliente está satisfeito com o software -o software realiza tudo o que o cliente deseja);
   - ✓ **Integridade conceitual**: os componentes do sistema funcionam bem em conjunto como um todo e se mantém útil com o tempo.

7) Visão completa: o produto deve ser visto como um todo, não apenas de uma parte ou como um subsistema.

### 2.5.6. DSDM – Dynamic Systems Development Method

O DSDM surgiu com base no RAD (Rapid Application Development) em meados da década de 90 no Reino Unido. Ele é um dos métodos ágeis que fornece um desenho para que se possa construir e manter sistemas que satisfaz as restrições do prazo apertado por meio do uso de prototipagem incremental em um ambiente que o projecto controlado.

Como o próprio nome diz, esta metodologia é dinâmica porque desenvolve sistemas de uma forma dinâmica e utiliza a prototipagem incremental que é um método de desenvolvimento rápido para as aplicações.

A procura e o ritmo acelerado para alta qualidade do mundo moderno criou-se uma grande pressão sobre o tempo de desenvolvimento de um sistema para produzir um software final em menor tempo possível. Devido a estes factores (tempo e qualidade) os processos que precisam ser seguidos devem ser ágeis e com o levantamento de requisitos sob procura (CRADDOCK, 2006).

O DSDM favorece a filosofia de que nada é perfeitamente construído em um primeiro momento e olha o desenvolvimento de software como um esforço exploratório (CRADDOCK,2006). O DSDM sugere uma filosofia que é emprestada de uma versão modificada do princípio da Pareto. Nesse caso, 80% de uma aplicação pode ser entregue em 20% do tempo que levaria para entregar a aplicação completa (100%). (Pressman, 2000)

Segundo Abrahamsson (Abraham, 02), o DSDM proporciona um excelente método que permite que os sistemas robustos sejam entregues em pequeno espaço de tempo. Este método faz parte de uma organização cujo nome é *DSDM Consortium* que é um grupo mundial de empresas que "defendem" esse método. Esse grupo definiu um modelo ágil de processo em que o denominou como ciclo de vida do DSDM.

O ciclo de vida do DSDM encontra-se dividido em três fazes, que normalmente são chamados de ciclos iterativos. A figura 23 descreve como funciona o ciclo de DSDM quando se está a desenvolver um projecto.


Figura 21 – Ciclo de Vida de um projecto em DSDM
Fonte: http://www.steveforceconsulting.com/Agile/Agile.html

- Pré – Projecto: nesta primeira fase, o primeiro passo a dar é verificar se a metodologia DSDM é adequada para desenvolver o projecto em causa, o segundo passo a fazer é verificar se o plano de financiamento é o suficiente para manter o projecto até a sua recta final e, assim por diante;
- Projecto: para concluir esta fase a equipa desenvolvimento terá de trabalhar bem e organizada para poder concluir as cincos actividades que existem dentro desta fase e, ao concluírem estas actividades, criam um SI (Sistemas de Informação). Pressman em 1999 deu o seu contributo no que diz respeito a essas actividades que fazem parte do projecto (Pressman, 2000):
  - ✓ Estudo de viabilidade estabelece os requisitos básicos e as restrições dos negócios associados ao projecto em causa;
  - ✓ Estudo do negócio estabelece os requisitos funcionais e de informação que permitirão a aplicação fornecer valor de negócio;
  - ✓ A iteração do modelo funcional produz um conjunto de protótipos incrementais que demonstram a funcionalidade para o cliente;
  - ✓ Iteração de projecto e construção revista dos protótipos construídos durante a iteração do modelo funcional para garantir que cada um tenha passado por engenharia de modo que seja capaz de fornecer valor de negócio operacional para os utilizadores finais;
  - ✓ A implementação coloca o último incremento de software no ambiente operacional.
- Pós- Projecto: esta última fase tem o dever de garantir que o sistema funcione correctamente. Isto é alcançado através da manutenção e dos melhoramentos de acordo com os princípios do DSDM.

*Um projecto realizado em DSDM tem de seguir os nove princípios desta metodologia:*

1. O envolvimento activo do utilizador é imperativo;
2. O tempo deve ter o poder para tomar decisões;
3. O foco é na entrega frequente dos produtos;
4. O encaixe ao propósito do negócio é o critério essencial para a aceitação das entregas;

5. O desenvolvimento iterativo e incremental é necessário para convergir com precisão a solução do negócio;
6. Todas as mudanças durante o desenvolvimento são reversíveis;
7. Requisitos são alinhados em um alto nível;
8. O teste é integrado por todo o ciclo de vida ;
9. Uma abordagem colaborativa e cooperativa entre as partes envolvidas são essenciais.

### 2.5.7. Crystal

Este método foi desenvolvido antes dos métodos XP e FDD, teve o seu início em 1992, mas, só ficou conhecido como *Crystal* em 1997, antes do manifesto ágil.

Foi desenvolvido por um senhor chamado Cockburn. Ele definiu este método como um processo que é diferente do XP, FDD entre outros métodos ágeis pelo facto de que o Crystal é uma família de metodologias e não somente uma metodologia (COCKBURN, 2008).

Pressman definiu a metodologia Crystal como sendo uma família de métodos ágeis de processo que podem ser adoptados para as características especificas de um projecto (Pressman, 2000).

Essa metodologia pode ser adaptada em diferentes tipos de projectos e equipas (grupos de 6 a 80 membros).

Uma das características mais importantes deste método é a escolha dos membros para realizar o trabalho. A esses membros dá-se o nome de membros da família Crystal.

Os membros da família crystal são agrupados por cores (Branco, amarelo, laranja e vermelho) para poder indicar o seu “peso”.

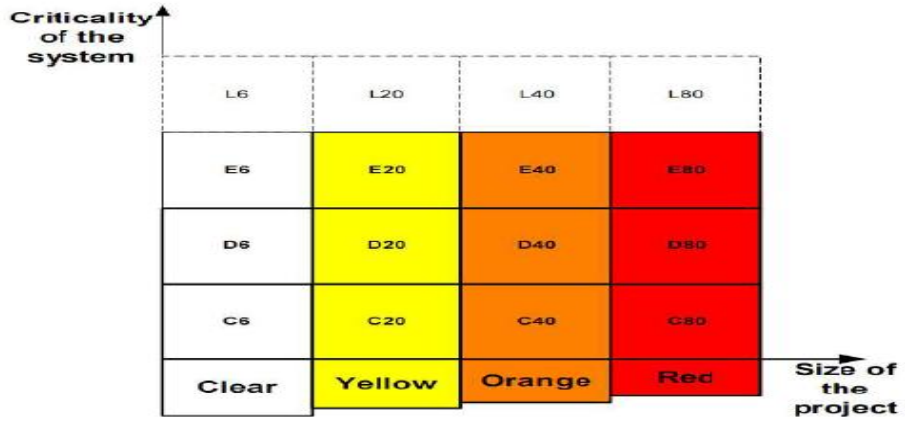



Figura 22 – Distribuição dos métodos da família crystal através de duas dimensões
Fonte: http://pt.scribd.com/doc/20603045/FDD-e-Crystal-Conceitos-Vantagens-e-Desvantagens

Os diferentes membros da família podem ser adaptados para caber em variáveis circunstanciais (RUSK, 2006).

Quanto mais às cores ficam escuras, mais "pesada" é a metodologia, isso porque um processo com mais de cem membros necessita de metodologias mais robustas do que um com somente alguns membros.

*A família Crystal possui dois valores que são seguidos por todas suas metodologias:*

- Foco na comunicação interpessoal;
- Alta tolerância.

Penteado (2006) definiu duas regras mais comuns que existem dentro da família Crystal:

- O projecto deve usar um desenvolvimento incremental, com incrementos de quatro meses ou menos (com uma forte preferência para incrementos de um a três meses);
- A equipa deve possuir oficinas para reflexão no pré e pós-incremento.

### 2.5.8. SCRUM

#### 2.5.8.1. História

A palavra Scrum teve a sua origem no futebol Americano (Rugby), onde esta palavra é utilizada para fazer o alinhamento dos jogadores (Schwaber et Beedle, 2002). Nesse tipo de jogo há diversos tipos de jogadores, com características diferentes, mas, todos com o mesmo objectivo.

No final da década de 90 três senhores cujos nomes são: Jeff Sutherland, John Scumniotales e Jeff McKenna, com ajuda das suas equipas aperceberam que quando se desenvolve um projecto com pequenas equipas e multidisciplinares, acabavam por obter melhores resultados. A partir daquele momento os três metodologistas começaram a conceber, documentar e a implementar o Scrum. Na época trabalhavam na *Easel Corporation*. Ao mesmo tempo, outro metodologista chamado Ken Schwaber estava a utilizar essas mesmas práticas para gerir o desenvolvimento do software dentro da sua empresa (Sutherland, 2005) e, em 1995 Schwaber formalizou a definição do Scrum, o que ajudou-o muito na sua implementação e no desenvolvimento de software no mundo inteiro.

Muito antes da concepção deste método, algumas empresas como a Honda, Fuji-Xerox, Canon, etc., já utilizavam as práticas do método Scrum e vinham sendo observados por dois professores Japoneses (Hirotaka Takeuchi e Ikujiro Nonaka) que chegaram a conclusão que quando utilizavam equipas pequenas e multidisciplinares, obtinham um resultado muito acima do esperado. Com base nesses factos, os dois professores publicaram todos os seus estudos em 1986 no Harvad Business Review[3].

Scrum junta os conceitos de Lean, desenvolvimento iterativo e o estudo de Hirotaka Takeuchi e Ikujiro Nonaka. (Wikipédia)

[3] Harvard Business Review é uma revista de administração e negócios publicada desde 1922 pela Harvard Business Publishing, editora da Harvard Business School

#### 2.5.8.2. Abordagem

O método Scrum é uma das metodologias ágeis que apresenta uma grande comunidade de utilizadores. Como já foi dito no capítulo 2.3, hoje em dia a maioria parte dos desenvolvedores de software vem demonstrando cada vez mais as suas preferências pelo método scrum.

Sobre a definição do método scrum existem diferentes opiniões, mas, todas elas acabam por reunir e apontar os mesmos objectivos:

- Scrum é um processo de desenvolvimento iterativo e incremental para a gestão de projectos e desenvolvimento ágil do software (Wikipédia);
- Scrum é um processo para projectos e desenvolvimento de software orientado a objecto, focado nas pessoas e indicado para ambientes em que os requisitos aparecem e mudam rapidamente (Schwaber, 2001);
- Scrum é um processo ágil que pode ser usado para gerir e controlar software complexo e desenvolvimento de produtos utilizando as práticas do modelo iterativo e incremental (*ControlChaos*);
- Mike Cohn definiu o Scrum em 100 palavras (Mike Cohn, 2009):
  - ✓ Scrum é um processo ágil que permite manter o foco na entrega do maior valor de negócio, no menor tempo possível;
  - ✓ Isto permite a rápida e contínua inspecção do software em produção (em intervalos de duas a quatro semanas);
  - ✓ A necessidade de negócio é que determina as prioridades do desenvolvimento de um sistema. As equipas se auto-organizam para definir a melhor maneira de entregar as funcionalidades de maior prioridade;
  - ✓ Entre cada duas a quatro semanas todos podem ver o software real em produção, decidindo se o mesmo deve ser aceite ou continuar a ser aprimorado por mais um "Sprint".

A metodologia scrum foi desenvolvida para a gestão de processos de desenvolvimento de sistemas.

Hoje em dia muitas empresas conhecidas no mercado utilizam o método scurm e, ele tem sido utilizado para desenvolver vários tipos de sistemas, como por exemplo: software comercial, projectos de preços fixos, desenvolvimento interno, desenvolvimento contratado (terceirização), aplicações financiadas pela ISSO 90001, jogos, web, sistemas para controlo de satélites, telemóveis, etc.

O scrum concentra-se na forma como os membros da equipa devem se portar a fim de produzirem o sistema de forma flexível e em um ambiente de mutação constante. (Schwaber; Beedle, 2002)

Os projectos dentro da metodologia Scrum são divididos em ciclos (tipicamente mensais) que são chamados de *Sprints*. Sprint representa um *time box* no qual um conjunto de actividades deve ser executado. Esses sprints são de curta duração, pois só duram 30 dias.

Para realizar um sprint é necessário formar uma pequena equipa de até sete membros que realizam todo o trabalho de acordo com as funcionalidades (requisitos) definidas no início de cada sprint, sendo esta equipa responsável pelo desenvolvimento da funcionalidade do sistema.

Todo o processo do desenvolvimento do sprint é realizado dentro do ciclo de vida do scrum.

O ciclo de vida do scrum é dividido em três fases principais, que demonstra como todo o fluxo do processo scrum é realizado. As três principais fases são: Pré-Game, Mid-Game e Post-Game ou Pré-Sprint, Sprint e Pós-Sprint. A figura seguinte demonstra como é dividido o ciclo de vida do scrum.

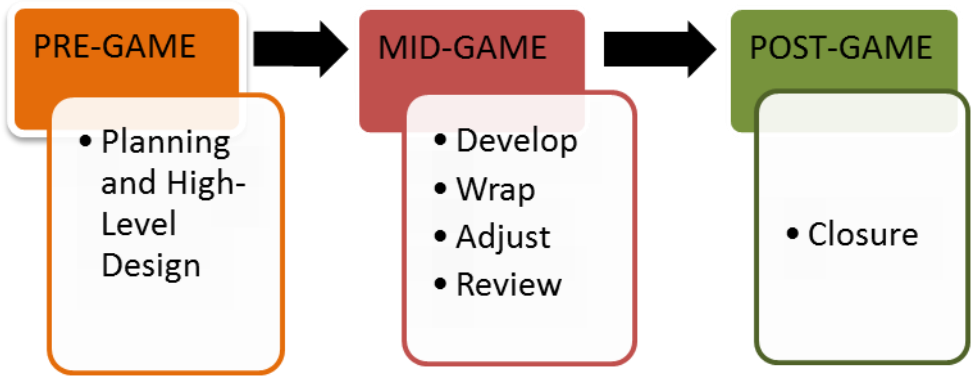



Figura 23- Ciclo de vida do Scrum
Fonte: Adaptado do Site - www.codeproject.com

A fase inicial Pré-Game / Pré-Sprint é a de preparação, que é considerada como uma fase preliminar ao projecto, onde é feito todo o planeamento e as decisões sobre o orçamento, a concepção do software, a equipa, etc., ou seja, nesta fase é definida a lista do *Backlog* do produto, sendo que quem define esta lista é o *Product Owner*.

O Mid-Game / Sprint é tida como a parte ágil do processo, porque é nesta fase que se desenvolve os sprints. E a partir dali, cada sprint desenvolve o seu papel.

Os sprints são ciclos ou iterações periódicos que varia de três a quatro semanas, sendo que ao mesmo tempo é desenvolvido o produto real. Cada sprint inclui fases tradicionais de desenvolvimento de software (engenharia de requisitos, análises, concepção e entrega). Um sprint pode durar de uma semana a um mês.

Quando falamos dos sprints, estamos a falar do desenvolvimento de todo o processo do scrum, visto que é nesta fase que todo o trabalho é realizado. A esse processo dá-se o nome de ciclo de um sprint.


Figura 24 – Ciclo de um sprint
Fonte: http://blog.fasagri.com.br/

Um sprint é alimentado por duas subfases, que são realizadas antes da concepção de um sprint: Product Backlog e Sprint Backlog.

O product backlog é uma lista de todas as funcionalidades desejadas para o produto, sendo esta lista definida pelo Product Owner (dono do produto). O product backlog pode não estar completo no início de projecto mas, a medida que o projecto vai crescendo, ele vai crescer juntamente com o projecto.

A cada novo ciclo poderão ser alterados os itens deste backlog, podendo-se inserir, mudar ou excluir requisitos que foram especificados antes. Após a definição do backlog do produto, organiza-se o backlog do sprint (backlog que será executado durante o ciclo do sprint).

O *sprint backlog* é uma lista de tarefas que a equipa do scrum (Team Scrum) se compromete a desenvolver em um sprint. Todos os itens do sprint backlog são retirados do product backlog, com base nas prioridades definidas pelo Product Owner.

Após estar definido todos os requisitos do produto, começa-se o desenvolvimento (sprint).

O ciclo de um sprint começa com o Sprint *Plannig Meeting* que é realizado dentro de cada sprint.

O *Planning Meeting* é uma reunião realizada no início de cada sprint, em que se decide o que vai ser feito durante o processo do sprint que está a ser desenvolvido. Os participantes do planning meeting são: Product Owner, Scrum Master e todo o Scrum Team.

Durante essa reunião o Product Owner descreve as funcionalidades prioritárias para a equipa. Em seguida a equipa em conjunto, define o que pode entrar no desenvolvimento do próximo sprint. Durante todo o processo de incremento, são realizadas pequenas reuniões diárias, de curta duração que tem o nome de Daily Scrum ou Scrum Diário.

O Daily Scrum é uma reunião que é realizada para ajudar os membros da equipa a manter-se no rumo. Eles se reúnem com os seguintes objectivos:

- ✓ O que se fez ontem?
- ✓ O que se vai fazer hoje?
- ✓ Quais são os obstáculos?

No final de cada sprint, é realizado um Sprint Review Meeting, em que a equipa do scrum mostra o que foi alcançado durante o processo do sprint. O ponto mais importante desta reunião é que todo o objectivo do sprint esteja realizado.

Logo após é realizado o Sprint Retrospective, onde o sprint é colocado em funcionamento para verificar o que funciona bem, o que poderia ser mudado e quais as acções que vão ser tomadas para melhora-las.

Após todo esse processo segue-se a fase final do projecto que é o Post –Game, que é a fase da entrega do produto final e fim do projecto, ou seja, quando todas as variáveis de ambiente estão em equilíbrio e os requisitos iniciais foram alcançados o projecto está apto a ser entregue. (SCHWABER; BEEDLE, 2002).

**Responsabilidades dentro do SCRUM**

Antes de se começar a desenvolver o sistema, é necessário definir o papel de cada participante do projecto.

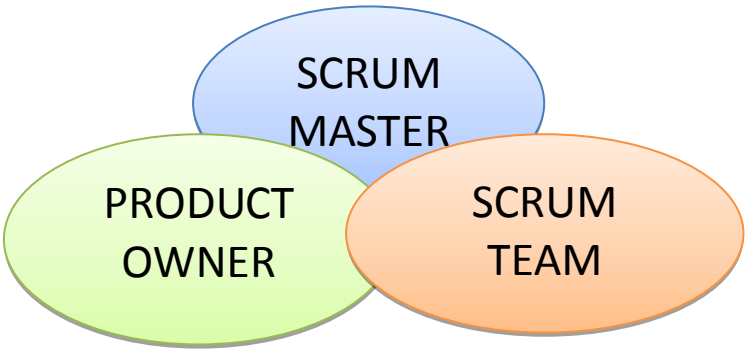



Figura 25 – Responsabilidades e Regras do Scrum

Scrum Master é tida como o gestor do projecto, em que este tem as seguintes funções:

- Assegurar que a equipa siga e respeite as práticas do Scrum;
- Protege a equipa para que não se comprometa com mais do que possa realizar no sprint;
- Actua como facilitador do Daily Scrum, remove qualquer obstáculo que seja levantada na reunião.

O Product Owner é o dono do produto, ou seja, o cliente é quem dita todas as funcionalidades do programa e, esse cliente é Product Owner. Ele é quem intende do negócio.

O Scrum Team é a equipa que vai desenvolver o programa. Geralmente são equipas pequenas de até 10 pessoas. Essa equipa tem de ser multifuncional. A equipa reporta todos os seus actos ao Scrum Master e ao Product Owner.

#### 2.5.8.3. Valores

Os valores do scrum estão interligados aos valores descritos no manifesto ágil.

*Michele Slige descreveu os cincos valores do Scrum:*

1. Compromisso: quando estamos despostos a assumir o compromisso de uma meta e, fazer o melhor para cumprir essa meta;
2. Foco: concentrar todos os seus esforços e competência no trabalho que lhe foi entregue, ao invés de se deixar distrair por outras coisas;
3. Abertura: o scrum mantém tudo sobre um projecto visível para todos, ou seja, nós entendemos e partilhamos tudo sobre um projecto de uma forma honesta e verdadeira;
4. Respeito: todos os membros da equipa têm de se respeitar mesmo que sejam diferentes;
5. Coragem: ter a ousadia e a coragem de se comprometer, de agir, de estar aberto e aguardar o respeito.

#### 2.5.8.4. Princípios

O princípio é a chave desta metodologia, sendo um reconhecimento que durante o processo de um projecto os clientes podem mudar de opiniões sobre o que eles querem e precisam.

*Os princípios do Scrum (ADM 96):*

1) Pequenas equipas de trabalho são organizadas de modo a "maximizar a comunicação, minimizar a supervisão e maximizar o compartilhamento de conhecimento tácito informal";
2) Os processos precisam de ser adaptáveis tanto as modificações técnicas quanto as de negócios "para garantir que o melhor produto possível seja produzido";
3) O processo produz incrementos frequentes de software "que podem ser inspeccionados, ajustados, testados, documentados e expandidos";
4) O trabalho de desenvolvimento e a equipa que o realiza é dividido "em partições claras, de grupos pequenos, ou em pacotes";

5) Testes e documentações constantes são realizados a medida que o produto é construído;
6) O processo scrum fornece a "habilidade de declarar o produto 'pronto' sempre que necessário (porque a concorrência acabou de entregar, porque a empresa precisa de dinheiro, porque o utilizador/cliente precisa das funções, porque foi para essa data que foi prometido…) ".

#### 2.5.8.5. Práticas

A metodologia scrum não exige quaisquer métodos e também não fornece práticas específicas para serem utilizadas no desenvolvimento do software. Mas, ao invés disso requer certas práticas de gestão e ferramentas nas diferentes fases do processo do scrum, em que com isso procura evitar confusões que são causadas pela complexidade e imprevisibilidade dos processos.

*As práticas do scrum são:*

*i. O Product Backlog*

Essa prática define todo o trabalho a ser feito no projecto, ou seja, define o que é necessário para a concepção do produto final, com base no conhecimento actual. É feita uma lista, priorizada e constantemente actualizada dos requisitos técnicos e de negócio para o sistema.

O backlog contém todas as tarefas directas, caso de uso, as histórias, características, funções, etc.

Segundo o Abrahamsson (2002), o dono do produto é responsável pela manutenção do Product Backlog.

*ii. Estimação do Esforço*

É um processo iterativo, no qual todas as estimativas de um item do Backlog centram-se num nível mais preciso quando existem mais informações disponíveis sobre um determinado item.

O proprietário do produto juntamente com o Team Scrum são responsáveis pelo desempenho da estimação de esforço. (ABRAHAMSSON, 2002)

*iii. Sprint*

É um ciclo iterativo de desenvolvimento de alguns requisitos do projecto. O sprint também é tido como um processo de adaptação a mudança de variáveis de ambientes. O scrum tem uma equipa que se organiza para poder produzir um novo incremento no produto executável em um sprint que tem uma duração de 30 dias.

As ferramentas de trabalho da equipa são Sprint Planning Meetings, Sprint Backlog e Daily Scrum meetings (reuniões diárias). (SCHWABER; BEEDLE, 2002)

*iv. Sprint Planning Meeting*

É uma reunião que é realizada no início de cada sprint, contendo duas fases. A primeira fase da reunião conta com a participação dos utilizadores, gestores, o dono do produto e a equipa do scrum, em que vão decidir quais as funcionalidades e as metas do próximo sprint. Na segunda fase só se reúnem a equipa do scrum e o scrum Master, em que decidem como é que o incremento do produto será implementado no sprint.

*v. Sprint Backlog*

É o ponto de partida para cada sprint. É uma lista de tarefas ou itens de um produto backlog seleccionado pela equipa do scrum juntamente com o scrum Master e o dono do produto para serem implementados.

*vi. Daily Scrum Meeting*

São reuniões organizadas para manter o controlo contínuo sobre o andamento da equipa, monitorizando o seu desempenho, além de servir como reuniões para o planeamento: o que foi feito ontem? O que vai ser feito hoje?

*vii. Sprint Review Meeting*

É uma reunião informal que é feita ao final de cada sprint, onde a equipa e o Scrum Master mostram os resultados obtidos ao cliente/utilizador e o Product Owner, o que foi criado durante todo o ciclo de vida de um sprint. Uma reunião do sprint review tem uma duração de 4 horas.

#### 2.5.8.6. Conclusões

Esta metodologia apresenta vantagens e desvantagens como qualquer outra metodologia de desenvolvimento de software.

*Vantagens*

- Velocidade;
- Motivação maior dos programadores;
- Evita surpresas com os resultados;
- Diminuição dos Bugs;
- Prioridades podem ser alteradas;
- Funcionalidade que agrega valor.

*Desvantagens*

- Sensação de informalidade;
- Prazo;
- Falta de planeamento do desenho;
- Projecto não documentado;
- Papéis indefinidos.

## 2.6. Vantagens e desvantagens de metodologias ágeis

*Vantagens*

- Metodologia ágil tem uma equipa adaptativa que é capaz de responder as novas exigências;
- A equipa não tem que investir tempo e esforço para descobrir que no momento da entrega do produto o cliente mudou de ideias a cerca dos requisitos do sistema;
- Comunicação face a face e entradas contínuas de representante do cliente não deixa espaço para conjecturas;
- A documentação é nítido e directo ao ponto para economizar tempo;

- O resultado final é um software de alta qualidade em menor tempo possível e, o cliente satisfeito.

*Desvantagens*

- No caso de algumas entregas de software, especialmente de grandes portes, é difícil avaliar o esforço necessário no início do ciclo de vida de desenvolvimento de software;
- Há falta de ênfase necessário para a concepção e documentação;
- O projecto poderá facilmente tomar um rumo diferente e, o resultado final pode ser o oposto do que o cliente precisa, se o representante do cliente não for objectiva e directo;
- Somente programadores seniores são capazes de assumir e tomar tipos de decisões necessárias durante o processo de desenvolvimento. Por isso, não tem lugar para programadores novos, a menos que combinado com recursos experiente.

## 2.7. Comparação entre as metodologias ágeis e tradicionais

Depois de analisar individualmente cada uma das características das metodologias ágeis e tradicionais, neste ponto vai ser feito uma comparação entre as dois métodos, em particular do modelo em Cascata (metodologia tradicional) e o método Scrum (metodologia ágil).

Segundo Cockburn (Cockburn et al., 2001), as metodologias ágeis não possuem nada de novo. O que as deferências das metodologias tradicionais é o enfoque e os valores.

As metodologias ágeis têm como uma das suas principais preocupações gastar menos tempo com a documentação e mais com a implementação.

Esses dois tipos de processos de desenvolvimento vão ser analisados e comparados da seguinte forma:

***1) Requisitos vs User Stories***

O **modelo em cascata** define detalhadamente os requisitos, em que normalmente estes não podem ser alterados porque quando se altera os requisitos tem de se voltar ao início do projecto e começar a desenvolver um novo código, ou seja, começar tudo de novo. Cada aspecto ou etapa do projecto é definido e documentado. Normalmente, um processo de sign-off muitas vezes segue o acordo contratual feito entre o desenvolvedor e o cliente.

O **método Scrum** define os requisitos de acordo com os históricos dos clientes, sendo que estes não definem os requisitos técnicos mas, ao invés disso eles descrevem os objectivos dos utilizadores e as suas tarefas. Cada histórico é realizado no momento em que ele é construído, em que a equipa começa a trabalhar sobre os históricos mais importantes dos utilizadores.

***2) Preditivo vs Empíricos***

**Cascata:** os requisitos são definidos logo no início do projecto e são aprovados pelos clientes antes da sua concepção.

**Scrum:** os requisitos são definidos empiricamente, em vez de definir tudo no princípio. Todas as necessidades do cliente vão ser trabalhadas de acordo com a ordem do valor do projecto. O Scrum aceita mudança de requisitos ao longo da construção do projecto.

***3) Indivíduo vs A equipa***

**Cascata:** existe um responsável pelo projecto. O modelo em cascata tem uma estrutura hierárquica que é constituída pelo gestor do projecto, que tem a responsabilidade global sobre o projecto e os indivíduos que muitas vezes trabalham em vários projectos ao mesmo tempo.

**Scrum:** nesse método é atribuída uma equipa ao projecto, que acompanha o projecto desde a sua fase inicial até a sua fase final. A equipa é constituída por pequenos grupos. Não existe hierarquia e toda a equipa assume a responsabilidade pela entrega do projecto.

### *4) Sequencial vs Iterativo*

No modelo em **cascata** é feito o cronograma do projecto com fases sequenciais, em que estas fases são dependentes umas das outras. O cronograma é planeado logo no início do projecto. O projecto é entregue quando todo o cronograma estiver concluído.

No **Scrum**, o cronograma do projecto é dividido em iterações, que duram de 2 a 4 semanas. No início de cada iteração a equipa e o cliente reúnem para decidirem quais os recursos que vão ser implementados e entregues.

### *5) Custo de mudanças vs mudança incentivado*

**Cascata:** para fazer mudanças no projecto, muitas vezes é difícil, visto que exige e implica um aumento do orçamento e a ampliação do cronograma. A mudança neste modelo pode ser caro.

**Scrum:** a mudança é sempre bem-vinda em todas as fases do projecto. Uma iteração concluída pode ser alterada mais tarde no projecto. Todas essas mudanças podem ser feitas sem ter que aumentar o orçamento.

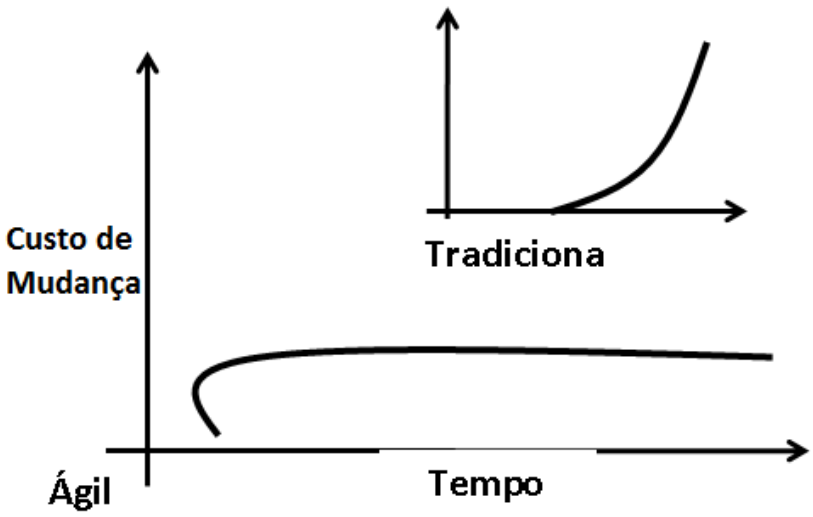



Figura 26 – Custo das alterações no desenvolvimento Ágil e Tradicional
Fonte: http://devagil.wordpress.com/2007/07/07/introducao-ao-desenvolvimento-agil/

# Capitulo III

## *3. Ganhos de produtividade e de sucesso de metodologias ágeis vs. metodologias em cascata no desenvolvimento de projectos de software*

Quando se fala do desenvolvimento de software existe uma grande tensão entre a qualidade, os custos, o tempo de concepção e entrega do software. Estes três termos são muito valorizados pelos engenheiros de software e não só. Mas, também existe um outro ponto que é muito valorizado, tanto por parte dos desenvolvedores de software como para os clientes, que são os retornos de investimentos (ganhos) obtidos com os projectos.

Para desenvolver um projecto é preciso investir tempo e recursos e, estes investimentos são considerados satisfatórios quando são compensados através do valor gerado, ou seja, quando o software satisfaz as expectativas de quem investiu nele.

Rico em 2008 através de um estudo que efectuou, concluiu que:

- As indústrias de software obtiveram uma receita de 393 milhões de dólares, o B2C e B2B no comércio electrónico alcançou os 220 bilhões de dólares e 2,7 milhões de dólares;
- Depois dos sites web, a TI é a segunda principal contribuinte para a economia não só dos EUA como também do mundo. Promove mais de metade do crescimento da produtividade do trabalho.

Os projectos desenvolvidos pelos métodos ágeis são tidos como processo de desenvolvimento baseado em bens e serviços.

Por outro lado, as metodologias ágeis são utilizadas para alcançar os objectivos e as satisfações dos clientes, através da sua implementação rápida e dos testes antecipados tendo assim efeitos na satisfação dos clientes e na respectiva qualidade do produto.

Na maioria dos casos os benefícios quantitativos são tidos com um termo económico. Como já demonstrado, o uso dos métodos ágeis é concebido num menor espaço de tempo do que os métodos tradicionais, sendo efectivamente considerado como uma vantagem económica directa no que diz respeito ao aumento da produtividade.

O valor comercial das metodologias ágeis quebra muitos mitos e equívocos fundamentais ao redor do fenómeno moderno dos métodos ágeis para criação dos produtos de

software, fornecendo uma comparação do valor total de negócios entre os dois métodos (David Rico).

As metodologias ágeis estão a ganhar cada vez mais força no mercado, principalmente no que diz respeito a implementação do método Scrum. Este método vem surpreendendo de uma forma muito positiva e demonstrando que os ganhos podem ser substancialmente superiores às expectativas. Os principais ganhos que as metodologias ágeis trouxeram para as empresas com a sua adopção foram:

- A flexibilidade: atende as mudanças do meio ambiente competitivo e as variações do mercado;
- Time-to-Market: reduzido para garantir a entrada de novos produtos e introduz as soluções antecipadamente de forma a ser o primeiro a faze-lo e, com isso obtém a um lugar diferente dentro do mercado competitivo;
- Diminuição de custos: como resultado da diminuição do trabalho a ser efectuado, a eficiência do desenvolvimento aumenta significativamente.

Os Factores Críticos de Sucesso de um projecto podem ditar o insucesso ou o sucesso do mesmo, ou seja, podem determinar os benefícios/ganhos ou os custos/perdas.

Para que se possam definir os Factores Críticos de Sucesso de um projecto de uma forma mais exacta possível e com a criação de valor acrescentado, deve-se basear essa mesma definição na análise do projecto a desenvolver, na sua globalidade. Neste contexto, podem-se enumerar alguns exemplos de Factores Críticos de Sucesso já tipificados ao longo de vários projectos e no âmbito das Tecnologias de Informação, como por exemplo:

- A disponibilização atempada, por parte do Empresa que compra ou pretende que lhe seja instalado um determinado software, não possuir a arquitectura necessária para a instalação do respectivo *Hardware* e *Software;*
- A ausência da disponibilização de uma infra-estrutura técnica para o ambiente de desenvolvimento (que engloba também os testes) em tudo, idêntica à existente para o ambiente de qualidade e no final, para o ambiente de produção, sendo este último quando o sistema já está a ser utilizado pelo Cliente.

Para que, se possa afirmar com segurança quais os ganhos de produtividade que podem ser atingidos com a implementação de um software e que, evidentemente vão existir

ganhos, este Factor Crítico de Sucesso (ambiente de desenvolvimento, qualidade e produção) é, talvez um dos mais importantes a ter em conta na implementação desse mesmo software, ou seja, é recomendável, se não fundamental a criação de dois ambientes distintos, que irão coexistir durante a realização do projecto:

- O Ambiente de Desenvolvimento e o Ambiente de Qualidade, devem ser utilizados para a execução de todas as actividades inerentes ao desenvolvimento, configuração, testes e alteração dos componentes base da plataforma envolvida na solução proposta;
- No Ambiente de qualidade podem ser executadas mais uma fase/bateria testes, tanto a nível funcional como técnico. Os desenvolvimentos são assim novamente validados, verificando se estão de acordo com os requisitos identificados, se têm em consideração as normas e procedimentos de qualidade que o Cliente exige nos seus projectos. Após aprovação por parte do Cliente desta fase (Ambiente de Qualidade) com a assinatura de um documento a formalizar a aprovação (Ambiente de Qualidade) e que se pode prosseguir para o Ambiente de Produção.

Neste âmbito, o Ambiente de Produção é o ambiente para onde serão migrados todas as configurações e desenvolvimentos efectuados no Ambiente de Desenvolvimento/Qualidade. Neste ambiente não serão realizados desenvolvimentos ou testes uma vez que todos os procedimentos já foram efectuados nos ambientes já acima referidos, os quais garantem assim a qualidade da informação existente.

A quando da implementação do projecto, o Cliente pode solicitar a alteração a um procedimento já anteriormente aprovado e já testado no Ambiente de Desenvolvimento e de Qualidade, logo é considerada esta medida uma alteração ao âmbito, a qual pode originar custos e riscos para a empresa que esta a desenvolver o software. Por conseguinte, o procedimento correcto é efectuar novamente os três Ambiente já referidos e alertar o Cliente de uma possível alteração no valor do seu projecto provocada pela pedido de alteração ao âmbito inicial.

- Monitorização e controlo do projecto - ainda sobre os ganhos de produtividade, outro Factor Critico de Sucesso é a monitorização e controlo do projecto. Para que se receba um retorno positivo na implementação do projecto, esta deve ser baseada numa correcta verificação do estado das diferentes actividades das

unidades de trabalho, face ao progresso planeado para as mesmas, devem ser verificados os estados dos diferentes entregáveis (entenda-se, a documentação que sustenta a evolução do respectivo projecto, sendo esta fundamental para memória futura) do projecto. Com esta documentação, devem ser verificadas e garantidas de que as milestones (datas previamente estabelecidas) são cumpridas nas datas previstas. Ainda é importante a verificação do estado das diferentes acções faces às datas de conclusão previstas e a verificação da operacionalidade da estrutura de projecto;

- Métricas e controlo de custos para todos os procedimentos de monitorização e controlo do projecto –Ao longo do desenvolvimento do projecto devem ser implementados mecanismos específicos, tais como métricas e controlo de custos por forma a que não existam custos imprevistos ou derrapagens no valor total do respectivo projecto. Adicionalmente, serão implementados mecanismos de reporting dos resultados dessas actividades ao Cliente.

De acordo com a prática, com vista à obtenção de ganhos de produtividade num projecto, à sua melhor coordenação e ao seu sustentado acompanhamento (tendo em conta a complexidade dos projectos, e os diferentes tipos de necessidade dos mesmos), estão tipificados / definidos cinco níveis de gestão para que, efectivamente se possa atingir de uma forma mais fácil, esses mesmos ganhos de produtividade, a boa coordenação e o seu sustentado acompanhamento:

- A importância da correcta identificação dos utilizadores das áreas abrangidas pela implementação deste projecto;
- A importância da aprovação da análise e desenho apresentado para a Solução a implementar (desta forma as expectativas dos utilizadores estarão correctamente enquadradas com os objectivos definidos para o Projecto e com o trabalho de análise efectuado pela equipa funcional);
- A importância da performance do Sistema (a adequação do dimensionamento da infra-estrutura às necessidades de utilização do sistema);
- A compreensão total dos processos de trabalho, para que estes espalhem as actividades / necessidades da Empresa Nossa Cliente;

- Formação e apoio aos futuros utilizadores (é fundamental os futuros utilizadores serem integrados e acompanhados de forma cuidada, caso contrário, podem não achar mais valia no software implementado).

Muitas empresas e autores (Scott Ambler, versionone, SEI, David Rico, etc.) fizeram pesquisas sobre os ganhos e os sucessos das metodologias de desenvolvimento de software. O autor Scott Ambler fez uma pesquisa e chegou a conclusão de que em termos dos ganhos, a pontuação da metodologia ágil é de 3,9 sobre 0,2 da metodologia tradicional numa escala de -10 a +10. (Ambysoft)

Nos primeiros estudos realizados sobre a gestão de projectos com processos ágeis, foram apresentados os seguintes valores: 10% a 20% de melhorias nas receitas, na qualidade, tempo e redução do ciclo de vida do projecto e 54% nos custos. Mas, em 2008 a Universidade de Maryland desenvolveu uma base de dados onde é armazenado todos os dados dos estudos feitos sobre os custos e os benefícios de gestão dos projectos ágeis e tradicionais, e chegaram a conclusão que os ROI dos projectos ágeis (métodos ágeis) eram seis vezes maior do que os projectos tradicionais (métodos tradicionais) e com melhor qualidade.

| Metric | Agile | Traditional | Difference |
|---|---|---|---|
| Cost Reduction | 29% | 20% | 9% |
| Schedule Reduction | 70% | 37% | 33% |
| Productivity Improvement | 117% | 62% | 55% |
| Quality Improvement | 74% | 50% | 24% |
| Customer Satisfaction Imp. | 70% | 14% | 56% |
| Return on Investment | 2,811% | 470% | 2,341% |

Tabela 1: ROI entre as duas metodologias de desenvolvimento de software
Fonte: David Rico, 2010

Em 2009 realizaram um outro estudo onde o ROI ( ROI apresenta efectivamente uma estimativa das vantagens que um projecto poderá trazer) dos projectos com os métodos ágeis foram dez vezes maior do que com os métodos tradicionais. Sendo os métodos ágeis são 20 vezes mais produtivos, e teve cinco vezes melhor custo e qualidade do que os métodos tradicionais.

Um projecto ágil bem gerido traz benefícios impressionantes, em que os principais benefícios são o aumento da produtividade e a qualidade. A produtividade vem da sua natureza racional e qualidade da sua disciplina inflexível. No entanto, o seu poder real vem da

sua adaptabilidade à mudança, natureza colaborativa, e o foco nos resultados finais de negócios para o mercado (David Rico).

John Scumniotales também fez uma pesquisa em 2009, onde recolheu dados específicos que mostram os ganhos utilizando metodologias tradicionais (não incremental) versus uma metodologias ágeis (incremental). John publicou um artigo onde neste apresenta duas tabelas, uma nos mostra o ROI/NPV e a outra apresenta um gráfico onde nos apresenta os benéficos líquidos das duas metodologias.

| | Traditional | Agile |
|---|---|---|
| Year 1 ROI | 0% | 80% |
| Year 2 ROI | 72% | 210% |
| Year 3 ROI | 130% | 264% |
| Payback (months) | 30 | 14 |
| NPV | $663.287 | $2.271.181 |

Tabela 2: ROI entre as metodologias tradicionais vs metodologias ágeis
Fonte: John Scumniotales (2009)

Os valores apresentados nesta tabela foram com base nas entrevistas feitas nas empresas de desenvolvimento de software:

- Com base nos projectos de software (ágeis e tradicionais) que foram entregues e consideradas "perfeito", sendo que para cada uma das equipas (ágil e tradicional) o valor total distribuído é de 1 milhão de dólares;
- O valor real não é imediatamente apercebido na entrega do produto, visto que o valor do projecto tradicional no primeiro ano é realizado de forma linear após a conclusão do projecto. Enquanto que num projecto ágil, faz-se o planeamento do valor do projecto quatro meses após a sua iniciação;
- Depois de ser entregue todos os valores, o custo de manutenção que é de 20% anual, são atribuídos ainda por mais dois ou três anos.

De acordo com as melhores práticas ao nível da implementação dos projectos, entende-se como critérios fundamentais para o sucesso dos mesmos a sua entrega dentro prazo, dentro do orçamento e com os atributos e as funcionalidades originalmente atribuídas/definidas pelo Cliente.

A instituição The Standish Group foi uma das primeiras instituições a apresentar um relatório sobre os projectos TI, apresentando as percentagens dos sucessos, fracassos e os projectos que foram modificados. Em 2009 a instituição apresentou os principais factores que ajudaram nos sucessos dos projectos:

- Envolvimento do utilizador;
- Apoio executivo;
- Declaração clara dos requisitos e âmbito;
- Planeamento apropriado;
- Equipa competente;
- Visão e objectivos claros;
- Trabalho duro e equipa focada.

Em 2006 a *Computer World* apresentou um relatório em que os projectos com sucesso foram de 35% e os restantes 65% se dividiram entre os fracassos e os projectos mudados mas, em 2009 a instituição *The Standish Group* fez um estudo onde apresentou um relatório (chaos report) onde apenas 32% dos projectos obtiveram sucesso, enquanto que 24% foram de fracassos e os restantes 44% foram modificados.

Já com base nesses anos nota-se uma diferença, visto que o número de sucessos dos projectos diminuíram e os números dos fracassos aumentaram.

O seguinte gráfico mostra-nos a percentagens sucessos, os fracassos e os projectos modificados desde do ano de 1994 até ao ano 2010.

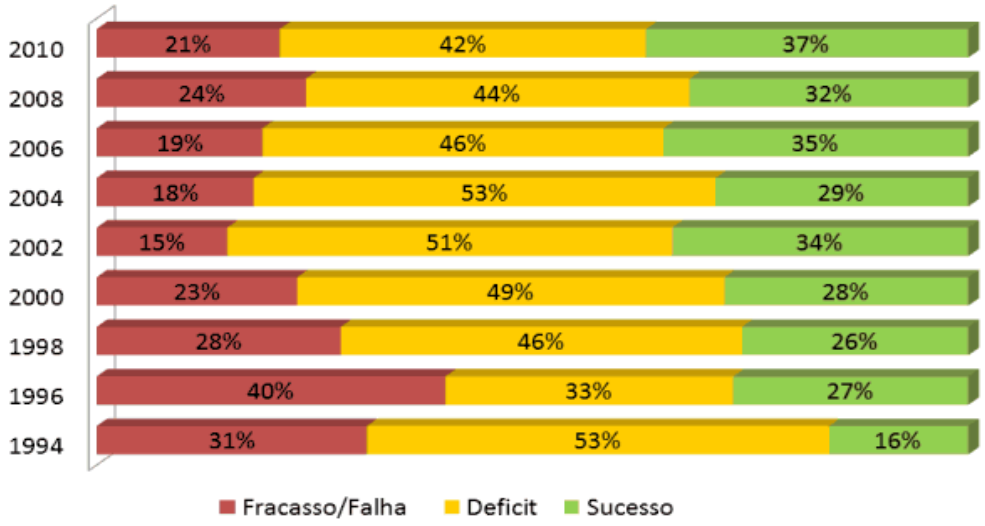



Figura 27- Taxas de sucesso, fracasso e modificados dos projectos TI
Fonte: The Standish Group: Chaos report 2011

Com base neste gráfico conclui-se que a taxa de sucesso dos projectos aumentou de 16% para 37% desde 1994 até 2010 e, a taxa de fracasso diminui de 31% para 21%. Já as taxas dos projectos modificados também baixou mas não o suficiente o quanto esperado.

Scott Ambler também fez um estudo onde apresentou um gráfico onde compara as taxas de sucessos, fracassos e modificações entre as duas metodologias desenvolvimento de software (ágeis e tradicionais). Este estudo foi realizado com base nos inquéritos feitos aos engenheiros de software, equipas de desenvolvimentos, etc., mas todos com uma larga experiencia no que diz respeito ao desenvolvimento de software.

Ambler realiza as suas pesquisas sobre os sucessos dos projectos com base em quatro factores que ele considera importante para que um projecto seja bem-sucedido:

1. Functionality (funcionalidade)
2. Quality (qualidade)
3. Time/Schedule (tempo)
4. Money (dinheiro)

Com base nesses quatros factores ele apresentou um gráfico que mede as taxas do sucesso dos projectos.

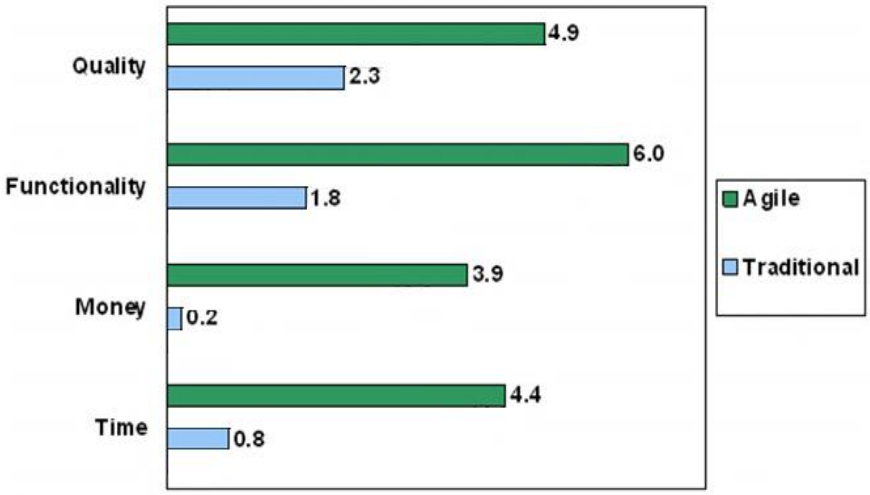



Figura 28 – Factores de sucesso das metodologias de desenvolvimento de software
Fonte: Scott w. Ambler 2010

No gráfico seguinte Scott apresentou as taxas de percentagens derivado ao ano 2010 das duas metodologias de desenvolvimento de software.

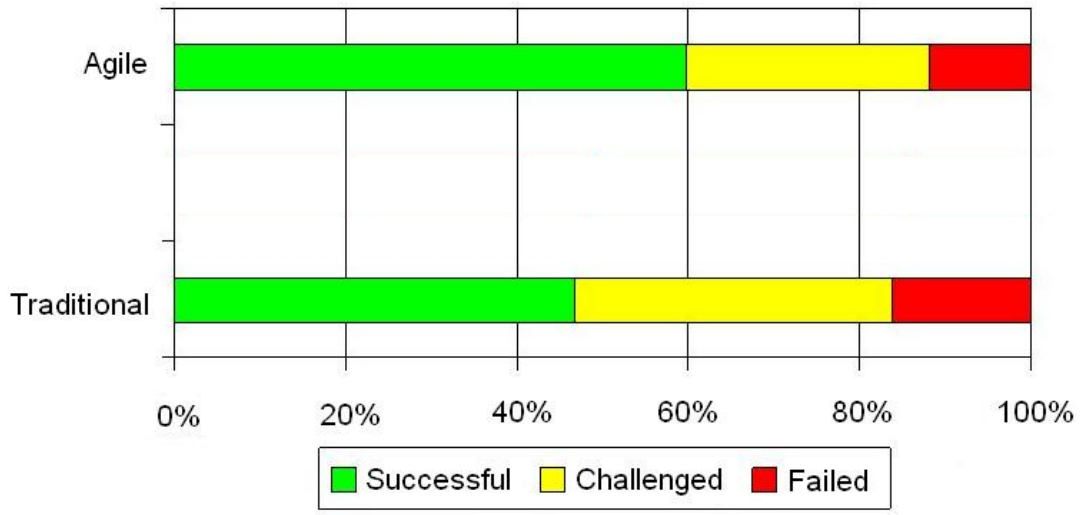



Figura 29 – Sucesso entre metodologia ágil x metodologia tradicional
Fonte: Scott w. Ambler 2010

Pode-se concluir que as taxas do sucesso das metodologias ágeis é superior as das metodologias tradicionais, visto que para alem de serem mais rápidas a apresentar resultados, fornecem rapidamente funcionalidades para utilizadores de negócios e sobre tudo valorizando a comunicação e colaboração de todos os membros de uma equipa que estão envolvidos no projecto, com o enfoque no rápido retorno de investimento e na qualidade do produto. Enquanto que nas metodologias tradicionais o custo das mudanças é muito elevado, o cliente quase não participa no ciclo de vida do projecto e isso muitas vezes faz com que o produto não seja do agrado do cliente ou os requisitos iniciais não se encontram no produto.

## *3.1. Conclusão do inquérito online sobre a validação dos métodos ágeis*

Neste trabalho foi feito um inquérito para validar a utilização e a adopção dos métodos ágeis dentro das organizações. O inquérito foi feito online, onde foi possível recolher informações sobre os quais nos dão a conhecer o porque da mudança dos métodos tradicionais para os métodos ágeis e, o que podem (as organizações) fazer para continuarem sempre ágil.

Segundo Svensson e Host (2005), a maior parte das pesquisas sobre implementação e o uso de métodos ágeis nas organizações são baseadas nas opiniões dos times que aplicaram os métodos.

Através do inquérito apercebe-se que muitas organizações começaram a adoptar os métodos ágeis a alguns anos como alternativa aos métodos tradicionais, com a implementação dos métodos ágeis tiverem bons resultados em relação ao tempo, satisfação dos clientes, qualidade dos produtos, etc. Muitos que responderam ao inquérito estavam satisfeitos com a implementação desses métodos.

O inquérito foi respondido por alguns gestores de projectos (alguns deles com alguns anos de experiencia em métodos ágeis e outros estão ainda a começar a utilizar os métodos ágeis) e não só.

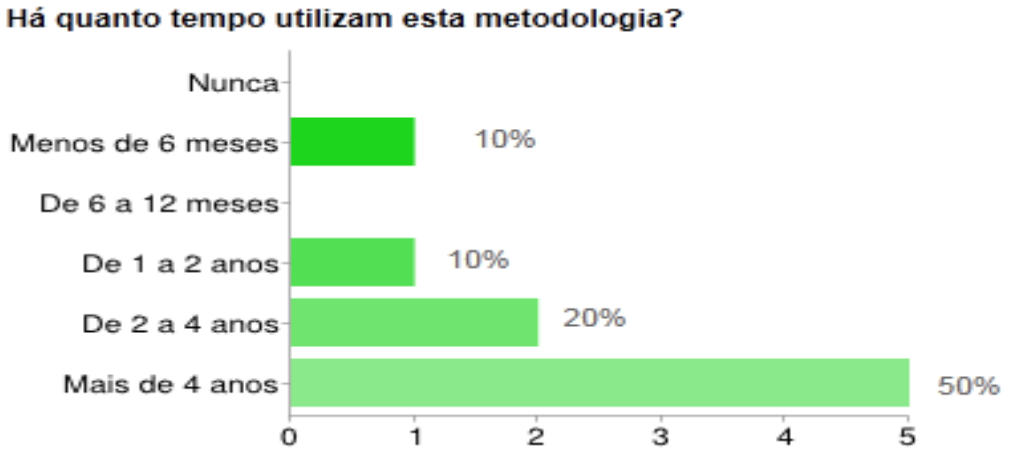



Figura 30 – Tempo de Utilização das metodologias ágeis

As empresas que contribuíram para esse inquérito foram: Xpand IT, Capgemini, Microsoft, Reditus, Adsoul, Strongstep-Innovation in Software Quality, Cotec, Normática, PMO, etc. Das 12 respostas obtidas só 10 foram consideradas, devido a respostas não válidas. Cada resultado estatístico será apresentado, acompanhado de alguns comentários e conclusões.

Figura 31 – Opinião sobre os métodos ágeis

Através do gráfico pode-se concluir que os projectos ágeis são mais rápidos de se concluir, obtendo 80% da resposta.

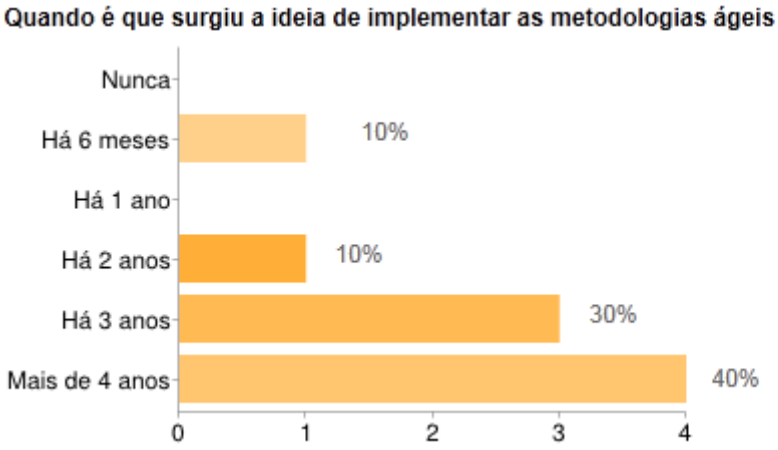



Figura 32 – Anos da implementação das metodologias ágeis

Nesse quadro os valores são claros, onde podemos ver a implementação das metodologias ágeis dentro das organizações é ainda "muito" recente. Existem empresas que já vem praticando e utilizando os métodos ágeis logo apos a sua apresentação no mercado mas também há muitas que ainda estão a descobrir e juntamente com isso estão a usufruir das vantagens desta metodologia.

Todas as empresas que contribuíram para inquérito actuam praticamente no mesmo mercado mas, cada um com o seu ramo específico como: serviços BI, SOA, Mobile, desenvolvimento de software a medida, comunicação e marketing e imagem, projectos transversais, etc.

As figuras abaixo indicam os principais motivos de mudança de metodologias tradicionais para as metodologias ágeis.

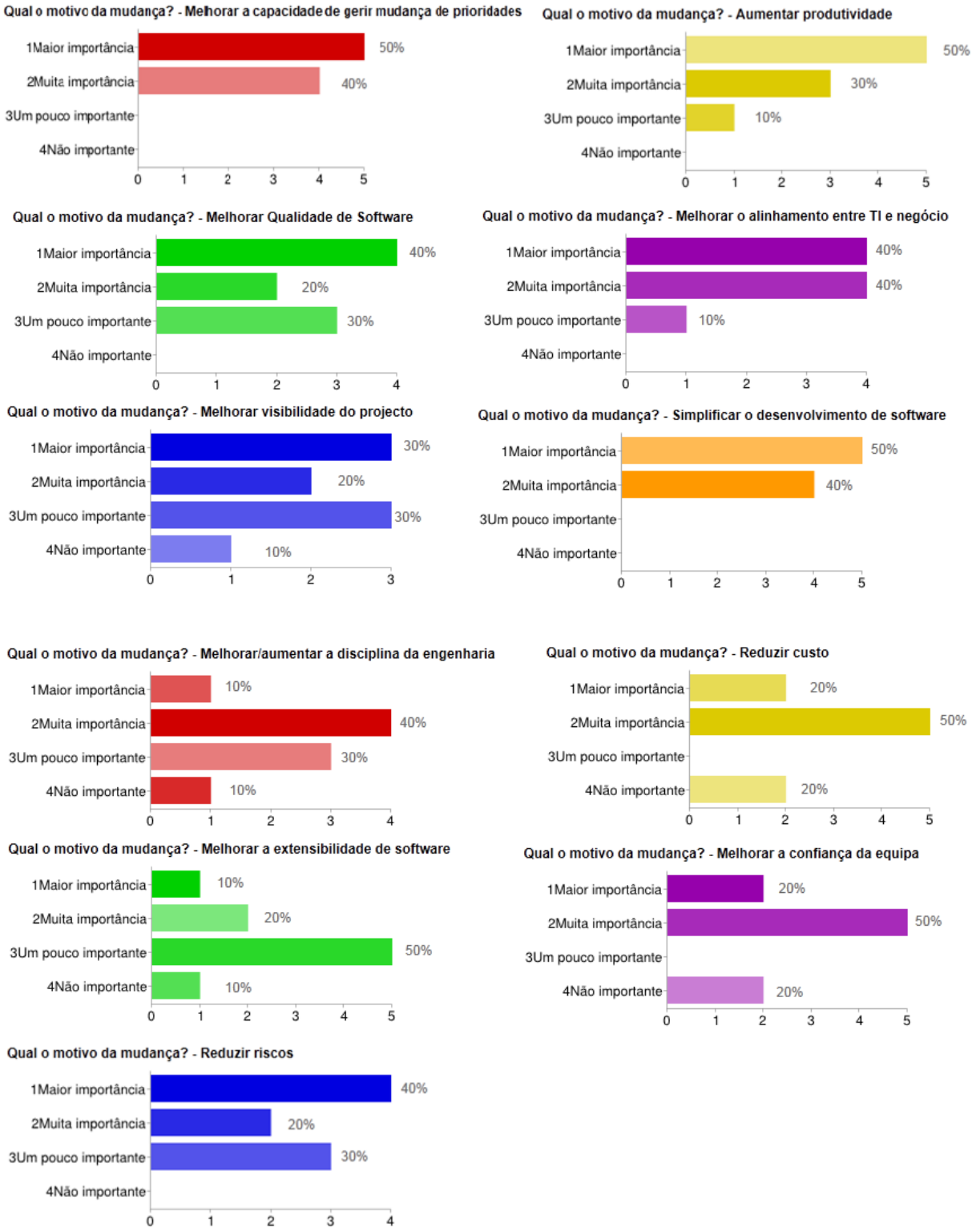



Figura 33 - Motivos que levaram a mudança de metodologias tradicionais para metodologias ágeis.

Apos a análise desses gráficos pode-se concluir que os motivos que levaram as organizações a mudarem para os métodos ágeis são as melhorias de capacidade na gestão de

mudança de prioridade, a melhoria da confiança da equipa, melhor extensibilidade do software, a redução de riscos e o alinhamento de TI e o negócio.

Todas as organizações antes de adoptarem os métodos ágeis utilizavam os métodos tradicionais, ainda apoiam em alguns métodos tradicionais para desenvolverem alguns projectos de grandes portes.

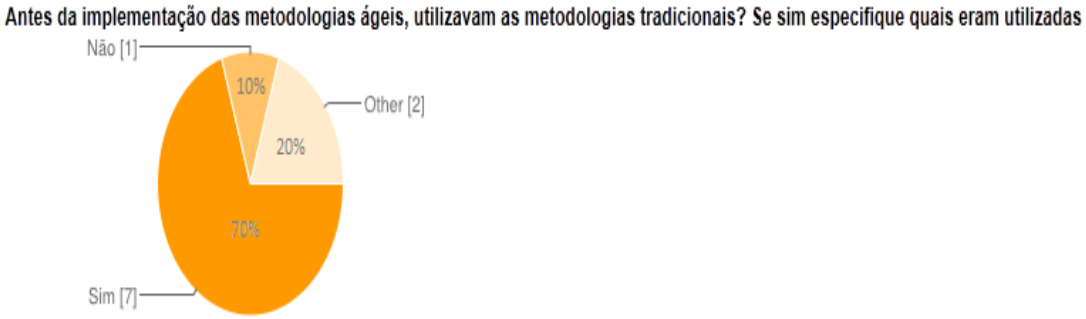



Figura 34- A utilização dos métodos tradicionais (cascata) antes dos métodos ágeis

Mais 50% dos entrevistados deixaram de utilizar por completo as metodologias tradicionais.

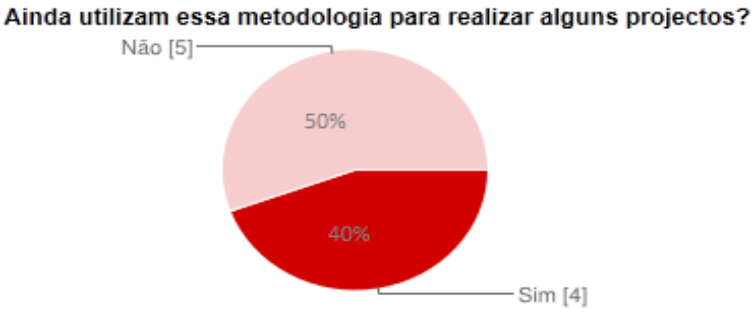



Figura 35 – Desenvolvimento de projectos com os métodos tradicionais

Como tudo sem o seu ponto positivo e negativo, a implementação das metodologias ágeis em muitos casos são de difícil implementação. Muitos afirmaram que ao implementarem os métodos ágeis pela primeira vez nas suas organizações, tiveram muitos desafios que levaram a perda do controlo de gestão, tempo de desenvolvimento, qualidade do produto, etc.

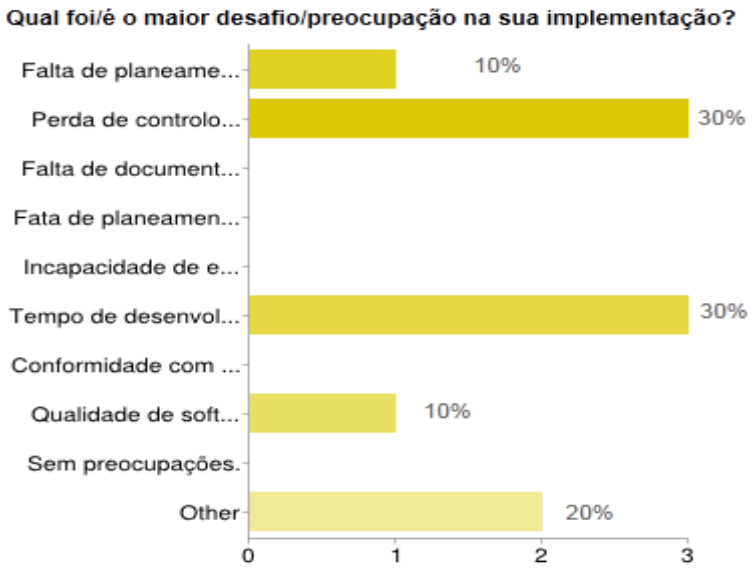



Figura 36 – Desafios na implementação das metodologias ágeis

Existem algumas dificuldades durante a implementação dos métodos ágeis dentro das organizações que é o apoio a gestão, e a percepção do tempo para a transição.

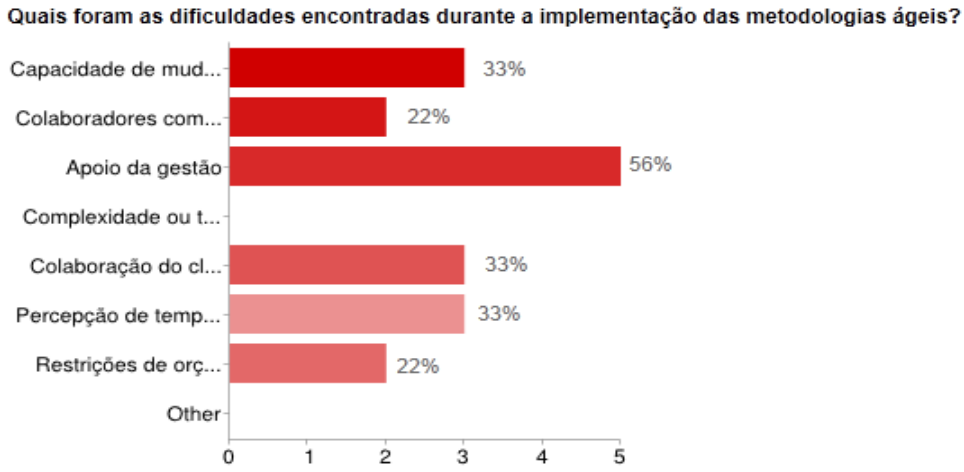



Figura 37 – dificuldades na implementação das metodologias ágeis

A implementação desta metodologia nas empresas requer alguns requisitos, algo dispendioso. Muitas empresas têm necessidade de despender muito dinheiro com a formação do pessoal, reorganização dos espaços e coaching, etc.

Apos a implementação dos métodos ágeis verificaram um aumento significativo de produtividade e reduziram os custos de produção.

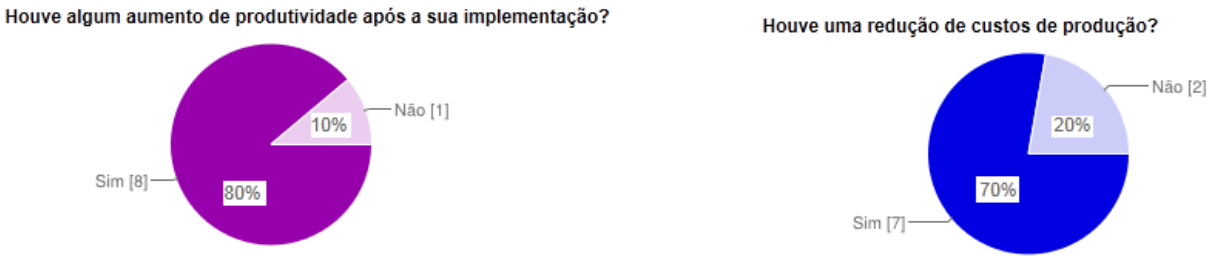



Figura 38 – Aumento de produtividade e redução de custos de produção na implementação dos métodos ágeis

Os métodos ágeis mais utilizados dentro as metodologias ágeis são a Extreme Programming, SCRUM e o Scrum/XP Hibrido.

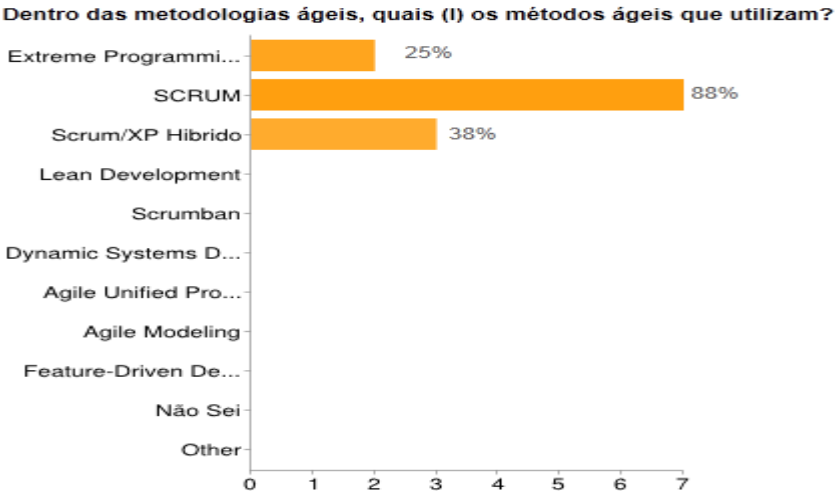



Figura 39 – Os métodos mais utilizados dentro das metodologias ágeis

As práticas ágeis são muito utilizadas dentro das organizações sendo que as mais utilizadas são os planeamentos de iteração, daily standup, gráfico burndown e o cliente presente em todas as actividades de desenvolvimento.

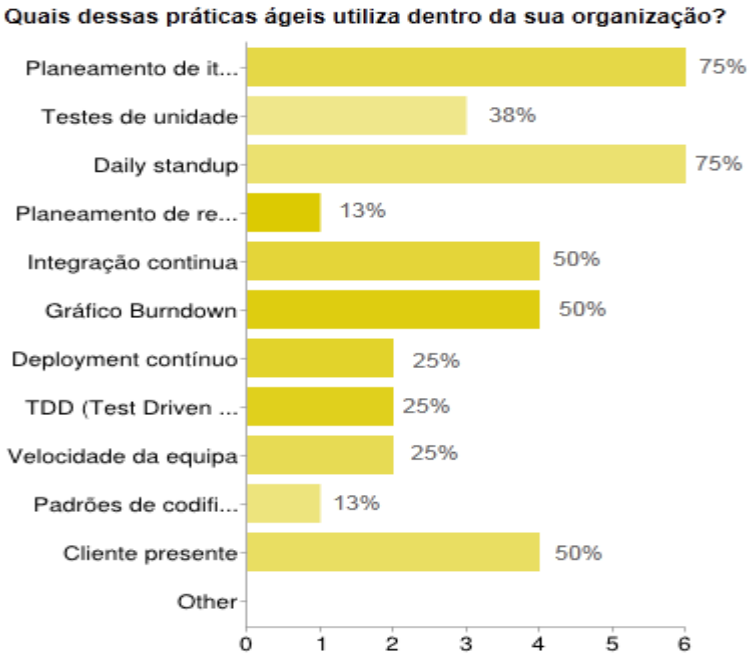
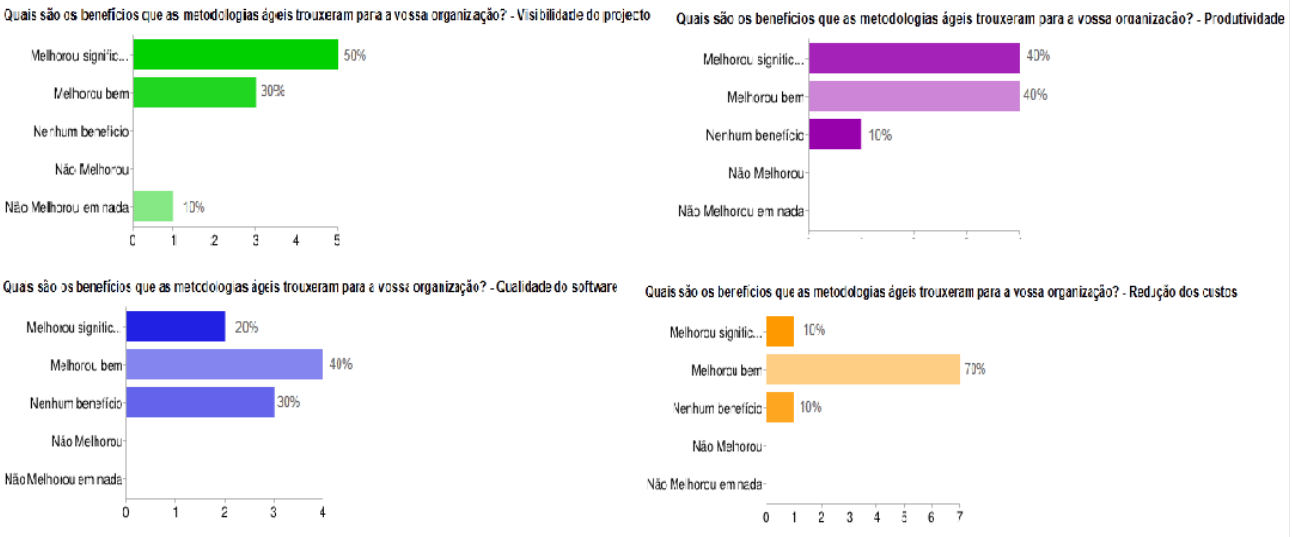



Figura 40 – As práticas ágeis mais utilizadas

Em termos de ganhos que as empresas ganharam com a implementação das metodologias ágeis foram: satisfação do Cliente, projectos mais baratos, mais projectos, melhor visibilidade do que se está a produzir, envolvimento real do cliente final, capacidade de se efectuar maior número de projectos por ano, equipas mais pequenas velocidade time-to-market e redução do número de projectos em que se verificou derrapagem.

Os benefícios adquiridos com a implementação das metodologias ágeis foram: visibilidade do projecto, redução dos custos, desenvolvimento de software mais simplificado, capacidade de mudança e principalmente a melhoria do moral da equipa.



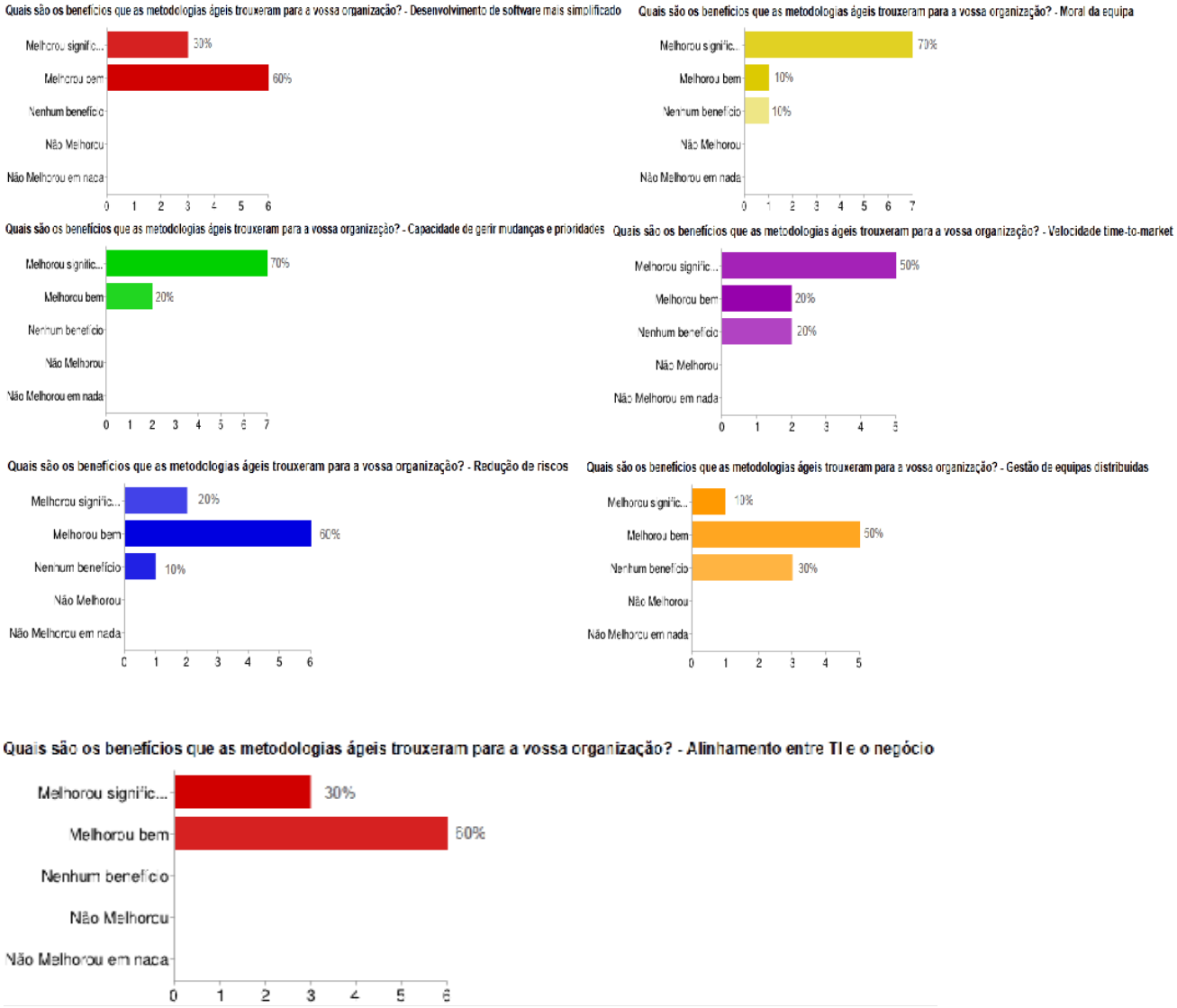



Figura 41 – Benefícios obtidos com a implementação das metodologias ágeis

## Conclusão

Qualquer organização, dependendo do projecto que desenvolve e independentemente de qualquer metodologia que utiliza, o seu foco principal é sempre a qualidade final do produto e a satisfação do cliente.

O presente trabalho fala sobre as duas metodologias de desenvolvimento de software mais utilizadas. Definiu-se cada uma delas, fazendo uma comparação as mesmas, descrevendo as suas características, os seus princípios, destacando principalmente qual das duas é que tem mais ganho de produtividade e sucesso.

As duas metodologias descritas apresentam aspectos positivos e negativos. Quando se está a pensar em desenvolver um projecto com base nas metodologias tradicionais é preciso ter em conta que, esta metodologia requer um planeamento forte e muita disciplina a qual, é baseada em documentações muito grandes. Nesse caso ela deve ser utilizada em projectos estáveis e, com os requisitos bem definidos e sem mudanças ao longo da vida do projecto. Já no que diz respeito as metodologias ágeis o processo é diferente, ela é mais adequada a projectos que estão em constante mudança, o cliente deve estar sempre presente e é necessário muita criatividade.

Podemos afirmar que, as metodologias ágeis já dominaram o mercado de gestão de projectos TI. As metodologias ágeis estão a ser aplicadas não só nos sectores privados como também nos sectores públicos (projectos dos governos, etc.). As metodologias ágeis são misturas entre as práticas das metodologias tradicionais com outras práticas modernas para criação de produtos e serviços inovadores. Elas são tidas como metodologias leves, flexíveis e adaptáveis as mudanças mas, também disciplinados.

A estrutura leve de gestão das metodologias ágeis, conduz a uma melhor produtividade e a uma maior eficiência na tomada de decisão, tornando as suas características de qualidades superiores e melhores do que as metodologias tradicionais. A interacção frequente entre os clientes e a equipa de desenvolvimento e os testes frequentes são tidos como um resultado muito positivo no mercado e, isso aumenta a satisfação do cliente melhorando a sua confiança, trazendo novos negócios. Todos esses benefícios dos métodos

ágeis traduzem em benefícios económicos como: aumento das vendas, aumento das receitas e rentabilidade global.

A gestão dos projectos ágeis é directamente focado na captura e, é implementado apenas no mercado onde há maior prioridade e nas necessidades dos clientes que resultam em retorno de investimento (ROI) para eles, ou seja, proporciona aos clientes um ROI maior, reduzindo a carga do trabalho operacional entre os membros da equipa. Isto melhora a sustentabilidade moral e o desempenho empresarial. (David Rico)

De acordo com esses pontos, pode-se concluir que as metodologias ágeis surgiram como alternativa para o ambiente actual das organizações. Os métodos ágeis apresentam actividades bastantes semelhantes em relação aos seus processos de desenvolvimento, mas algumas dessas actividades apresentam particularidades como por exemplo: a programação em dupla, as users stories escritas pelos clientes e a escrita dos testes antes da implementação do XP; o planeamento diário das reuniões e a revisão adoptado pelo Scrum*;* as inspecções de código de FDD, entre outros. Apesar de muitas vantagens, as metodologias ágeis tem as suas desvantagens como por exemplo o contacto constante com o cliente que nem sempre é possível manter, entre outros.

Por outro lado as metodologias tradicionais são tidas como métodos pesados, de muita documentação e burocráticos. A documentação da metodologia tradicional ajuda os desenvolvedores/pessoas a perceberem melhor o funcionamento do software. Elas são consideradas uma "óptima" solução para os projectos de grandes portes. Os seus métodos apresentam actividades, muitos semelhantes visto que todos baseiam no processo de ciclo de vidas para desenvolver todos os projectos, mas cada um tem as suas particularidades e os seus próprios princípios: apesar de ser uma metodologia de desenvolvimento iterativo o RUP concentra-se na redução dos riscos de projecto; o CMMI auxilia as organizações na melhoria dos seus processos de desenvolvimento e manutenção de produtos e serviços; o PSP resposta directamente a todas as necessidades, melhorando as praticas, técnicas e as habilidades individuais dos engenheiros/desenvolvedores de software; o modelo em cascata que serve como base para muitos projectos, é conhecido como ciclo de vida clássico e é um modelo linear e sequencia. Foi o primeiro método de desenvolvimento de software a aparecer.

Como desvantagem a metodologia tradicional temos o exemplo nos clientes visto que eles só vêem o produto final quando o projecto estiver terminado, ou seja, os clientes não acompanham o ciclo de vida do projecto.

Actualmente, o uso das metodologias ágeis nas empresas fornece aos desenvolvedores/engenheiros a sensação de que a organização confia nas suas competências, visto que juntando as práticas do Scrum, que focam totalmente nas iterações entre os indivíduos da equipa e, com isso tem-se um ambiente favorável ao crescimento profissional, onde há uma grande troca de conhecimentos devido a formação de equipas multidisciplinares.

Nos que diz respeito aos métodos tradicionais já o caso muda de figura visto que nem sempre as equipas trocam opiniões entre eles e, nem todos estão autorizados a darem as suas opiniões. Só há uma pessoa responsável pela equipa e ele também é conhecido como o chefe da equipa e gestor de todo o projecto. Muitas vezes as equipas não são valorizadas dentro das empresas, visto que são elas que levam as empresas a alcançarem os seus objectivos através de desenvolvimento de produtos cujos requisitos estipulados no início do projecto, no final quando o projecto ia ser entregue não correspondiam as verdadeiras necessidades dos clientes.

Com este estudo podemos concluir que para adoptar as metodologias de desenvolvimento de software, principalmente os métodos ágeis no processo de desenvolvimento de software, depende de vários factores como: a complexidade do produto de software, a maturidade e o comprometimento das equipas, a disponibilidade do cliente, a cultura da empresa, entre outros. A adaptação das metodologias de desenvolvimento de software varia de empresa para empresa e com a cultura de cada uma.

A implementação de uma metodologia ágil em um ambiente tradicional é muito complicado e requer todo o esforço de todos que estão envolvidos neste processo para que seja efectuado com sucesso.

**Limitações encontradas**

Apesar de muitas pesquisas sobre as metodologias de desenvolvimento de software e os estudos realizados sobre as mesmas, houve algumas limitações ao longo do desenvolvimento deste trabalho.

Primeiramente foi a dificuldade de acesso a algumas informações que são cruciais para a abordagens do ponto mais importante deste trabalho e a disponibilidade da parte dos gestores/desenvolvedores a darem os seus contributos sobre a validação dos métodos ágeis nas organizações. Mas, os resultados obtidos com a realização deste trabalho podem ser considerados satisfatórios e, acredita-se que contribuirá para que novos estudos sejam realizados com o intuito de se obter ou de se aproximar o mais perto possível dos métodos de gestão de projectos de software, tendo um conhecimento cada vez mais amplo sobre como utilizar estes métodos, podendo escolher o melhor e a que se adapte mais a estrutura da sua organização e não só.

**Trabalhos futuros**

Sugere-se como trabalho futuro um estudo mais aprofundado e detalho sobre as metodologias ágeis e especificando as suas ferramentas. Fazer uma pesquisa aprofundada sobre as técnicas dos testes ágeis para gestão de projectos, os users stories, planeamento ágil e estimativas ágeis.

Apesar das metodologias ágeis estarem a dominar o mercado, existem alguns pontos que são considerados fracos onde é preciso encontrar uma forma de os melhorar. Esses desafios são tidos como uma proposta futura para melhorar ainda mais o desempenho das metodologias ágeis, como a falta de análise de riscos sem torna-las metodologias pesadas, como por exemplo uma das preocupações dentro da XP é que não se tem a preocupação formal em fazer a análise e o planeamento de riscos, ou seja, deve-se, portanto, procurar implementar uma estratégia de gestão de riscos sem tornar a metodologia muito complexa.

Um outro desafio para o futuro é de fazer com que as metodologias ágeis sejam usadas em grandes empresas e em equipas grandes, uma vez que normalmente essas metodologias são baseadas em equipas pequenas. Neste caso, pelo menos é necessário

resolver os problemas de comunicação interno dentro da equipa, uma vez que é comum em grandes empresas os funcionários estarem separados geograficamente.

Quanto mais organizações usam as metodologias ágeis, melhores serão os resultados empíricos em termos de vantagens, desvantagens e procedimentos para sua adopção nas organizações, apesar de haver ainda casos de sucessos em projectos grandes e críticos. Mas, mesmo assim, os resultados iniciais em termos de qualidade, confiança, datas de entrega e custo são promissores.

## Bibliografia

PRESSMAN, Roger (2006) - S. Engenharia de *Software*. 6ª ed. São Paulo: McGraw-Hill, 2006.

SOMMERVILLE, Ian (2007) - Engenharia de Software. 8ª ed. São Paulo: Pearson Addison-wesley, 2007.

HIGHSIMTH, Jim (2002) - Agile Project Management. Boston: Pearson Addison-wesley, 2002.

MIGUEL, António (2003) - Gestão de projectos de software. 2ª ed. Actualizada e Revista. Lisboa: FCA – Editora Informática, 2003.

MIGUEL, António (2010) - Gestão de Projectos de Software. 4ª ed. Actualizada. Lisboa: FCA – Editora Informática, 2010.

LAROSE, Daniel T. (2006) - Data Mining: Methods and Models. 1ª ed. New Jerjey: John Wiley & Sons, Inc., Publication, 2006.

PONNIAH, Paulraj. (2010) - Data Warehousing: Fundamentals for IT Profissionals. 2ª ed. New Jerjey: John Wiley & Sons, Inc., Publication, 2010.

Ambler, S. – Agile adoption rate survey [consult. 5 Setembro 2011 a 15 Janeiro 2012] Disponível em WWW: <URL: http://www.ambysoft.com/surveys/>

CAMARA, Bruno. Desenvolvimento Ágil usando Scrum, 2005. [consult. 10 Outubro 2011].Disponível em WWW:< URL: http://iscte.pt/~mms/events/agile_seminar/apresentacoes/bruno_camara.pdf>

Rational Unified Process - Visão Geral. [consult. Outubro a Novembro 2011] Disponível em WWW: <URL: http://www.wthreex.com/rup/portugues/index.htm>

Versionone – Agile development Survey. [consult. Novembro 2011 e Janeiro 2012].Disponível em WWW: <URL: http://www.versionone.com/About_Us/ http://www.versionone.com/state_of_agile_development_survey/10/page3.asp >

Scrum Alliance. [consult. De Setembro a Novembro 2011]. Disponível em WWW: <URL: http://www.scrumalliance.org >

Manifesto for Agile Software Development. [consult. Outubro 2011]. Disponível em WWW: <URL: http://www.agilemanifesto.org >

WELLS, Don – Extreme Programming. [consult. Outubro de 2011 e Janeiro de 2012]. Disponível em WWW: <URL: http://www.extremeprogramming.org/ >

Heptagon [consultada em 2011]. Disponível em WWW: <URL: http://www.heptagon.com.br/fdd >

DSDM Consortium [consultada em 2011]. Disponível em WWW: <URL: http://www.dsdm.org/ >

SCRUM : A definição do método Scrum. [consultada em 2011 ]. Disponível em WWW: <URL: http://www.codeproject.com/KB/architecture/scrum.aspx# >

FRICO, David – Agile Survey. [consult. em Novembro 2011 e Janeiro 2012]. Disponível em WWW: <URL: http://davidfrico.com
http://davidfrico.com/rico07b.pdf >

SCUMNIOTALES, John – ROI das metodologias ágeis x tradicionais nas empresas. [consult. em Novembro 2011]. Disponível em WWW: <URL: http://agile.scumniotales.com/agile-roi/ >

EVELEENS, Laurenz; VERHOEF, Chris - The Rise and Fall of the Chaos Report Figures, 2010. [consult. em Setembro 2011]. Disponível em WWW: <URL: http://www.few.vu.nl/~x/chaos/chaos.pdf >

JACOBSON, Ivar - A resounding "yes!" to agile processes - but also to the "next big thing". [consult. em Novembro 2011]. Disponível em WWW: <URL http://www.ibm.com/developerworks/rational/library/2814.html >
CONTROL CHAOS - Origens do método Scrum. [consult. em outubro 2011]. Disponível em WWW: <URL: http://www.controlchaos.com/ >

BUZZLE – Intelligence life on the web. [consult. em agosto a novembro 2011]. Disponível em WWW: < URL: http://translate.google.pt/translate?hl=pt-PT&langpair=en%7Cpt&u=http://www.buzzle.com/articles/waterfall-model/ >

SATO, T. Danilo. Uso eficaz de métricas em métodos ágeis de desenvolvimento de software, 2007. [consult. em 2011]. Disponível em WWW: < URL: http://grenoble.ime.usp.br/~gold/orientados/dissertacaoDaniloSato.pdf >

WIKIPEDIA [consult. em 2011 e 2012]. Disponível em WWW: <URL: www.wikipedia.com >

ROYCE, Winston w. Managing the development of large software systems, 1970. [consult. em 2011]. Disponível em WWW: <URL: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf >