FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Integração de modelos de desenvolvimento de software mais e menos ágeis

Pedro Miguel Ribeiro Veloso Gomes

VERSÃO FINAL

Relatório de Dissertação
Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Carlos Pascoal de Faria (Doutor)

Co-orientador: Ademar Manuel Teixeira de Aguiar (Doutor)

Julho de 2009

Integração de modelos de desenvolvimento de software mais e menos ágeis

Pedro Miguel Ribeiro Veloso Gomes

Relatório de Dissertação
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Ana Cristina Ramada Paiva Pimenta (Professor Auxiliar)

_____

Arguente: José Carlos Nascimento (Professor Auxiliar)

Vogal: João Carlos Pascoal de Faria (Professor Auxiliar)

15 de Julho de 2009

Resumo

A utilização de sistemas de informação continua a ter um crescimento exponencial ao longo dos últimos anos e um impacto cada vez mais acrescido. Estes sistemas são intensivamente baseados em software, pelo que a gestão de projectos de desenvolvimento desta natureza assume uma importância acrescida.

A gestão de um projecto de desenvolvimento de software implica a utilização de práticas profissionais de gestão que controlem o âmbito, o custo e o prazo. Por outro lado, um projecto de desenvolvimento de software requer a utilização de práticas de engenharia eficientes, que garantam soluções de qualidade, sustentáveis e adaptadas à utilização social e humana.

Os dados de mercado mostram que continuamos a ter uma percentagem elevada de projectos de software cancelados e uma percentagem muito elevada de projectos que não cumprem com os objectivos de âmbito, prazo e custo estabelecidos inicialmente.

A engenharia de software é recente quando comparada com outras engenharias e tem sofrido evoluções constantes ao longo dos últimos anos. Surgiram múltiplos modelos e evoluções de modelos de desenvolvimento de software, que sendo de natureza distinta já demonstraram ter sucesso em contextos diversos.

Mais recentemente emergiram uma série de metodologias de desenvolvimento de software designadas de ágeis, que vêm dar ênfase na capacidade que um processo deve ter para absorver mudanças ao longo de um projecto e na vertente humana, de auto-organização da equipa e de contacto próximo com o cliente e de confiança com o mesmo.

Esta tese pretende trazer um maior esclarecimento sobre os métodos de desenvolvimento de software mais utilizados actualmente, os mais e menos ágeis, e mostrar como a sua combinação adequada pode trazer um valor acrescido na gestão de projectos de desenvolvimento de software. Esta combinação será instanciada através de um método que fornecerá critérios e pressupostos para a criação de uma metodologia adaptada às necessidades específicas de uma organização.

Abstract

The information system's usage has experienced an exponential growth over the past years as well an increasing impact on human life. These systems are much more software intensive than before and for that software project management presents itself as a crucial discipline.

The management of a software development project implies professional practices that assure the project's scope, cost and schedule control. On the other hand, a software development project requires efficient engineering practices that assure high quality solutions, sustained and adapted to social and human behavior.

Market figures show that we still have a big percentage of canceled software projects, and a very high percentage of those who fail to meet the initial objectives for scope, cost and schedule.

Software Engineering is recent, compared to other engineering disciplines, and it has suffered a constant evolution over the past years. Multiple models and evolution to the existent ones came up. Although presenting distinct natures, these models have proved their way in diverse contexts.

More recently a series of methodologies for software development emerged which put the emphasis on the capacity for a process to absorb changes, throughout the project and also on the human side, promoting self-directed teams, a trustworthy and close contact with the customer.

This thesis aims to bring a higher degree of clarification on the most used software development methodologies, more and less agile, and show how their combination can bring value added to the software development project management.

This combination will be instantiated on a method that will supply criteria and pre-requisites for a methodology adapted to the specific organization needs.

Agradecimentos

Quero agradecer aos meus orientadores, Pascoal Faria e Ademar Aguiar pelo apoio e conhecimento prestado.

Um agradecimento para o Jim Over do Software Engineering Institute (SEI), cujas conversas sobre engenharia de software são sempre inspiradoras.

Uma menção para Hilel Glazer do SEI que permitiu uma conversa muito interessante sobre a combinação de métodos ágeis e não ágeis – com uma perspectiva muito agnóstica e agregadora.

Aos meus sócios e equipa do FEUP CIQS (agora Strongstep), porque acima de tudo são meus amigos.

Por fim, e mais importante do que tudo, à Susana, aos meus pais e irmãos.

Pedro Gomes

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Definições

AGILE	Agilidade no desenvolvimento de software
BACKLOG	Termo utilizado nos métodos ágeis que designa um conjunto de funcionalidades a serem implementadas numa *release*
BURNDOWN CHART	Gráfico que foca no esforço a realizar num projecto
CMMI	Capability Maturity Model Integration
CMMI-ACQ	CMMI para aquisições de software
CMMI-DEV	CMMI para desenvolvimento de software
CMMI-SERV	CMMI para serviços de software
EARNED VALUE	Métrica de gestão de projectos que foca no trabalho realizado
ESI	European Software Institute
PAIR PROGRAMMING	Programação de software executada por dois programadores no mesmo terminal
REFACTORING	Redesenho de software
RELEASE	Resultado produzido que marca o final de uma iteração ou sprint
RUP	Rational Unified Process
SCRUM	Metodologia ágil de desenvolvimento de software focada na gestão do projecto
SEI	Software Engineerig Institute
SPRINT	Termo utilizado nos métodos ágeis que designa um intervalo fixo de tempo destinado ao desenvolvimento de funcionalidades
SPRINT RETROSPECTIVE	Termo utilizado nos métodos ágeis que designa uma reunião de lições aprendidas relativa a um sprint
SPRINT REVIEW	Termo utilizado nos métodos ágeis que designa uma reunião de apresentação das funcionalidades desenvolvidas

STANDUP MEETING	Termo utilizado nos métodos ágeis para uma reunião, que é feita em pé, de ponto de situação de desenvolvimento
TSP/PSP	Team Software Process / Personal Software Process
USER STORIES	Requisitos escritos de uma forma funcional simplificada em forma de história
XP	Extreme Programming - Metodologia ágil de desenvolvimento de software focada em técnicas de engenharias software

1 Introdução

Este trabalho é realizado no âmbito do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto e visa fornecer uma perspectiva sobre a utilização integrada e adequada de modelos de desenvolvimento de projectos de software mais e menos ágeis.

Pretende-se promover alguma clarificação sobre os diversos modelos existentes, a sua natureza, os seus pontos comuns e os seus pontos distintos.

Com a introdução dos modelos de desenvolvimento ágeis surgiu uma certa confusão na comunidade profissional, ao nível de gestores e técnicos sobre as características efectivas de cada modelo e o seu correcto modo de utilização. Este trabalho vai revisitar as características de cada modelo específico, dos mais proeminentes, e promover um método que permita a uma organização, ou a um projecto, conseguir tirar o melhor partido dos mesmos (ver Anexo A – Mapa conceptual).

Este trabalho promoveu uma recolha de referências sobre a literatura mais relevante para os tópicos em questão, nomeadamente através da consulta dos autores mais relevantes, dos artigos publicados nos últimos anos, com subsequente consulta de referências a outros artigos, conferências da especialidade, e com reuniões com algumas pessoas de referência do Software Engineerig Institute (SEI).

1.1 Enquadramento

"*There is no question that better management helps, but I soon realize that until we changed the practices of software professionals themselves, we could never achieve a truly expert software engineering capability.*[1]"

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

"Continuous attention to technical excellence and good design enhances agility. [2]"

[1] Winning with software – An executive strategy – Watts S. Humphrey – Addison Wesley 2002
[2] Agile Manifesto – http://agilemanisfesto.org – 2001

"*For any given project, the project manager, in collaboration with the project team, is always responsible for determining which processes are appropriate, and the appropriate degree of rigor for each process.*[3]"

Embora estas três frases estejam alinhadas no essencial do seu conteúdo elas provêm de correntes completamente distintas de metodologias de gestão. Provavelmente indica-nos que a sua essência provavelmente está correcta e que é necessário perceber qual a adequabilidade dos contextos de aplicação.

Boehm utiliza a metáfora do macaco e do elefante para ilustrar a diferença entre os métodos ágeis e não ágeis de desenvolvimento de software [Boehm2004]. O macaco que reage rapidamente às adversidade e consegue atingir pontos elevados da floresta e o elefante que suporta grandes pesos, e que depois de estar em andamento consegue percorrer grandes distâncias de uma forma robusta.

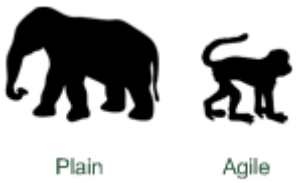


Figura 1 - Combinando práticas ágeis e não ágeis

Cada um destes animais está mais adaptado para determinadas situações específicas e em conjunto conseguem alcançar mais resultados, do que se operarem de uma forma separada.

O desenvolvimento de software apresenta-se como uma tarefa essencial no mundo actual, dada a dependência crescente na utilização de sistemas de informação. Esta dependência que é exponencialmente crescente, não está sustentada por processos de engenharia maduros, tais como os casos da engenharia civil, electrotécnica, mecânica, etc.

Na engenharia de software tem existido uma evolução nos modelos e boas práticas que regem o seu funcionamento. Estas boas práticas, que têm uma tradição da engenharia de sistemas promoveram a qualidade técnica dos produtos implementados, sustentada por processos de desenvolvimento.

Tais boas práticas, que tiveram origem para contextos específicos de desenvolvimento de software, promoveram a elaboração de documentação detalhada em cada projecto, para que a sustentação das decisões e o capital de conhecimento adquirido pudesse ser reaproveitado nas evoluções e nos projectos futuros. Por outro lado, os modelos de qualidade utilizados de base, com uma tradição histórica na indústria automóvel japonesa, estavam orientados para a optimização de processos que se baseavam em trabalhadores industriais por contraponto com a sociedade e trabalhadores do conhecimento actual. Aliado a estes factores, um último parece ter contribuído para o nascer de uma filosofia ágil de gestão e desenvolvimento de projectos, que é o facto de que os processos de desenvolvimento precisam de ser adaptados à realidade dos clientes, empresas e projectos específicos.

[3] Project Management Body of Knowledge 4th edition – Project Management Institute - 2008

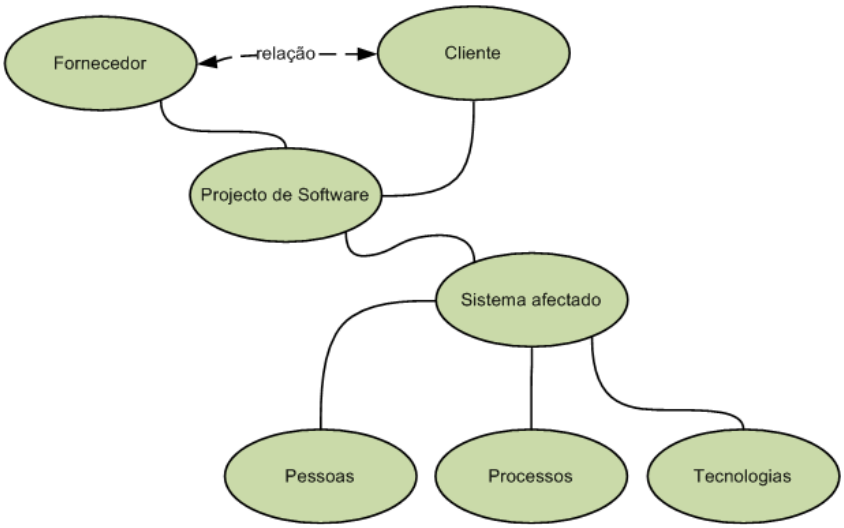


Figura 2 - Projectos de desenvolvimento de Software

Cada projecto de software envolve um fornecedor (ou mais) com determinada experiência, com determinados processos intrínsecos, envolve também clientes, com níveis de experiência diferentes, que por sua vez têm uma relação com níveis de maturidades específicos. Por outro lado, estas duas entidades, cliente e fornecedores, implementam um projecto de software que afecta determinado sistema, que envolve pessoas, processos e tecnologias. Os processos envolvidos podem ter diferentes graus de criticidade e dinamismo variáveis (Figura 2).

Dadas todas estas condicionantes envolvidas, existirá um único processo que responde de forma satisfatória a todo o tipo de projectos de software?

E o que se entende por responder de forma satisfatória a um projecto?

Tanto as comunidades de profissionais ágeis e menos ágeis parecem ser pouco conhecedoras das práticas que ambas advogam, tomando por isso perspectivas pouco agnósticas sobre a adopção de modelos de desenvolvimento. Por outro lado, alguns profissionais, parecem ter encontrado nas filosofias ágeis o pretexto ideal para deixar de efectuar algumas práticas que não estão relacionadas directamente com a codificação de produtos de software, que promovem uma gestão sustentada, quantitativa e assente numa visão de médio e longo prazo no desenvolvimento de software.

Por outro lado os movimentos ágeis vieram tornar mais explícito e trazer de novo para a ordem do dia a importância do factor humano, na gestão das equipas, na relação entre o cliente e o fornecedor (parceiro).

Este trabalho foi realizado no seio de uma organização que é um grupo de inovação em qualidade de software que se depara regularmente com os desafios de empresas de desenvolvimento de software, que tem parcerias com os maiores referências mundiais na área do desenvolvimento de software, nomeadamente o SEI e o European Software Institute (ESI).

Após uma experiência de gestão de dezenas de projectos de desenvolvimento de software, de diferentes dimensões, diferentes níveis de criticidade, com modelos de organização internacionais e com impactos organizacionais e sociais distintos, e após o contacto com dezenas de empresas de desenvolvimento de software, nacionais e internacionais, foi possível constatar que não existe uma orientação esclarecida sobre a forma de abordagem dos projectos de desenvolvimento de software, adaptada às diferentes realidades sociais e tecnológicas, à

relação entre os actores envolvidos, à maturidade e às características intrínsecas de cada projecto. Talvez como consequência da natureza humana, as organizações tendem a replicar fórmulas conhecidas, assentes sobre modelos em que se sentem confortáveis, para diferentes problemas e contextos.

1.2 Motivação e objectivos

A motivação deste trabalho é fornecer um contributo para a gestão do desenvolvimento de software através da indicação de formas de utilização e articulação de modelos de desenvolvimento de software mais e menos ágeis.

Este trabalho deve conseguir responder às seguintes questões:

- Quais as diferenças principais entre os principais modelos de desenvolvimento de software, mais e menos ágeis?
- Que tipo de modelo de desenvolvimento de software deve uma organização adoptar?
- Para que tipo de projectos se aplica cada tipo de modelo?

Para este objectivo foi efectuado um levantamento sobre os modelos de desenvolvimento mais utilizados a nível mundial, nomeadamente: o Capability Maturity Model Integration (CMMI), o Team Software Process (TSP/PSP), o SCRUM e o Extreme Programming (XP), consolidada numa análise detalhada comparativa.

Durante o levantamento foram efectuados alguns comentários que se focarão no levantamento de pistas para a integração e conjugação de modelos.

Foram também analisados os principais esforços da comunidade científica e empresarial, para a articulação dos diferentes modelos de desenvolvimento.

Baseada na informação recolhida é proposto um método que permite definir, de acordo com a natureza do sistema envolvido no desenvolvimento do projecto, quais os modelos de desenvolvimento a aplicar.

1.3 Estrutura da dissertação

Para além da introdução, esta dissertação contém mais 4 capítulos.

No capítulo “Gestão do desenvolvimento de software”, é descrito o estado da arte associado aos modelos de desenvolvimento de software e são apresentados respectivos trabalhos relacionados com a articulação entre os diversos modelos.

No capítulo “Análise comparativa e critérios ” é feita uma análise das principais práticas que podem afectar a decisão relativa ao planeamento da gestão do desenvolvimento de software.

No capítulo “Método de selecção de modelos de desenvolvimento de software” é consolidada a informação do capítulo anterior e é proposto um método, para apoio na selecção das práticas de desenvolvimento de software para um determinado projecto.

No capítulo “Conclusões e trabalho futuro”, são apresentadas as conclusões do trabalho e perspectivas de trabalho futuro.

2 Gestão do desenvolvimento de software

Neste capítulo são percorridas as maiores tendências metodológicas inerentes ao desenvolvimento de software, tanto nas áreas ditas mais como menos ágeis.

Em primeiro lugar será focada a crescente utilização de software no mundo actual e a necessidade de processo, pessoas e ferramentas capazes para o sucesso do projecto.

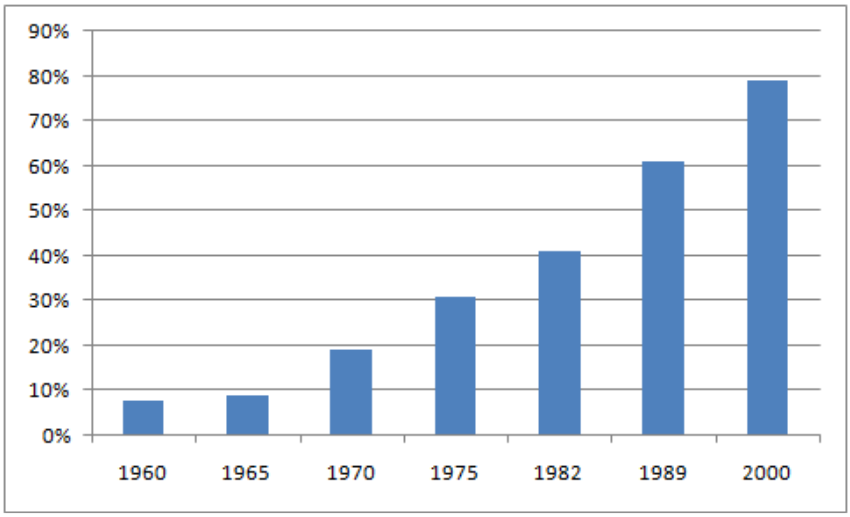
De seguida são analisadas as principais correntes associadas à gestão de projectos de desenvolvimento de software, começando pelo modelo CMMI focado na gestão de processos de uma organização, de seguida o PMBOK, orientado para as melhores práticas de gestão de projectos, de seguida é feita uma análise do TSP/PSP que pode ser considerada uma das metodologias mais maduras (grau de definição, implementação e gestão quantitativa) em termos de desenvolvimento de software, e por fim dois métodos de desenvolvimento ágeis: Scrum mais orientado para a gestão de projecto e o XP mais orientado para a engenharia.

Por fim é feita uma análise das iniciativas de combinação dos métodos mais e menos ágeis.

2.1 Introdução

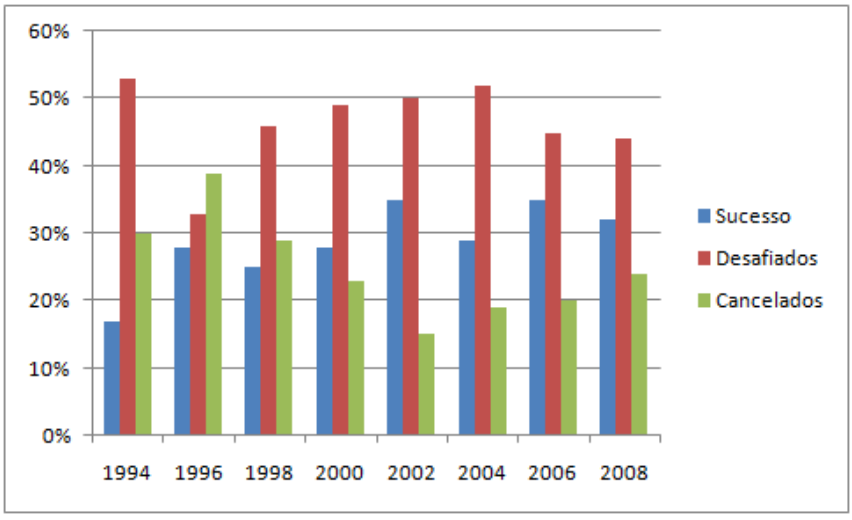
Cada vez mais o nosso mundo depende de informação e conhecimento, que são maioritariamente suportados por sistemas de informação e como tal o desenvolvimento e utilização de software são um factor exponencialmente crescente – veja-se o exemplo da utilização de software na aviação militar americana (Tabela 1).

Tabela 1- Percentagem de funções realizadas por software nos aviões militares [4]



Apesar da importância crescente do software nas organizações é espantoso verificar os números associados a projectos que terminam com sucesso, que são cancelados ou desafiados (Tabela 2).

Tabela 2 - Sucesso nos Projectos de Software [5]



- Aproximadamente 22% dos projectos de software em 2008 foram cancelados
- Aproximadamente 44% dos projectos de software em 2008 foram desafiados, ou seja, terminaram sem cumprir de uma forma satisfatória um ou mais dos objectivos âmbito, prazo ou custo.

Estes números, caso não sejam suficientemente esclarecedores, representam milhares de milhões de euros de perdas directas e outros mais de perdas indirectas.

[4] Winning with Software – An executive summary, Watts S. Humphrey, Addison-Wesley, 2002
[5] Standish group report – Successful software Projects - 2008

O número de projectos desafiados, que não cumpriram com o âmbito, custo ou prazo inicialmente definidos, tem mantido uma tendência estável com uma pequena variação, ao longo dos últimos 14 anos. O número de projectos com sucesso sofreu uma ligeira melhoria por contraponto com os projectos cancelados. Relativamente aos projectos cancelados, não concluídos, ou entregues e não utilizados, não deixa de ser estranho que nos últimos 6 anos tem existido uma tendência consistente de crescimento.

Estamos a falar de projectos de software com uma engenharia associada recente (software), que procura ainda encontrar o seu estado de maturidade.

O software é construído por pessoas que devem estar motivadas, essa disciplina é catalisada através de processos (formais e informais) e tornada mais eficiente através de ferramentas – são as três dimensões críticas em que as organizações se focam [Konrad2007].

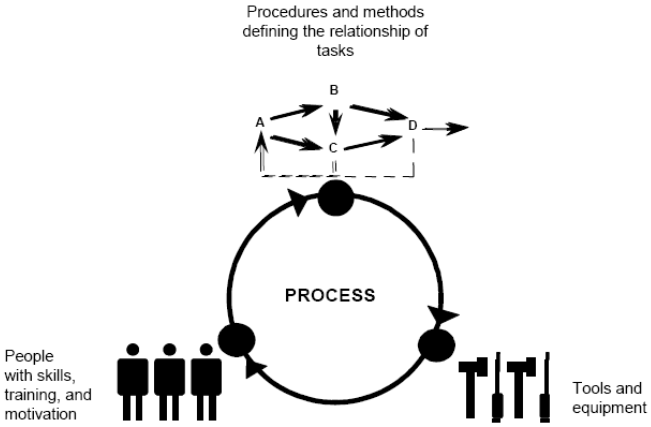


Figura 3 - Dimensões de gestão de organizações [6]

Quando não existem processos, o sucesso dos projectos depende em grande parte dos "bombeiros" das organizações, a visibilidade sobre o estado de um projecto é limitada, não há tempo para pensar em melhorias, provocando uma postura reactiva na organização, e por vezes os bombeiros queimam-se e as "cinzas" voltam a acender-se.

A existência de processos promove uma cultura de confiança na organização, através da previsibilidade dos procedimentos existentes, e permite uma gestão quantitativa, baseada em factos.

Watt Humphrey [Wats2002] definiu que para uma boa gestão de software devem ser observados três princípios:

1. Reconhecer que estamos no negócio do software
2. A qualidade deve ser a prioridade máxima
3. Software de qualidade é desenvolvido por pessoas disciplinadas e motivadas

Estes princípios reforçam a importância crescente que o software tem na sociedade actual, como a qualidade é fundamental na sua construção e como as pessoas são quem produz e utiliza o software.

[6] CMMI, 2nd Edition – Guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad and Sandy Shrum, Addison Wesley, 2007

2.2 CMMI

O Capability Maturity Model Integration (CMMI) é um modelo de maturidade para melhoria de processos de desenvolvimento de produtos e serviços [Konrad2007]. Está dividido em três constelações principais (Figura 4):

- CMMI-DEV – Modelo de processos para organizações que desenvolvem produtos e serviços.
- CMMI-ACQ – Modelo de processos para organizações que fazem aquisições de produtos e serviços.
- CMMI-SERV – Modelo de processos para organizações que disponibilizam e gerem serviços.

Embora pensadas não só para a indústria de software estas constelações permitem abarcar o ciclo completo de desenvolvimento de software, desde o seu desenvolvimento ou aquisição até a manutenção em produção. Neste trabalho será focado o modelo de desenvolvimento CMMI-DEV.

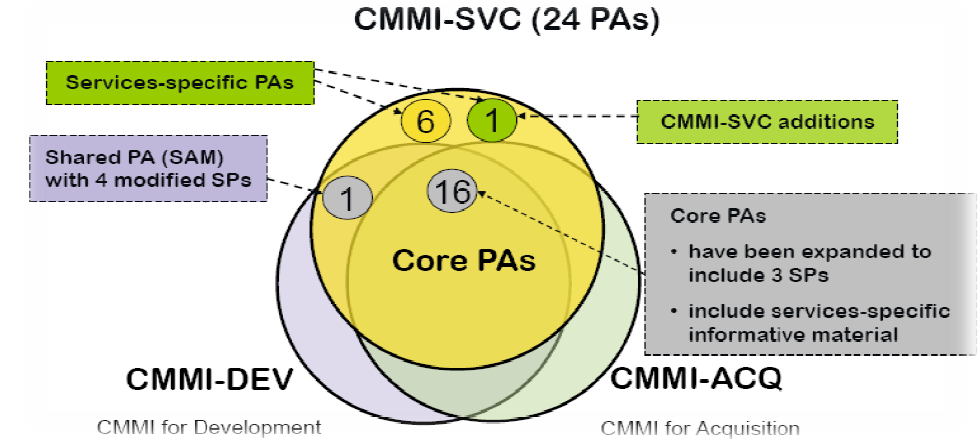


Figura 4 - CMMI - Framework de modelos

Este modelo de maturidade está dividido em áreas de processo que fornecem práticas processuais que as empresas devem adoptar, não prescrevendo no entanto a forma de como o devem fazer.

Este conceito é fundamental pois ao contrário do TSP/PSP, do Scrum ou do XP é um modelo não prescritivo, que permite que as práticas obrigatórias sejam instanciadas de formas distintas, nomeadamente combinando diversos modelos. É normal encontrar empresas que implementam o modelo CMMI e que ao mesmo tempo, tenham práticas de gestão de projectos alinhadas com o PMBOK, ou com normas de gestão de qualidade (ex: ISO 9001) ou com práticas ágeis.

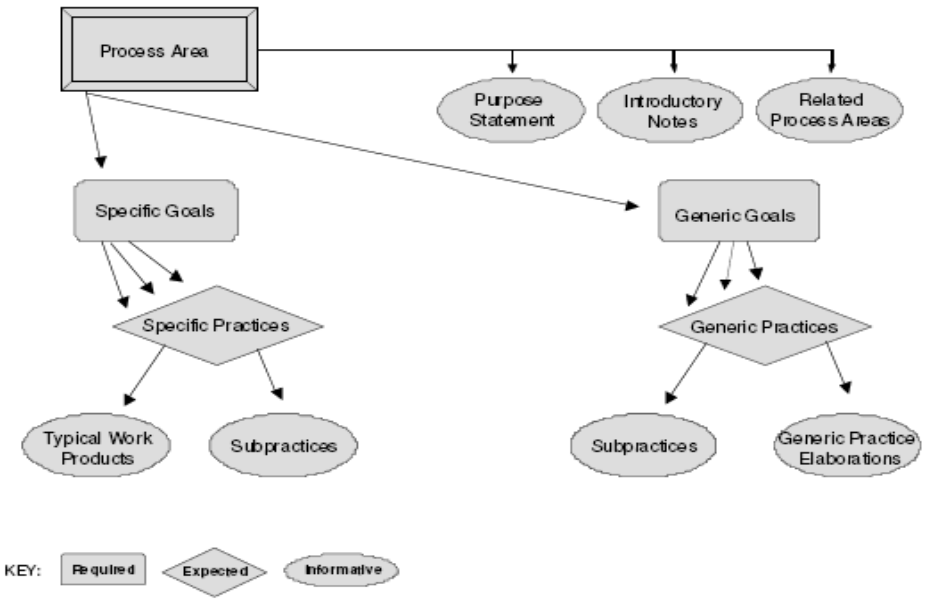


Figura 5 - CMMI - Estrutura das áreas de Processo[7]

As grandes áreas de processo do CMMI-DEV estão relacionadas com a organização, a gestão de projectos, engenharia e suporte.

A área de gestão de projectos contempla práticas de planeamento, monitorização e controlo, gestão de risco, gestão integrada, gestão das aquisições e gestão quantitativa de projecto.

A área de engenharia contempla a gestão e desenvolvimento de requisitos, a solução técnica (desenho mais codificação), a integração de produto, a verificação e a validação do produto.

A área de suporte prevê a gestão de configurações, métricas, análise e resolução de causas de problemas, análise e tomada de decisões, e controlo da qualidade dos produtos e dos processos da organização.

A força deste modelo está relacionada por um lado com a componente de gestão da organização, onde estão previstas práticas para a construção e definição de processos, alinhados com os objectivos estratégicos da empresa, gestão da formação, gestão da performance e gestão da inovação, e por outro com as práticas genéricas que definem o grau de institucionalização dos processos da empresa, ou seja, a forma como os processos fazem parte do dia-a-dia de cada colaborador - competência inconsciente da empresa (Figura 6).

[7] CMMI, 2nd Edition – Guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad and Sandy Shrum, Addison Wesley, 2007

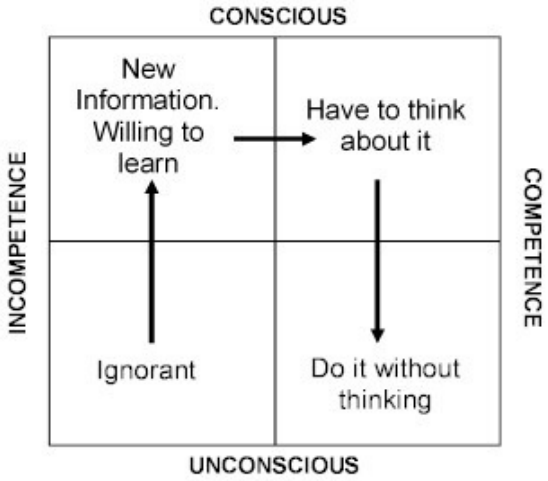


Figura 6 - Competência e consciência

Numa organização, mais importante que a definição de processos é a forma como os mesmos estão embebidos na cultura e nas actividades diárias da empresa.

Mais do que o conhecimento de engenharia e de gestão de projectos que o modelo CMMI aporta, a sua vertente de organização deve ser fortemente considerada na definição de processos de uma organização.

O elemento mais conhecido do CMMI é o facto de possuir uma representação de evolução baseada em níveis de maturidade (Figura 7):

1. Nível de maturidade 1 – Inicial (onde todas as empresas começam)
2. Nível de maturidade 2 – Gerido
3. Nível de maturidade 3 – Definido
4. Nível de maturidade 4 – Gerido quantitativamente
5. Nível de maturidade 5 – Em optimização

Os níveis de 4 e níveis 5 prevêem uma gestão quantitativa das áreas de processo.

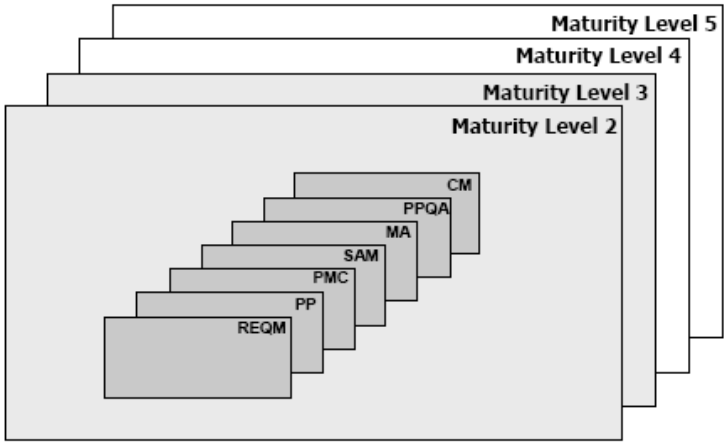


Figura 7 - CMMI - Níveis de maturidade

Este elemento permite um caminho de evolução para as empresas, com uma recomendação de prioridade de implementação de práticas, permite uma forma de benchmarking entre empresas e um grau de reconhecimento de mercado por parte dos contratantes.

Existe também uma representação contínua de evolução, que não prescreve áreas de processo prioritárias, ficando ao critério da organização a evolução desejada, área de processo a área de processo. Normalmente é aconselhável uma mistura das duas representações na implementação de processos nas organizações, dado que permite por um lado uma prioridade de implementação alinhada com as necessidades da organização com uma consequente possível certificação reconhecida pelo mercado.

A comunidade científica e o Software Engineering Institute abordam cada vez mais a implementação conjunta, quando aplicável do CMMI e das práticas ágeis [Alegria2006], [Glazer2008], onde as principais conclusões se focam na complementaridade e na necessidade de definição dos contextos adequados de utilização.

2.3 PMBOK

A gestão de projectos visa a concretização de um âmbito determinado, com um custo ou investimento definido num prazo determinado [PMBOK2008].

A decisão de se avançar com um determinado projecto está intimamente ligada a retorno previsto de investimento. Não basta portanto controlar uma das variáveis: âmbito, prazo e custo para que se possa considerar um projecto com sucesso. Adicionalmente podemos acrescentar os parâmetros qualidade e risco como variáveis importantes de controlo: a qualidade porque diferentes tipos de projecto podem requerer diferentes abordagens de controlo de qualidade (quando os projectos afectam vidas humanas, ou negócios de forma estrutural); o risco porque, por exemplo, por vezes certos projectos não podem ultrapassar, imperativamente, determinada data ou determinado orçamento.

O Project Management Body of Knowledge é hoje o *standard* mais utilizado em termos mundiais para a gestão de projecto. Este modelo é suficientemente abrangente para funcionar para qualquer área de negócio, desde que complementado com práticas específicas do negócio ou engenharia em questão.

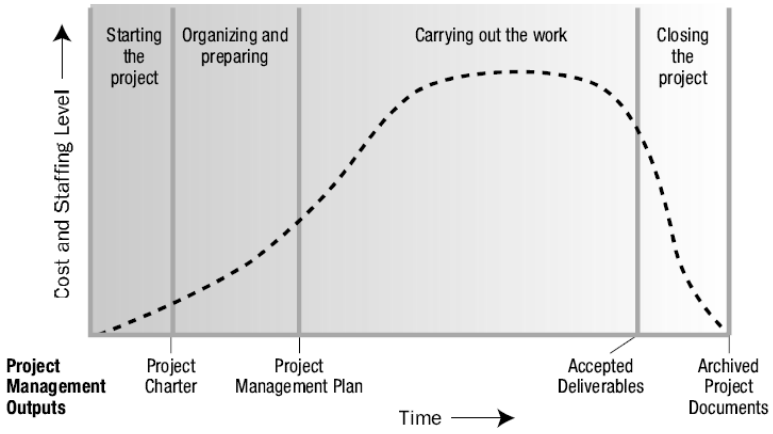


Figura 8 - Ciclo de vida de um projecto - PMBOK

O ciclo de vida nos projectos no PMBOK compreende 5 fases principais com alguma sequência mas que se podem sobrepor no tempo:

- Iniciação: onde se tomada a decisão efectiva de avançar com o projecto e se faz o arranque formal do mesmo.
- Planeamento: onde se planeiam todas as actividades do projecto.
- Execução: onde as actividades são executadas.
- Monitorização e Controlo: onde se processa a monitorização e controlo das variáveis de gestão do projecto.
- Fecho: onde se procedem às actividades de fecho do projecto.

Para as diversas fases do projecto estão previstas as seguintes actividades de gestão:

- Integração do projecto – controlo das actividades de ciclo de vida de projecto, bem como a integração com todas as actividades necessárias.
- Âmbito – controlo do âmbito global do projecto - não apenas requisitos.
- Custo – controlo do esforço de recursos humanos e custos de aquisições.
- Tempo – controlo das estimativas, prazo.
- Comunicação – controlo da comunicação entre os diversos *stakeholders*.
- Recursos Humanos – controlo da contratação das equipas.
- Aquisições – controlo de aquisições do projecto.
- Qualidade – controlo da qualidade dos produtos entregues.

Esta metodologia é importante pois é suficientemente completa para abarcar todas as actividades inerentes à gestão de projecto e por outro lado permite estar associada a diferentes métodos de engenharia, sejam eles de software ou não.

2.4 TSP/PSP

O Team Software Process (TSP) em conjunto com o Personal Software Process (PSP) foram desenhados pelo SEI com o objectivo de fornecer uma estratégia e um conjunto definido de procedimentos operacionais, para a implementação de métodos de desenvolvimento de software disciplinados, tanto ao nível do indivíduo como ao nível da equipa [Watts2002]. Estas metodologias devem ser sempre implementadas em conjunto.

O foco principal do TSP/PSP é a produção de software com zero defeitos, de acordo com os requisitos do cliente e no prazo estipulado. Caracteriza-se por utilizar métodos quantitativos na sua gestão, o que permite de forma rigorosa estimar e atingir os objectivos propostos como garantir qualidade de vida às equipas envolvidas.

Segue o seguinte princípio base de funcionamento: É necessário pensar antes de executar, e depois de executar é necessário rever, inspeccionar e testar. Este princípio garante eficiência no processo, nomeadamente quando a dimensão, dispersão e criticidade dos sistemas envolvidos aumentam. É um processo bastante prescritivo e bastante detalhado nos procedimentos que promove de forma a garantir uma disseminação de práticas efectiva entre todos os elementos das equipas.

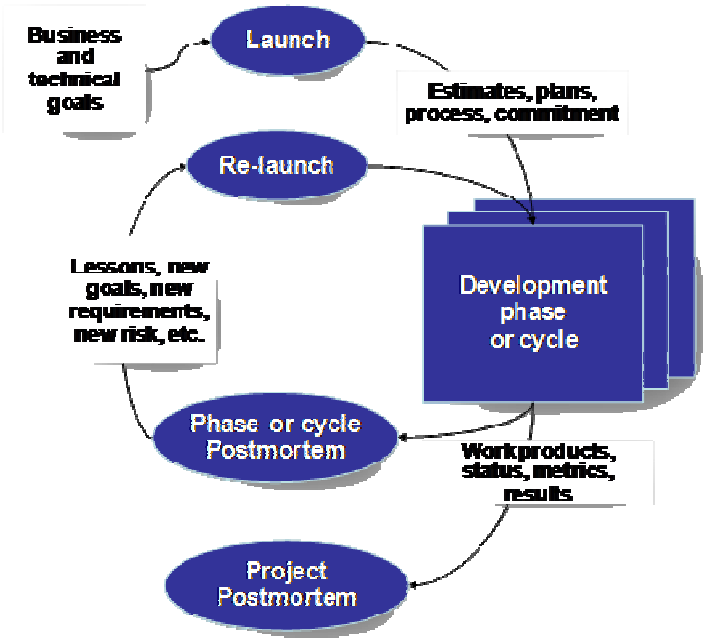


Figura 9 - TSP/PSP - Ciclo de vida

A sua estrutura base é iterativa, prevendo um processo de lançamento inicial (*launch*), para um planeamento macro do projecto e organização da equipa, seguido de relançamentos (*Re-launch*) para cada iteração (Figura 9).

O processo de lançamento que demora tipicamente 4 dias e tem como principal objectivo o planeamento detalhado da iteração (com uma duração típica de 1 a 4 meses) com a construção de uma equipa motivada, envolvida e comprometida com os resultados previstos (Figura 10).

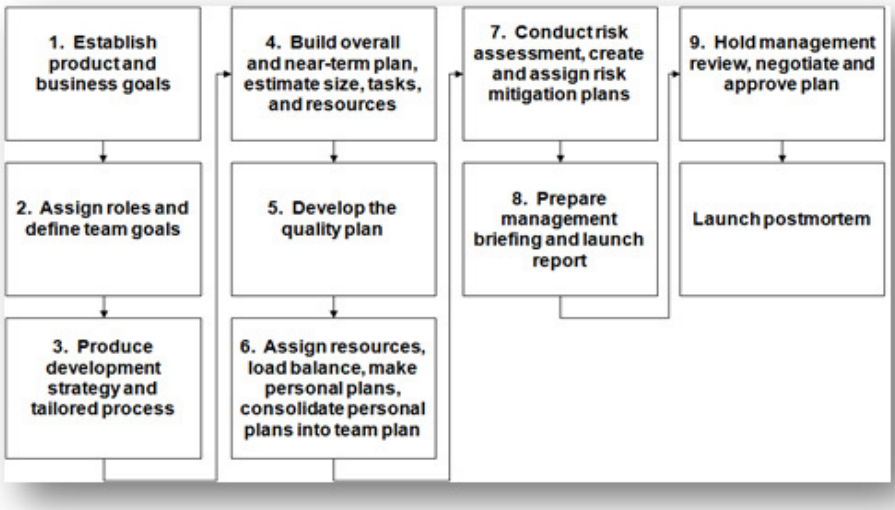


Figura 10 - TSP/PSP – Lançamento de iteração

Este lançamento [Humphrey2000] contempla a definição conjunta dos objectivos do produto e de negócio do projecto, atribuição de papéis e objectivos da equipa, a produção de uma estratégia e de um processo adaptado à realidade do projecto, definição de uma plano global e de um plano mais detalhado para a próxima fase, o desenvolvimento de um plano de qualidade, a atribuição de tarefas e balanceamento de carga, uma avaliação formal do risco e criação de respectivo plano de mitigação, preparação e apresentação do plano à gestão de topo e um *post mortem* do próprio lançamento.

A fase de desenvolvimento contempla as seguintes actividades embora possam ocorrer em iterações distintas:

- Captura de requisitos e respectiva revisão e inspecção
- Desenho de alto nível e inspecção
- Desenho de baixo nível, revisão e inspecção
- Codificação, revisão, testes unitários e inspecção
- Testes de integração
- Testes de sistema

A razão da utilização obrigatória de revisões e inspecções tem a ver com a eficiência que estes métodos possuem, face a testes realizados mais tarde no processo (será mais à frente detalhado). A revisão é feita pelo autor do artefacto em causa (código ou documento), enquanto que a inspecção é efectuada por outros membros da equipa.

A gestão quantitativa do processo é conseguida através de ferramentas que analisam a informação de quatro métricas base: tamanho, tempo, prazo e defeitos. Assim, cada elemento da equipa, através de processos preferencialmente automáticos deve fazer o registo e classificação de todos os defeitos que encontrou, o registo do tempo que demorou em cada tarefa, e o processo de estimativa é realizado através da avaliação do tamanho previsto do produto. O acompanhamento do projecto é efectuado através de algumas medidas entre as quais: *earned value,* acompanhamento do esforço realizado e previsto de recursos, perfil de qualidade, quantidade de defeitos eliminados em revisões e inspecções.

A utilização da métrica, tamanho do produto, permite obter uma correlação entre o tempo que foi previsto e o tempo efectivamente realizado para determinada dimensão de esforço. O processo de estimativa segue um método designado por PROBE – *proxy based estimation* – que primeiramente identifica quais os componentes relevantes para estimativas, para a tecnologia ou domínio concreto de desenvolvimento: classes, *forms*, *triggers*, funções, etc. De seguida as estimativas são efectuadas sobre o número de componentes que se prevê realizar, sendo que se classificam com três graus de complexidade. Para obter o tempo previsto para a realização dos componentes são utilizados os valores históricos do indivíduo, ou não existindo, valores das equipas, da organização ou médias de mercado, para a realização de cada tarefa. Desta forma científica, o grau de estimativa oferece um grau de precisão muito elevado.

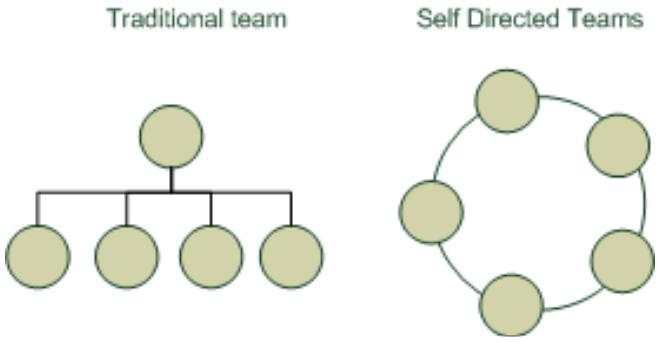


Figura 11- TSP/PSP - Equipas

As equipas TSP/PSP são auto dirigidas (Figura 11) na medida em que todos os elementos da equipa participam em todas as actividades de planeamento, monitorização e execução dos trabalhos, não sendo apenas dirigidas como nas equipas mais tradicionais. Uma equipa TSP/PSP prevê uma série de papéis que podem ser acumulados por uma única pessoa, sempre que necessário [Humphrey2005]:

- *Team leader* – com responsabilidade por garantir os objectivos do projecto, manter a equipa motivada e o cliente satisfeito.
- *Coach* – com responsabilidade por garantir e treinar os elementos no desenvolvimento das actividades TSP/PSP, normalmente é externo à equipa.
- Gestor de planeamento – com responsabilidade pelos planos do projecto e acompanhamento das actividades individuais e do projecto.
- Gestor de processo – responsável por ajudar toda a equipa em cumprir o processo conforme foi definido.
- Gestor de qualidade – garantir que a equipa segue o plano de qualidade e acompanhar os indicadores respectivos.
- Gestor de suporte – responsável por ajudar na correcta utilização de ferramentas e gestão das configurações.
- Interface com o cliente – garantir o alinhamento do projecto com os objectivos do cliente.
- Gestor de desenho – garantir um desenho de grande qualidade, envolvendo toda a equipa e fomentando a produção de boa documentação.
- Gestor de implementação – garantir uma boa implementação dos desenhos definidos e com qualidade.
- Gestor de testes – garantir que todos os membros criam e executam os planos de testes adequados.

2.5 Modelos ágeis

Os modelos ágeis de desenvolvimento de software foram primeiramente introduzidos através de Mikio Aoyama em 1998 junto das fábricas de software japonesa [Aoyama1998]. Agilidade significa uma boa capacidade de adaptação a requisitos e contextos envolventes e por isso é caracterizada pelos seguintes atributos:

1. Processo de desenvolvimento incremental e evolutivo
   Os processos menos ágeis tendem a fazer uma entrega total no final do projecto enquanto os métodos ágeis promovem entregas constantes e incrementais do produto, promovendo a emergência de novos requisitos e soluções.

2. Processo modular e simples (*lean*)
   Os processos devem ser simples e modulares de forma a poderem responder rapidamente - pouco baseados na produção de documentação.

3. Orientado ao tempo
   Nos processos ágeis o planeamento é orientado a intervalos fixos de tempo (iterações) e não ao volume de esforço necessário.

Estes atributos devem ser vistos e implementados como um todo e não de forma separada para que os processos possam ser considerados ágeis. Estes métodos são orientados às pessoas e menos aos processos, dado que cada indivíduo é responsável pela entrega de uma funcionalidade completa. Nesta abordagem inicial de métodos ágeis [Aoyama1998] foi referida a componente da possibilidade do desenvolvimento distribuído mas não foi a da articulação com o cliente.

Em 2001, uma série de signatários juntou-se para lançar um manifesto, que impulsionou de forma significativa o movimento dos métodos ágeis, através do Manifesto para o desenvolvimento ágil de Software [AgileMan2001]. A força deste manifesto reside também no facto de ter sido elaborada por personalidades proeminentes, relacionadas como desenvolvimento de software.

"*Manifesto for Agile Software Development*

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

Through this work we have come to value:

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

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

Face às ideias apresentadas em 2001 sobressai o ponto de colaboração com o cliente em relação à negociação do contrato. Este ponto é interessante pois é uma área de desafio forte para estas metodologias em relação ao trabalho distribuído e relação com o cliente.

De acordo com os valores apresentados pelo Manifesto foram estabelecidos 12 princípios que devem ser seguidos para que os processos de desenvolvimento sejam ágeis, apresentados de seguida de forma simplificada:

- As mudanças de requisitos são bem recebidas pois representam vantagens competitivas para os clientes.
- Devem existir entregas de software numa base semanal ou mensal.
- As pessoas de negócio e os desenvolvedores devem trabalhar conjuntamente e diariamente durante todo o projecto.
- O trabalho deve ser realizado por indivíduos motivados, sendo que para isso devem ser providenciadas as condições e o suporte necessários, bem como deve ser estabelecida uma relação de confiança.
- A forma mais eficiente de comunicação para e com o projecto é a conversação cara a cara.
- A principal medida de progresso é software que funciona.
- Os processos ágeis promovem um desenvolvimento sustentado. Os *sponsors*, desenvolvedores e utilizadores, devem estar habilitados para manter o mesmo ritmo de desenvolvimento de forma indefinida.
- Uma atenção contínua a bom desenho de soluções e excelência técnica são catalisadores da agilidade.
- A simplicidade é fundamental de forma a minimizar o trabalho a realizar.
- As melhores arquitecturas, requisitos e desenhos emergem de equipas que são autónomas, que se auto-organizam.
- Em intervalos regulares as equipas reflectem em como podem ser mais eficazes e eficientes ajustando os seus processos e métodos de acordo com as reflexões.

Seja qual for a metodologia de desenvolvimento de software, a aplicação da maior parte destes princípios é consensual e não representam algo de inovador naquilo que foi a evolução da história do desenvolvimento de software na sua componente de gestão de projectos e engenharia. A forma como são implementados os princípios é que pode por vezes suscitar alguma discussão. Por exemplo, é consensual que a forma mais eficaz de comunicação é a conversação cara a cara. Dizer isso não é dizer que não deve existir documentação adequada, para apoiar as comunicações entre centenas de colaboradores, ou para manter um histórico – que permite tornar eficiente evoluções futuras ou registar decisões estratégicas relevantes.

O princípio mais desafiador é o da obrigação da localização conjunta dos elementos da equipa, negócio e desenvolvedores, durante todo o projecto.

2.6 Scrum

O Scrum nasceu de um conceito importado do desporto *rugby*, onde é dado ênfase à comunicação entre a equipa (passes laterais e para trás) e ao avanço por *sprints*. É sobretudo um modelo que foca na gestão de projectos e não tanto em técnicas de engenharia de software. Schwaber que desenvolveu o Scrum referiu que para se ser verdadeiramente ágil, um processo precisa de aceitar as mudanças em vez de procurar controlar a previsibilidade [Schwaber, 2002].

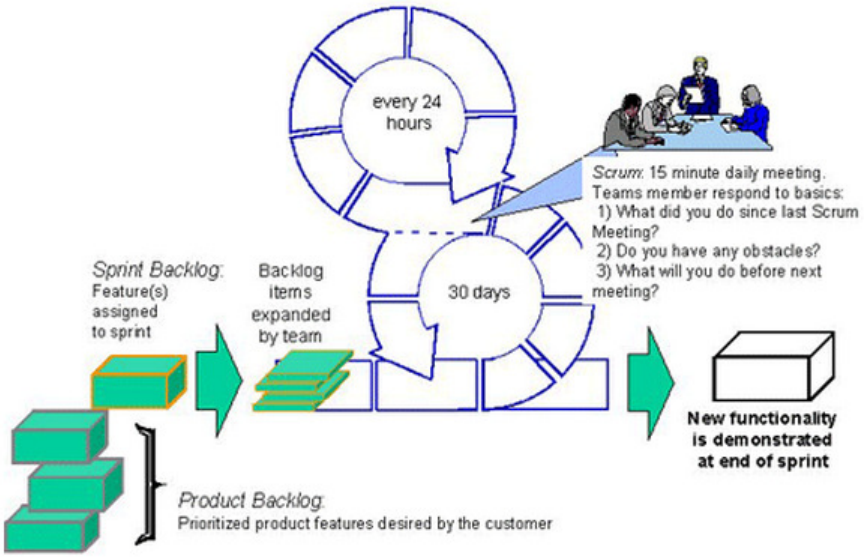


Figura 12 - Scrum

O Scrum tem no seu esqueleto iterações que podem ir de duas semanas a um mês e reuniões diárias [Schwaber2004]. Cada iteração pressupõe a entrega de funcionalidades que foram definidas a partir de um *backlog* de funcionalidades e alocadas a um determinado *sprint* (iteração).

Em cada iteração a equipa analisa os requisitos, considera a tecnologia disponível e avalia a sua capacidade e conhecimento para a implementação através do *sprint planning*. Colectivamente a equipa determina como vai construir a funcionalidade, modificando a abordagem diariamente sempre que encontra alguma dificuldade, complexidade ou surpresa. Em cada reunião diária (*daily scrum)*, que tem um intervalo de 15 minutos definido, cada elemento deve responder a três questões:

- Foi efectuado alguma coisa desde a última reunião?
- O que tem de ser feito entre esta e a próxima reunião?
- O que é que impede que o trabalho possa ser efectuada da forma mais eficaz possível?

Em cada iteração são realizadas duas reuniões, um *sprint review* para comunicar e analisar as funcionalidades desenvolvidas e um *sprint retrospective* para se analisar as lições aprendidas de melhoria para a próxima iteração.
Este processo criativo define o coração da filosofia Scrum.

Estão definidos três papéis principais: o dono do produto, a equipa e o *scrum master*. Todas as actividades de gestão estão divididas por estes três papéis. O dono do produto está focado nas funcionalidades do produto, os objectivos em termos de retorno de investimento e pela prioridade dos requisitos para a próxima iteração. A equipa é responsável pelo sucesso de cada iteração e o *scrum master* é responsável por garantir que as regras do Scrum estão a ser seguidas.

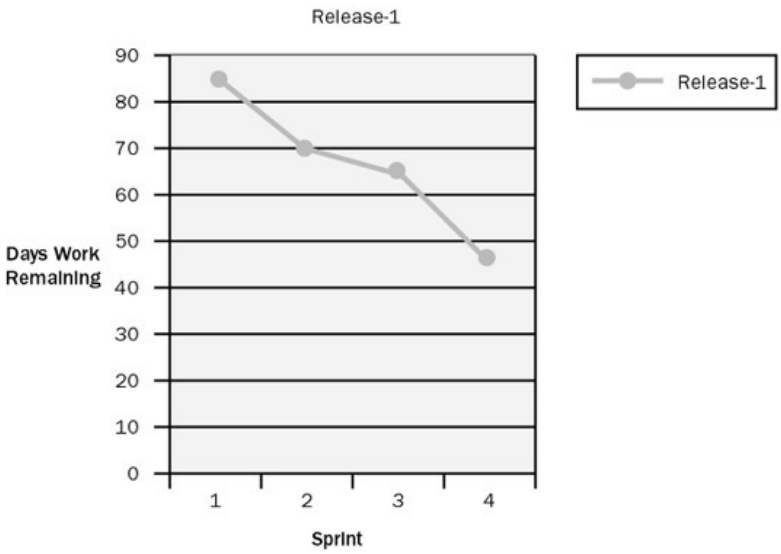


Figura 13 - Scrum Burndown Chart

A evolução do progresso de cada iteração é feita através de um *burndown* chart que mostra o número de horas executadas e previstas para terminar.

2.7 Extreme programming

O *Extreme Programming* (XP) nasce por volta do ano 2000, por Kent Beck e visa quebrar a premissa de que o custo de cada alteração num projecto aumenta exponencialmente à medida que o projecto avança [Beck2000]. Assim, através de uma série de práticas foi desenvolvido um modelo, que tenta tornar o custo das alterações do projecto igual em todas as fases do projecto e que segue os princípios dos métodos ágeis.

O XP está sustentado através de quatro valores [Stephens2003]:

- Comunicação: todas as equipas devem estar no mesmo sítio para evitar problemas de comunicação.
- Simplicidade: as tarefas devem ser feitas da forma mais simples possível – a ideia é que fazer as tarefas de uma forma complexa, demora mais tempo e o resultado obtido é o mesmo.
- Feedback: é importante fomentar a comunicação entre a equipa e com o cliente, de forma a obter a solução correcta o mais depressa possível. Seja nos requisitos, seja na arquitectura, o feedback deve ser procurado.
- Coragem: deve ser fomentada a coragem no projecto, nomeadamente para fazer alterações de desenho.

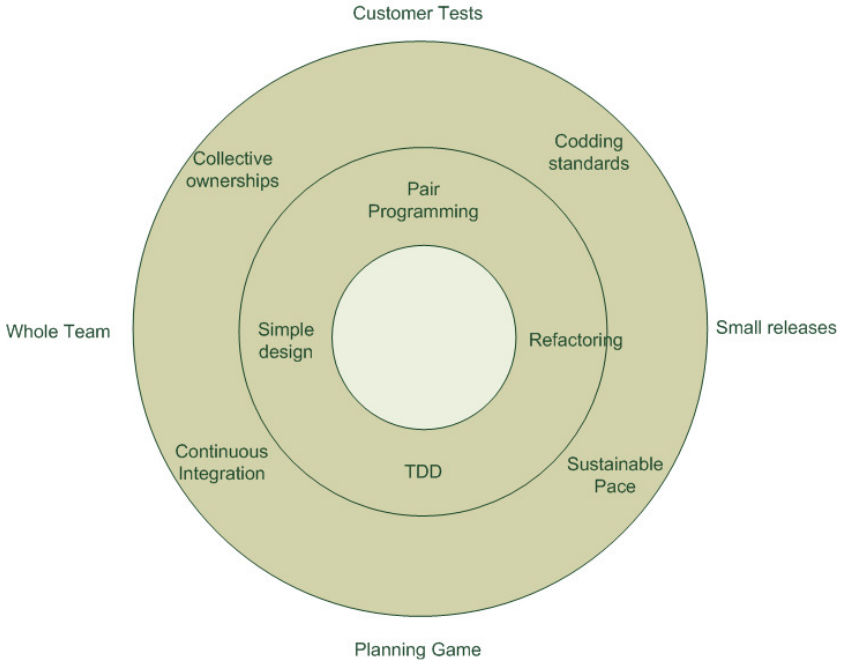


Figura 14 - Extreme Programming - Práticas

O XP é constituído por 12 práticas que têm de ser aplicadas em conjunto (Figura 14):

- *Test-driven development* – para cada classe desenvolvida de código, primeiro deve ser desenvolvido um teste que falhe, de seguida deve ser desenvolvido o código correspondente e de seguida o teste deve passar. Esta metodologia permite implementar um custo reduzido de mudança, pois qualquer alteração efectuada será imediatamente verificada pelos testes previamente produzidos, permitindo também incutir coragem para alterações de arquitectura.

- *The planning game* – Nesta prática são planeadas os requisitos, neste caso *user stories*, para que se decidam em que iteração devem ser desenvolvidos e entregues. As *user stories* devem ser estimáveis e testáveis. Cada *user story* (que fica num *story card*) é pendurada num quadro e deve ter uma estimativa de esforço que deve variar entre duas a quatro semanas. Para cada *story card*, devem ser criadas tarefas, de um a dois dias para facilitar a distribuição de trabalho. Para cada alteração de requisito ou pedido de documentação é criado um *story card* com um esforço associado. O esforço total dispendido por iteração é designado por *velocity* e deve ser mantido constante ao longo do projecto e como tal previsível.

- *Whole team (*anteriormente *on-site customer*) – O XP advoga que o cliente deve sentar-se com a equipa de desenvolvimento durante o projecto. Existiu uma evolução face ao publicado inicialmente onde agora o foco é ter um único representante do cliente – *proxy* (cliente esse que pode ser um conjunto de pessoas). Esta prática mais relaxada, parece vir responder à dificuldade de um princípio dos métodos ágeis, relativamente à necessidade da localização conjunta da equipa de desenvolvimento e do cliente.

- *Small releases* – Esta prática fundamental nos métodos ágeis é claro implementada pelo XP – o racional por detrás é o mesmo – incrementar o número de entregas o mais possível. Está também ligado ao princípio XP de obtenção de feedback o mais cedo possível. Recentemente o conceito de que uma *release* tem de ser algo completamente funcional num ambiente de produção foi relaxado, para algo que funcione apenas num ambiente de testes realizados por um cliente interno – algo aproximado a protótipos evoluídos.
- *Metaphor* – A arquitectura de sistema complementada com uma metáfora unificadora a algo existente, como por exemplo: "O sistema é como uma padaria. Tem o pão, o forno, os padeiros, a máquina registadora, etc, … ". Esta forma substitui a forma tradicional de documentação de arquitectura.
- *Simple design* – O princípio é de que desenho simples é mais simples de manter. Mais tarde foi introduzida a prática de *design* emergente, ou seja, um design que vai sendo construído à medida que o problema e a solução vão sendo melhor compreendidos.
- *Refactoring* – Embora seja recomendado o design simples, também é recomendado que se corrija constantemente o desenho que não esteja bem feito, tal como o código duplicado. Dado que o desenho da arquitectura não é feito previamente recomenda-se que o desenho vá emergindo e sendo corrigido ao longo do tempo.
- *Collective ownership* – Cada programador é responsável por todo o código, embora sejam responsáveis individualmente por cada tarefa. Por exemplo se um programador vê uma parte de código que ache que não esteja bem feito deve procurar corrigi-lo mesmo que não tenha sido ele a fazê-lo. É um conceito interessante que forma uma responsabilidade colectiva por aquilo que é produzido, mas que não deve servir de forma de desresponsabilização dos elementos na equipa.
- *Pair programming* – Esta é uma das práticas mais controversas e que diz respeito a que cada equipa, que tem de ser par, deve ser formada por conjuntos de duas pessoas, onde um codifica, outra verifica e ambos discutem os problemas e soluções. Dentro deste *pair programming* é aconselhado que exista uma troca de elementos regular entre os pares para fomentar a disseminação de conhecimento.
- *Continuous integration* – O código deve ser testado e integrado em poucas horas, no máximo um dia, e se no final do dia ainda existirem erros o código deve ser deitado fora. No dia seguinte o código deve estar isento de erros.
- *Sustainable pace* – O esforço da equipa deve ser sustentado, a semana não deve ter mais de quarenta horas, senão a qualidade diminui.
- *Programming standards* – recomendados em qualquer projecto de desenvolvimento de software, promovem a facilidade de compreensão e comunicação entre a equipa.

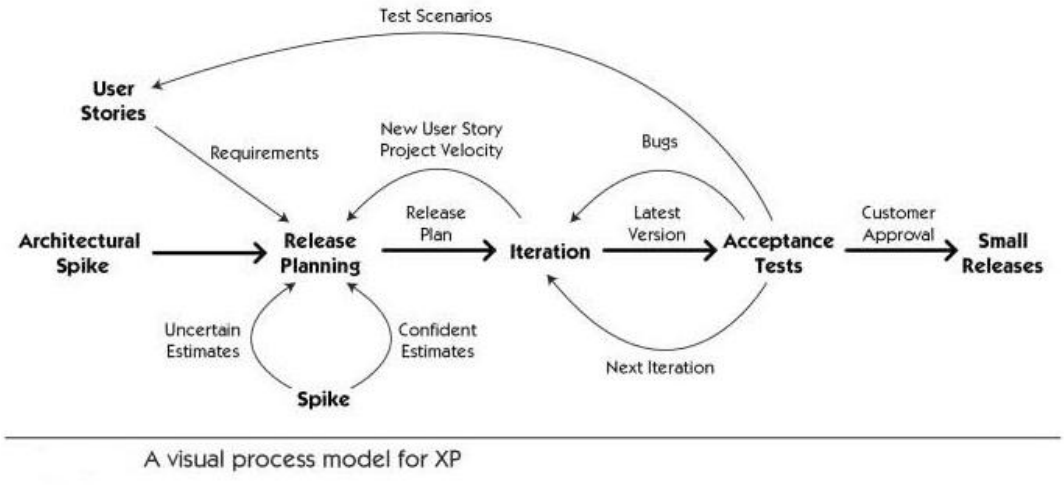


Figura 15 - Extreme Programming – Ciclo de vida

O XP descreve quatro actividades básicas no desenvolvimento de software: codificar, testar, desenhar e ouvir. Por outra ordem poderíamos mapear em requisitos (ouvir), desenhar (desenho), codificar e testar.

O ciclo de vida do XP define-se por iterações evolucionárias onde existem duas fases principais (Figura 15): o planeamento da *release* e a execução das iterações. Não confundir a evolução com pequenos incrementos, pois o processo de definição de requisitos, desenho, codificação e testes é feito quase em simultâneo [extremeorg2006].

Cada iteração está dividida em ciclos de duas semanas, onde uma versão é entregue ao cliente interno, onde posteriormente ele decide se pretende que a versão seja testada pelo utilizador final.

No início de uma iteração existe uma reunião com o cliente e com os programadores, onde são definidos as funcionalidades a implementar, através de *user stories*, os programadores estimam o tempo necessário para implementar cada *user storie*, e é decido em conjunto que funcionalidades ficam na próxima *release* e de fora. As *user stories* não devem ter muito detalhe, sendo que o detalhe é efectuado na prática quando o cliente escreve os testes de aceitação automatizados – de relembrar que aqui o cliente é um representante e pode ser alguém com background técnico. Diariamente existe uma reunião de pé (*stand up meeting*) onde são discutidos problemas, soluções e se promove o foco da equipa.

Estão definidos diversos papéis no XP entre os quais o programador, o testador, o *tracker*, responsável por seguir o tempo associado a cada tarefa e como tal a velocidade do projecto e o *coach* responsável por treinar os elementos nas práticas XP.

2.8 Combinando métodos ágeis e não ágeis

O ponto de partida de referência para a análise da combinação de modelos mais e menos ágeis foi efectuado sobre o trabalho de Barry Boehm e Richard Turner – Balancing agility and discipline [Boehm2004] onde é proposto um método assente numa análise de risco sobre os factores considerados mais relevantes para a aplicação ajustada.

Este método assenta sobre a análise de cinco factores principais (Figura 16):

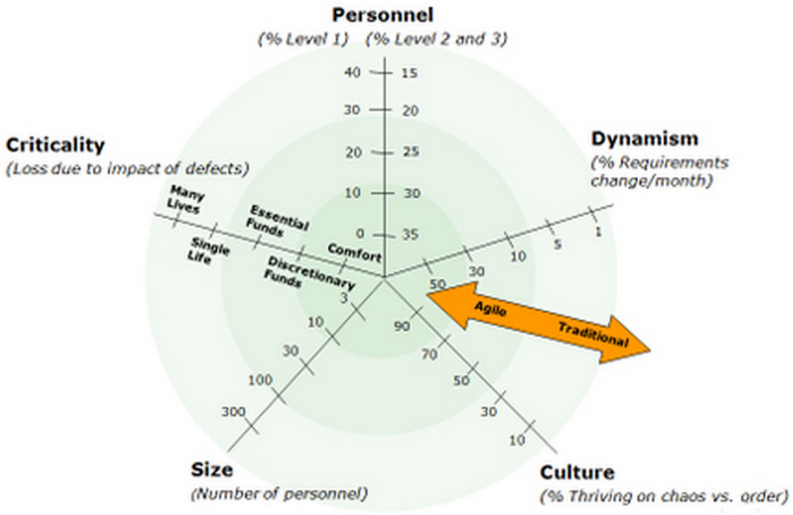


Figura 16 - Cinco factores para secção de métodos[8]

1. Tamanho: as metodologias ágeis estão mais adaptadas para equipas pequenas enquanto as metodologias menos ágeis são mais difíceis de adaptar para equipas pequenas.

2. Criticidade: as metodologias ágeis, até hoje foram pouco testadas em sistemas de grande criticidade, onde se adivinham dificuldades pela utilização de design simples e pouca documentação.

3. Dinamismo: o desenho simples de soluções e *refactoring* constante protege o custo de trabalho refeito em ambientes pouco estáveis em termos de alteração de requisitos, mas aumenta-o em ambientes estáveis.

4. Maturidade dos membros da equipa: os métodos ágeis funcionam melhor com membros de equipas com mais capacidades ao nível técnico e ao nível de entendimento processual - como existem menos processos formais as decisões ficam mais dependentes de cada indivíduo, e como tal as boas decisões dependem da sua capacidade.

5. Cultura: os métodos ágeis funcionam melhor em equipas onde os membros se sentem em ambientes com mais graus de liberdade de actuação (Figura 17).

Quanto mais a avaliação dos factores se aproxima do centro do gráfico mais ágil deve ser o método.

[8] Balancing Agility and Discipline, Barry Boehm, Richard Turner, Addison-Wesley, 2004

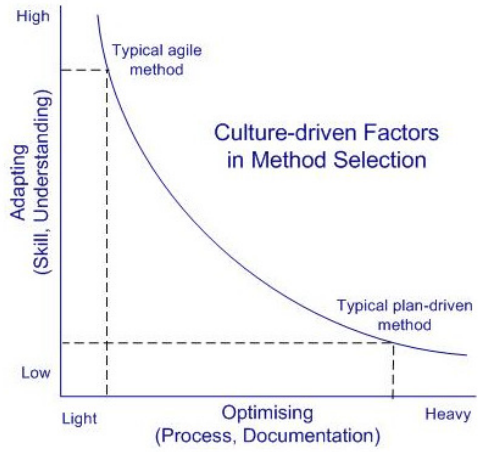


Figura 17 - Aspectos culturais na selecção do método[9]

Por outro lado considera também uma análise de risco associada ao tipo de abordagem, ponderando diversos factores, baseado num modelo de processo em espiral [Boehm2001] e [MBASE2003].

Considerando estes cinco factores é estabelecido um método que prescreve como devem ser utilizadas as abordagens ágeis e menos ágeis (Figura 18Figura 18 - Método Boehm de selecção de metodologia e Tabela 3).

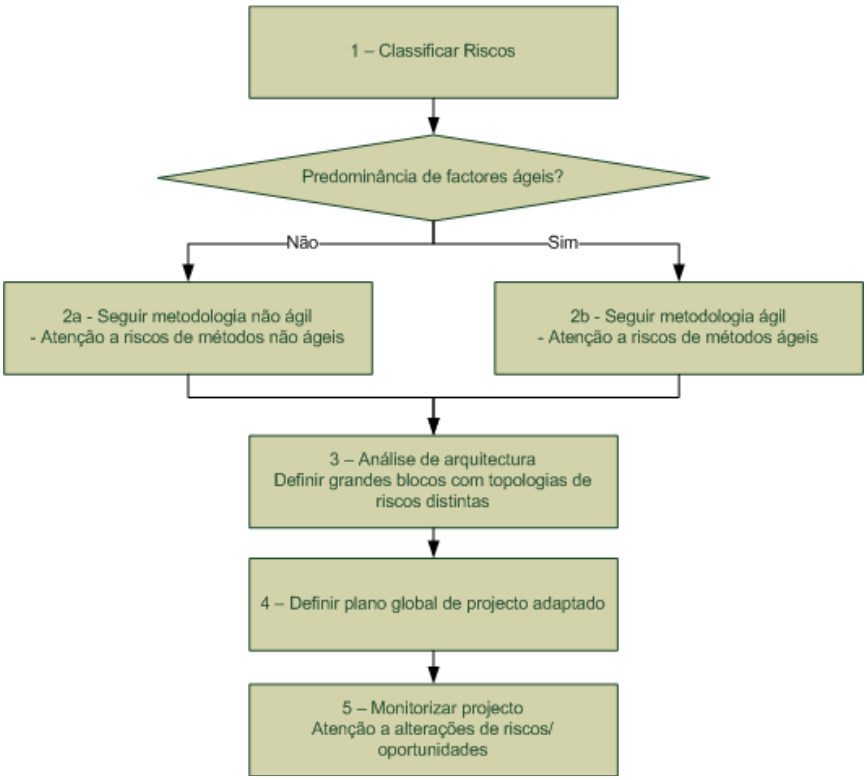


Figura 18 - Método Boehm de selecção de metodologia

[9] Balancing Agility and Discipline, Barry Boehm, Richard Turner, Addison-Wesley, 2004

Tabela 3 - Método de selecção de metodologia

Passo 1	Classificar os riscos associados de acordo com os riscos de ambiente, ágeis e não ágeis (Tabela 4)
Passo 2a	Se os riscos ágeis dominarem prosseguir com métodos não ágeis baseados numa gestão de riscos com foco nos riscos não ágeis
Passo 2b	Se os riscos não ágeis dominarem prosseguir com métodos ágeis baseados numa gestão de riscos com foco nos riscos ágeis
Passo 3	Subdividir a arquitectura e adoptar a abordagem apropriada caso parte das aplicações possam apresentar topologias distintas.
Passo 4	Estabelecer um plano global do projecto com os riscos associados
Passo 5	Monitorizar o progresso, os riscos e oportunidades, reajustando o balanceamento do processo conforme necessário.

Em [Edeen2007] os autores combinam esta metodologia com o conceito de organização ambidestra, onde se podem dividir os projectos ou componentes de desenvolvimento, dos projectos de software em duas subunidades da organização separadas, vocacionadas respectivamente para métodos mais e menos ágeis. Esta abordagem endereça de forma simples os riscos das necessidades de formação dos recursos nas respectivas abordagens metodológicas, A-Skill e P-Skill (Tabela 4), obrigando no entanto a uma dimensão mínima de organização e dupla cultura, o que implica um esforço extra de gestão.

Tabela 4 - Riscos metodológicos de Boehm

Risco de ambiente	
E-tech	Incerteza tecnológica
E-coord	Complexidade de stakeholders
E-cmplx	Complexidade de sistemas
Riscos ágeis	
A-Scale	Escalabilidade e criticidade
A-Yagni	Uso de desenho simples
A-Churn	Rotatividade de pessoal
A-Skill	Poucos recursos formados em métodos ágeis
Riscos métodos não ágeis	
P-Change	Mudança rápida de requisitos ou prioridades
P-Speed	Necessidade de resultados rápidos
P-Emerge	Requisitos facilmente emergentes
P-Skill	Poucos recursos formados em métodos não ágeis

Os autores de [Boehm2005] identificam os principais desafios em combinar metodologias ágeis em processos de desenvolvimento tradicionais, de acordo com três áreas principais: processo de desenvolvimento, processo de negócio e conflitos com pessoas.

Processo de desenvolvimento

- ➔ Variabilidade – a integração de equipas com ciclos de vida distintos pode levar ao desenvolvimento de produtos bastante distintos e difíceis de integrar.
- ➔ Ciclos de vida distintos – os métodos ágeis focam em entregas contínuas e antecipadas enquanto as não ágeis focam a sua entrega nas fases finais.
- ➔ Sistemas legados – os sistemas legados não são facilmente redesenhados (*refactoring)*, ou desacoplados para permitir entregas consecutivas e antecipadas.
- ➔ Requisitos – Os requisitos ágeis tendem a ser sobretudo funcionais e informais, dificultando processo de verificação e validação dos métodos não ágeis.

Processo de negócio

- ➔ Recursos humanos: as organizações devem aprender a acomodar assuntos relacionados com recursos humanos tais como registos de tempos, descrições de funções, prémios e gestão de competências. As pessoas por outro lado devem ser capacitadas para procurar abordagens não tradicionais.
- ➔ Gestão do progresso: os contractos tradicionais e as técnicas de recolha de informação podem ser consideradas desadequadas para suportar as práticas ágeis
- ➔ Avaliações de standards: a maior parte das práticas ágeis não estão preparadas para suportar as evidências necessárias, em termos de documentação, para suportar as avaliações relativamente a standards como o CMMI e a ISO 15504 (SPICE).

Conflitos com pessoas

- ➔ Atitudes de gestão: as organizações tradicionais tendem a definir papéis rígidos nas organizações, enquanto nas metodologias ágeis promove-se a flexibilidade de papéis.
- ➔ Assuntos logísticos: as equipas ágeis necessitam de estar colocadas de forma próxima com ferramentas específicas para *pair programming*. Algumas empresas não têm as condições necessárias em termos de espaço de trabalho.
- ➔ Gestão de pilotos: alguns gestores respondem aos sucessos dos pilotos ágeis com a separação das equipas, com vista a uma promoção ou disseminação de boas práticas. Este facto pode destruir as relações de bom trabalho que vão sendo criadas nas equipas.
- ➔ Gestão da mudança – lidar com recursos que se recusam a mudar de metodologia, pode ser crítico na implementação de métodos ágeis, dado que as suas práticas são baseadas em confiança e em muita comunicação directa.

Em [Edeen2007] os autores alegam que o conceito de agilidade – metodologias que são mais orientadas às pessoas do que aos processos - tem mais a ver com a área de sistemas de informação (com preocupações sociais e organizacionais) do que com o desenvolvimento de software.

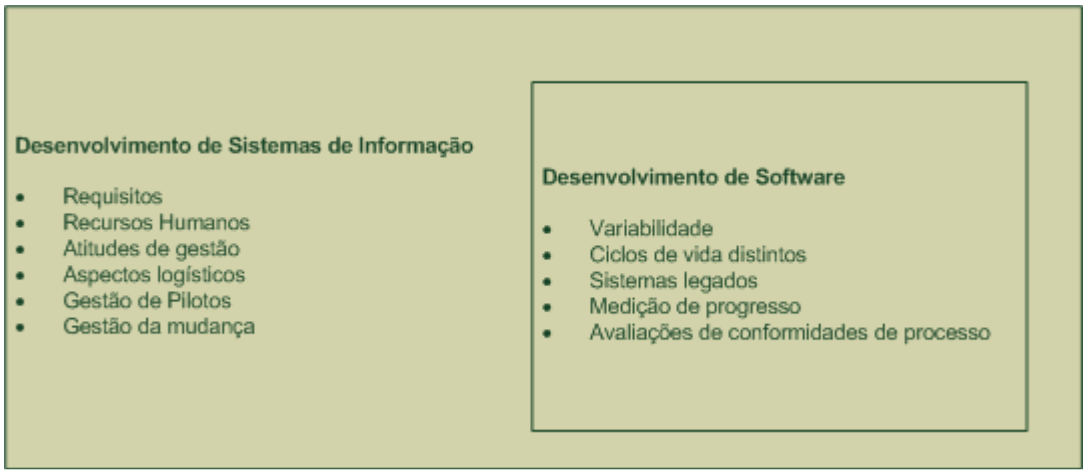


Figura 19 - Desenvolvimento de Sistemas de Informação e de Software

Em [Moe2008], os autores constataram que uma das maiores dificuldades na implementação das práticas ágeis tem a ver com o grau de especialização de alguns dos recursos e da dificuldade na criação de redundância nas equipas.

Em 2008, Kalpana e Jagadish da Wipro, partilharam as suas experiências na utilização de métodos ágeis, em ambientes de desenvolvimentos distribuídos, numa das maiores empresas de desenvolvimento de software offshore [Kalpana2008]. As suas principais conclusões passaram pela localização das equipas junto dos clientes em diversos momentos do projecto e pela utilização intensiva de ferramentas para a comunicação e pela utilização de documentação para permitir uma melhor colaboração entre equipas.

Os autores em [Paasivaara2008] também reforçam estes aspectos e falam da utilização de *proxys* de clientes que permitam uma comunicação directa entre pessoas. Em [Young2008] os autores exploram a utilização das novas tecnologias para lidar com os desafios da comunicação e da confiança.

3 Análise comparativa e critérios de selecção de modelos

Este capítulo começa apresentar uma série de conclusões sobre o estado da arte analisado no capítulo anterior, introduz o tema do custo da qualidade, e estabelece os princípios para um método global de planeamento de projectos de desenvolvimento de software, que integra os diversos modelos referidos.

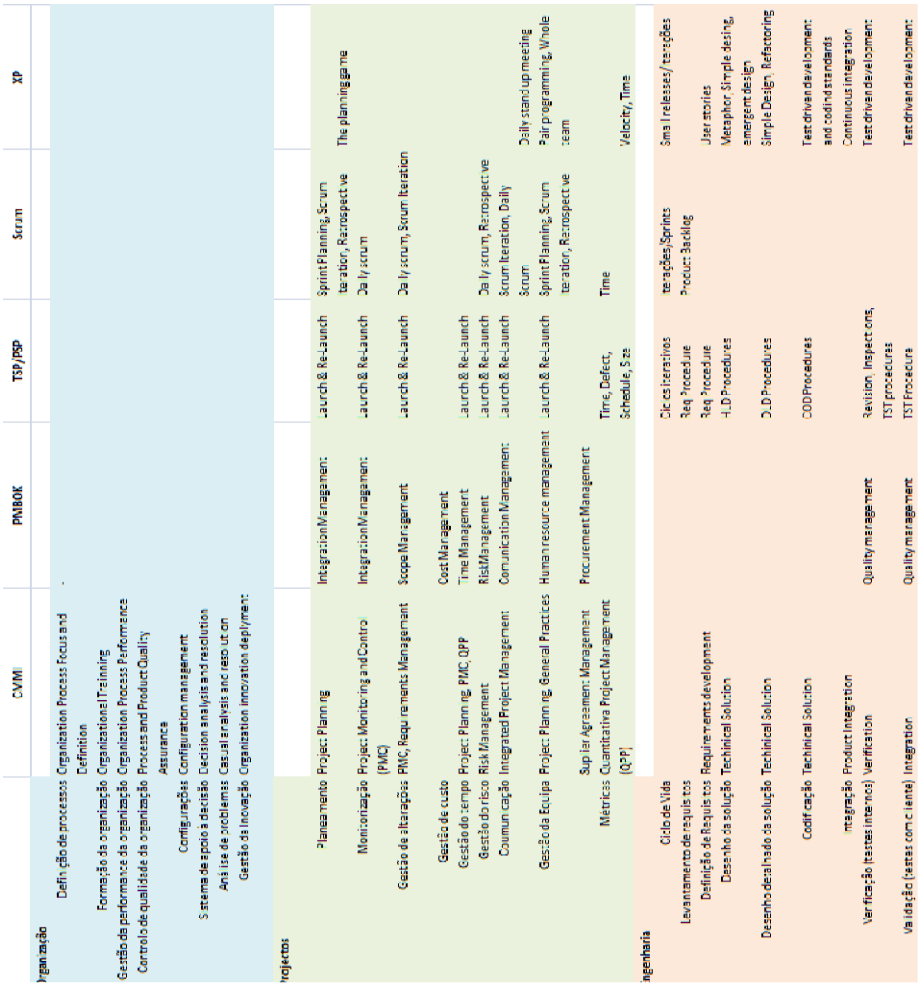
3.1 Análise comparativa dos modelos

A análise dos diferentes modelos, CMMI, PMBOK, TSP/PSP, Scrum, XP (Figura 20) e dos métodos de combinação permitem retirar as seguintes conclusões (Figura 21).

O CMMI é um modelo que está orientado para uma abrangência global do desenvolvimento de software, desde aos processos da organização, aos de gestão de projecto e aos de engenharia. É no entanto um modelo não prescritivo e que permite a adopção e combinação de outras metodologias ou parte delas. Em termos de certificação, de acordo com o modelo SCAMPI, obriga à apresentação das evidências de cada uma das áreas de processo, o que para uma metodologia baseada em pouca documentação pode ser desafiante. O seu factor distintivo está nos aspectos organizacionais: definição e controlo de processos, formação, inovação e análise e resolução de causas de problemas estruturais.

O PMBOK oferece a metodologia mais completa de gestão de projectos, contemplando aspectos como a gestão financeira, que não são abordados por nenhuma outra. Contempla também um pouco de organização mas ao nível de gestão global de projectos numa organização (*portfolio management).*

O TSP/PSP é uma metodologia iterativa de desenvolvimento de software, que abarca desde a gestão de projectos, gestão da equipa, qualidade de software, gestão quantitativa do processo, e prescreve uma sequência de actividades de especificação, elaboração, revisão e inspecção do trabalho produzido, sustentando por uma premissa de eficiência associada ao processo. Não apresenta indícios de problemas de escalabilidade ou de necessidade de localização.



		CMMI	PMBOK	TSP/PSP	Scrum	XP
Organização						
	Definição de processos	Organization Process Focus and Definition				
	Formação da organização	Organizational Trainings				
Gestão da performance da organização		Organization Process Performance				
Controlo de qualidade da organização		Process and Product Quality Assurance				
	Configurações	Configuration management				
	Sistema de apoio à decisão	Decision analysis and resolution				
	Análise de problemas	Causal analysis and resolution				
	Gestão da inovação	Organization innovation deployment				
Projectos						
	Planeamento	Project Planning	IntegrationManagement:	Launch & Re-Launch	Sprint Planning, Scrum Iteration, Retrospective	The planning game
	Monitorização	Project Monitoring and Control (PMC)	IntegrationManagement:	Launch & Re-Launch	Daily scrum	
	Gestão de alterações	PMC, Requirements Management	Scope Management	Launch & Re-Launch	Daily scrum, Scrum Iteration	
	Gestão de custo		Cost Management			
	Gestão do tempo	Project Planning, PMC, QPP	Time Management	Launch & Re-Launch	Daily scrum, Retrospective	
	Gestão do risco	Risk Management	RiskManagement	Launch & Re-Launch	Scrum Iteration, Daily Scrum	Daily standup meeting
	Comunicação	Integrated Project Management:	Comunication Management:	Launch & Re-Launch	Sprint Planning, Scrum Iteration, Retrospective	Pair programming, Whole team
	Gestão da Equipa	Project Planning, General Practices	Human resource management	Launch & Re-Launch		
	Supplier Agreement Management		Procurement Management			
	Metricas	Quantitative Project Management (QPP)		Time, Defect, Schedule, Size	Time	Velocity, Time
Engenharia						
	Ciclo de Vida			Ciclos iterativos	Iterações/Sprints	Small releases/ iterações
	Levantamento de requisitos			Req Procedure	Product Backlog	
	Definição de Requisitos	Requirements development		Req Procedure		User stories
	Desenho da solução	Technical Solution		HLD Procedures		Metaphor Simple desing, emergent design
	Desenho detalhado da solução	Technical Solution		DLD Procedures		Simple Design, Refactoring
	Codificação	Technical Solution		COD Procedures		Test driven development: and code standards
	Integração	Product Integration				Continuous integration
	Verificação (testes internos)	Verification	Quality management	Revision, Inspections, TST procedures		Test driven development:
	Validação (testes com cliente)	Integration	Quality management	TST Procedure		Test driven development:

Figura 20 - Comparando os métodos

O Scrum é uma metodologia ágil iterativa, orientada para a gestão de projectos, que promove a comunicação entre os elementos das equipas, desenho simples e pouco pensado a nível estratégico e pouca documentação. Está orientado para permitir mudanças de âmbito ao longo do projecto, promove entregas constantes e não é muito prescritivo nas práticas de engenharia.

O XP é uma metodologia ágil iterativa que foca em aspectos de engenharia, assenta sob actividades que devem ser realizadas em conjunto, tais como o *pair programming*, *refactoring*, *simple desing* e *test driven development*. É uma metodologia prescritiva em termos de processos de engenharia, que evita os requisitos detalhados e documentados, evita o desenho detalhado e a documentação.

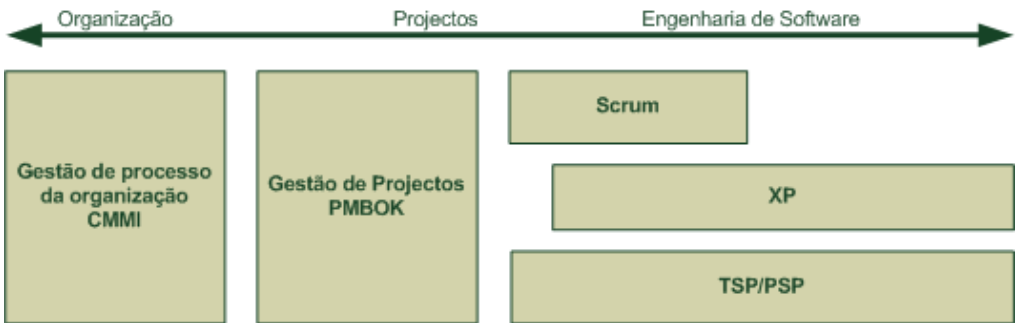


Figura 21 - Posicionamento das metodologias

Tanto o TSP/PSP, o Scrum como o XP assentam sobre princípios de iteração, e reconhecimento da importância e necessidade de *empowerment* das pessoas e equipas no processo de desenvolvimento.

A necessidade de localização do cliente junto com a equipa de desenvolvimento não parece ser uma necessidade efectiva de nenhuma das metodologias, embora os princípios ágeis o possam fazer transparecer.

As grandes diferenças entre o TSP/PSP e os métodos ágeis Scrum e XP assentam sobretudo:

- na perspectiva detalhada e documental das práticas de engenharia e gestão do projecto
- na abordagem quantitativa da gestão dos processos de gestão e engenharia
- no controlo de qualidade rigoroso associado

O TSP/PSP fá-lo e o Scrum e XP não. O TSP/PSP é portanto mais previsível e como tal escalável para grandes projectos em termos de dimensão, localização ou risco.

Como consequência o TSP/PSP apresenta maior grau de previsibilidade e o Scrum e XP maior grau de agilidade.

Num exercício simplista de relacionamento do nível de maturidade potencial de CMMI com as metodologias TSP/PSP, XP e Scrum, o TSP/PSP estaria algures no nível 4 [Wall2007], devido à sua gestão quantitativa e completude de definição de processos, o XP estaria a apontar para o nível três, devido às práticas de engenharia já definidas, embora sem as componentes de processo, e o Scrum no nível dois devido ao seu foco na gestão de projectos.

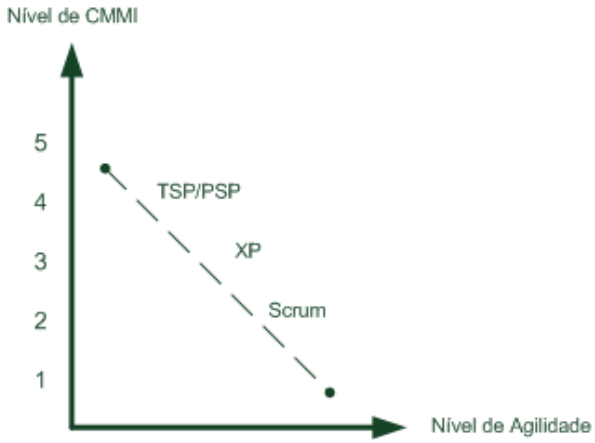


Figura 22 - CMMI, Scrum e TSP/PSP

Foi abordado que consoante a organização, o cliente, o projecto deve ser implementada uma postura mais ou menos ágil (Figura 23).

Figura 23 - Centro de gravidade de um processo

O modelo de Barry Boehm e Richard Turner [Boehm2004] contempla um método global de decisão sobre a aplicação de métodos ágeis e não ágeis que pressupõe a análise do contexto do projecto (mais ou menos ágil) consoante cinco factores: maturidade da equipa, dinamismo, criticidade, tamanho e cultura, e depois a subdivisão do projecto em eventuais perfis distintos de agilidade e posterior gestão de risco associado a cada perfil (ver capítulo Combinando métodos ágeis e não ágeis).

3.2 O custo da qualidade

O custo de um projecto de software pode ser analisado de acordo com o modelo apresentado pela Ratheon em 1995 (Figura 24), que faz uma decomposição em duas vertentes principais: o custo de performance e o custo da qualidade.

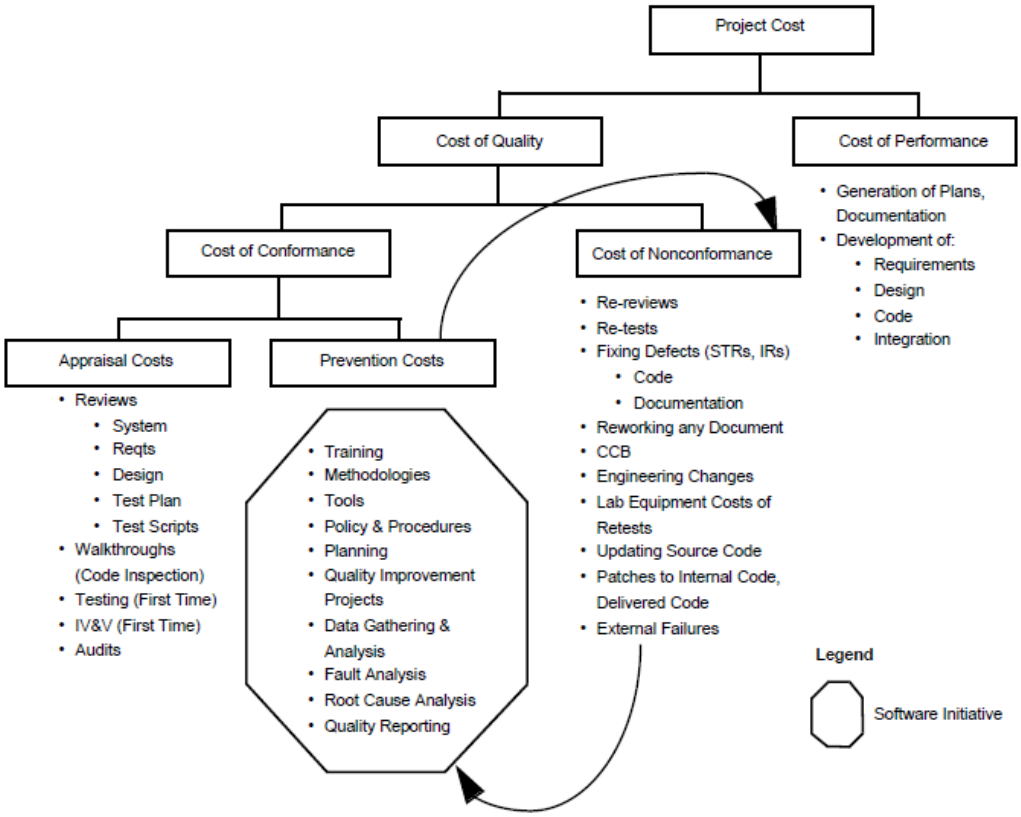


Figura 24 - Custo de performance e da qualidade[10]

O custo da performance tem a ver com todas as actividades que visam a produção de algo que necessariamente tem que acontecer no projecto e o custo da qualidade com o custo de actividades de controlo de conformidade (testes ou actividades de prevenção) e com o custo de actividades de correcção das não conformidades (trabalho repetido ou correcção de erros).

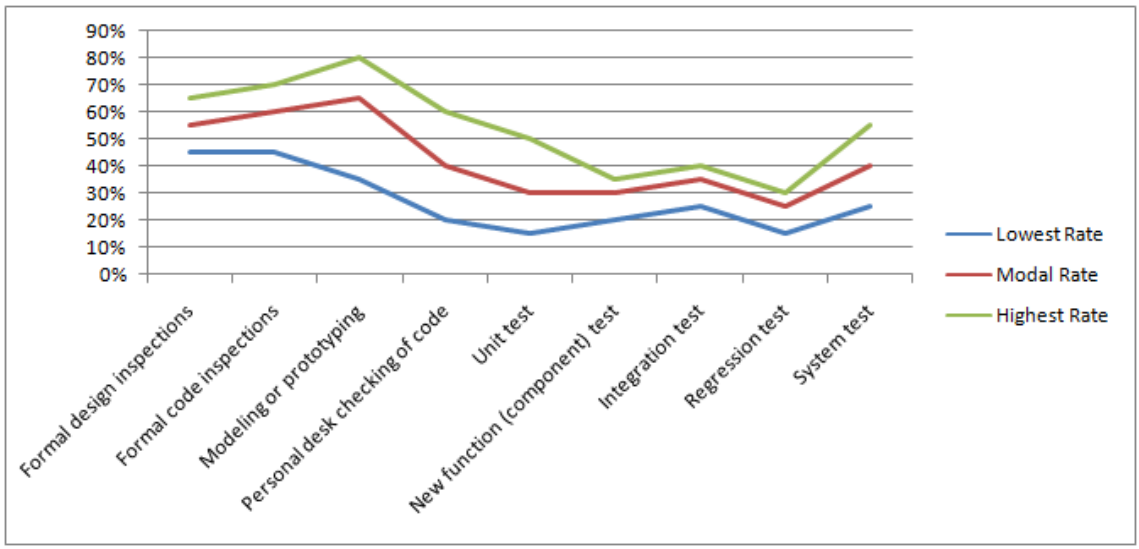
Em termos de performance as empresas podem utilizar várias técnicas, tais como a reutilização, através de módulos reutilizáveis, *frameworks*, *design patterns*, ou automatização, através da geração automática de código. Não podemos esquecer no entanto que as diversas actividades relacionadas com este modelo estão dependentes e relacionadas. Se eventualmente não fizermos qualquer tipo de investimento em análise de requisitos, muito provavelmente os erros encontrados vão aumentar e o trabalho repetido também. Pelo que importa garantir que as actividades relacionadas com a performance devem ser executadas de acordo com as melhores práticas.

Para evitar as actividades de não conformidade, devemos portanto investir na excelência de execução de actividades de performance e devemos atingir um equilíbrio com as actividades de controlo de conformidade para uma remoção eficiente e eficaz de defeitos encontrados. As actividades relacionadas com a prevenção de defeitos estão relacionadas sobretudo com a organização e respectiva gestão de processos, normas, formação, etc.

[10] Raytheon Electronic Systems Experience in Software Process Improvement, CMU/SEI-95-TR-017, 1995

Relativamente às actividades de eliminação de controlo reactivo de conformidade, como revisões, inspecções, testes, etc, em 1996 Jones [Jones1996] publicou um estudo que analisava a eficiência associada aos métodos de remoção de defeitos (Tabela 5).

Tabela 5 - Eficiência de remoção de defeitos



A análise efectuada à eficiência de remoção de defeitos de diversos métodos, que tem sido confirmada através de estudos posteriores [Nichols2009], vem demonstrar que os métodos de remoção de defeitos em fases anteriores do desenvolvimento, nomeadamente com revisões e inspecções formais, são mais eficientes do que testes unitários, testes integrados e de sistema.

3.3 Critérios na selecção de modelos de desenvolvimento

Dentro de uma organização um projecto avança: ou porque é obrigatório, por advir de uma necessidade de regulamentação ou de mercado; ou porque representa um valor acrescentado à organização. Tanto num caso como noutro existe a necessidade de garantir por todos os meios o retorno de investimento previsto ou a maior contenção possível de custos.

Muitas vezes alguns projectos não avançam porque não poderão estar realizados em determinado prazo, ou porque representam um custo muito elevado (ou sem retorno previsto) para a organização. Se um projecto está assente em determinados pressupostos de custos, prazos, risco, âmbito e qualidade, os processos pelos quais o mesmo é implementado devem garantir o máximo controlo destas variáveis. Este controlo está associado à previsibilidade.

Alguns valores são ingredientes chave para o sucesso dos projectos, e estes estão espelhados em todos os métodos apresentados: desenvolvimento em iterações, boa comunicação com o cliente e entre a equipa, equipa motivadas, capacitadas e comprometidas e boas práticas de engenharia.

Uma organização normalmente não executa apenas um projecto, pelo que devem ser definidas políticas organizacionais para a gestão de projectos, sendo que estas políticas devem ser suficientemente flexíveis para se poderem adaptar a contextos diferentes. A organização

deve procurar de forma uniforme, inovar, aprender com lições aprendidas, suportar a formação de todos os colaboradores, ter processos definidos que evoluem e se adaptam conforme as necessidade e devem ser geridas com métricas.

Os processos de engenharia de software devem procurar de forma eficiente a produção de produtos ou serviços de alta qualidade, pelo qual os mecanismos implementados devem ter em conta essa eficiência.

Talvez faltem estudos mais rigorosos que comparem os modelos económicos associados ao Scrum e XP quando comparados com o TSP/PSP.

Pela informação disponível o TSP/PSP parece ser de uma perspectiva racional mais indicado em termos de adopção como metodologia de desenvolvimento. Tem no entanto um custo de *setup* inicial elevado, que deve ser encarado como um investimento.

Para projectos de pequena dimensão ou criticidade, a utilização de métodos mais ágeis vai responder de forma mais rápida e por vezes mais ajustada, mas sem documentação sustentada – algo que por vezes o cliente pede.

Em relação ao método proposto por Boehm (ver capítulo Combinando), será feita uma proposta de modelo que abranja de uma forma mais global, a utilização dos modelos mencionados para aspectos de organização e gestão de projectos.

Será também complementado com alguns critérios de selecção na componente de relação com o cliente, na forma de detectar partes ágeis dos projectos e no facto da equipa estar ou não localizada em locais distintos.

O modelo referido pressupõe a análise dos componentes para se eventualmente detectar padrões ágeis de abordagem, mas não refere, que para além dos componentes, as fases dos projectos podem ser interessantes para aplicar este tipo de metodologias. Por exemplo: Pode fazer sentido utilizar uma abordagem ágil no início de um projecto, para a realização de uma prova de conceito [Edeen2007], e depois caso o projecto avance se proceda com uma metodologia menos ágil.

4 Método de selecção de modelos de desenvolvimento de software

Neste capítulo será apresentado um método para o planeamento de um projecto de desenvolvimento de software, baseado no modelo de Boehm, de acordo com os princípios estabelecidos no capítulo anterior. Serão dados alguns exemplos de aplicação do método em diferentes tipologias de projectos.

4.1 Método de planeamento

O método de planeamento está dividido 4 fases, de A a D, com alguns componentes transversais que devem ser implementados em qualquer situação e com a primeira fase opcional caso apenas se esteja a prever a realização de apenas um projecto na organização. Como resultado possível pode ser recomendada:

- A aplicação de metodologias ágeis como Scrum e XP
- Métodos ágeis (Scrum + XP) mas com geração de documentação
- TSP/PSP

O método inicia-se com a recomendação clara da necessidade de implementação de metodologias que enderecem as necessidades da gestão de uma organização, ao nível de definição e controlo de processos, formação, inovação e análise dos problemas ou lições aprendidas. Esta necessidade pode ser colmatada com os modelos CMMI ou SPICE, que não são prescritivos em relação às práticas de gestão de projectos ou de engenharia. O CMMI é especialmente recomendado dado o seu grau de implementação mundial, a comunidade envolvente que o utiliza, e o facto de fornecer uma perspectiva integrada de desenvolvimento, serviços e aquisições de software.

O segundo passo passa pela implementação de uma norma de gestão de projectos completa, que assegure todas as actividades necessárias à sua gestão. Neste caso as normas PMBOK ou Prince são aconselhadas dada a sua implantação e grau de completude.

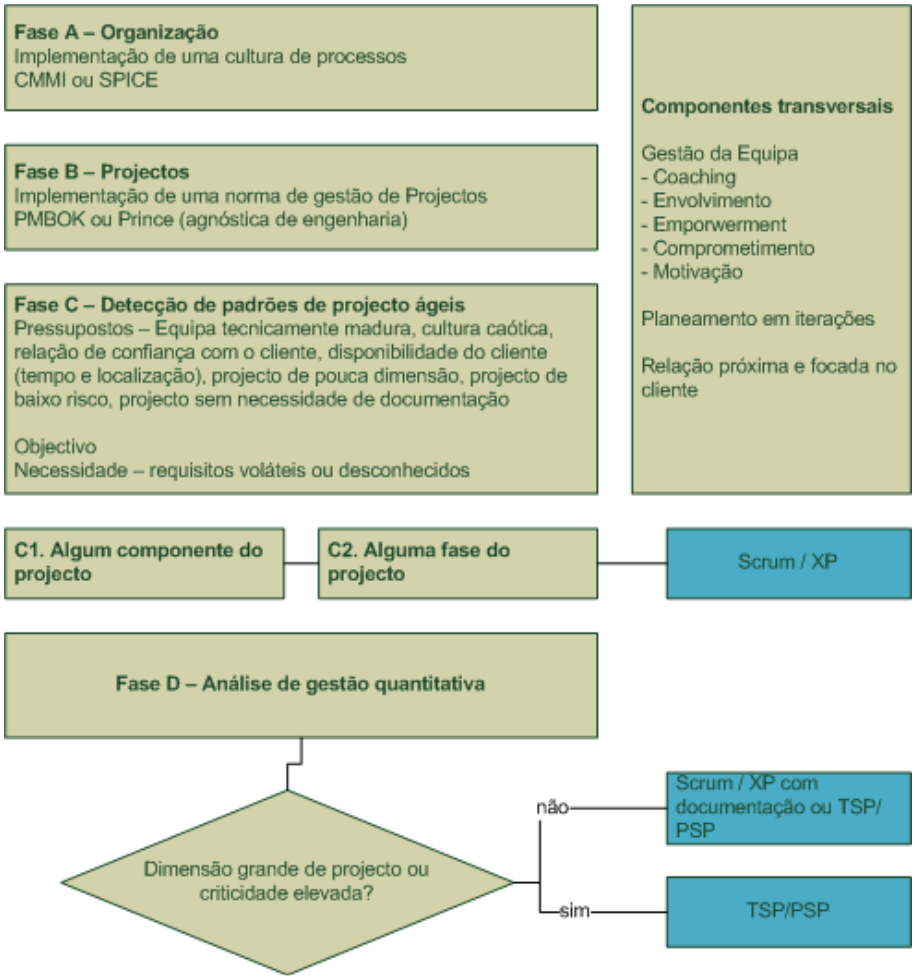


Figura 25 - Método para planear a qualidade

O terceiro passo tem a ver com a detecção de padrões de implementação de métodos ágeis nos projectos. A detecção destes padrões passam pelo objectivo inicial de que existe alguma indefinição ou volatilidade potencial ao nível de requisitos, para se poder explorar o conceito de agilidade, mas deve garantir que todos os pressupostos seguintes estão assegurados:

- A equipa tem alguma maturidade técnica
- A equipa está preparada para uma cultura não processual
- O projecto ou fase não tem uma grande dimensão
- O projecto ou fase não tem uma grande criticidade
- Existe uma relação de confiança com o cliente
- O cliente, ou representante, tem disponibilidade física e temporal
- A equipa de projecto não está localizada em locais distintos
- O cliente não solicita documentação exaustiva

A detecção destes padrões pode ser efectuada ao nível de um componente do projecto ou de uma fase do projecto. Caso seja encontrado este padrão, uma combinação de Scrum e XP parece aconselhada, dado que são as metodologias ágeis mais utilizadas a nível mundial.

Por fim, para as fases ou módulos restantes do projecto, deve ser adoptada uma filosofia não ágil, que pode ser baseada numa filosofia ágil mas que deve ser complementada com

princípios mais documentados, ou no TSP/PSP. Aqui recomenda-se a aplicação da prática a que a equipa está mais familiarizada ou em que seja oportuno investir.

Para projectos de grande dimensão ou criticidade recomenda-se a metodologia do tipo TSP/PSP.

Independentemente da metodologia a implementação de componentes transversais deve estar sempre presente:

- Gestão da equipa: deve ser promovida uma gestão de equipa que garantam a sua motivação e comprometimento. Isto é conseguido através do seu correcto envolvimento em todas as actividades que lhes afectam directamente, através de uma garantia de autonomia de actuação e com uma componente de *coaching*, que promova a excelência de práticas através da uma liderança baseada no exemplo.

- Desenvolvimento em iterações: todas as metodologias modernas promovem hoje a utilização de iterações nos projectos. Os seus benefícios são claros, pois promovem uma redução do risco do projecto, através de momentos antecipados de aceitação do cliente, realizações concretas e não teóricas dos projectos, emergência de requisitos e desenhos.

- Relação próxima com o cliente: o cliente não precisa de estar presente a todo a hora junto da equipa de desenvolvimento, mas pelo menos em pensamento sim. Deve existir um constante foco naquilo que são as necessidades do cliente, que se traduzem não só pelo garantir do âmbito correcto mas também do prazo e custo. Deve ser promovida uma relação de proximidade com o cliente, que se consegue através de um contacto frequente e de confiança. Essa confiança constrói-se com *feedback*, com cumprimento dos pressupostos estabelecidos.

4.2 Epílogo

O tigre representa a combinação de métodos ágeis e não ágeis (Figura 26). Tem o grau de robustez necessário para grandes projectos e desafios e por outro lado é capaz de reagir rapidamente a situações imprevistas.

Figura 26 - Combinando práticas ágeis e não ágeis

5 Conclusões e trabalho futuro

Neste capítulo são apresentadas as principais conclusões do trabalho, face aos objectivos propostos e são delineadas algumas linhas potenciais de trabalho futuro.

5.1 Satisfação dos objectivos

Esta tese propôs-se a responder a três questões principais:

- Quais as diferenças principais entre os principais modelos mais e menos ágeis?
- Que tipo de modelo de desenvolvimento de projectos de software deve uma organização adoptar?
- Para que tipo de projectos se aplicam cada tipo de modelo?

Pretendia-se com isto perceber de forma mais clara quais as diferenças, semelhanças e graus de aplicabilidade de diferentes metodologias de desenvolvimento de software, para assim conseguir uma gestão de projectos com maior sucesso. Este sucesso é medido através da satisfação do cliente e das suas expectativas indissociáveis: âmbito, custo, prazo e qualidade.

A resposta a estas questões foi efectuada através da análise dos modelos de desenvolvimento de software mais utilizados tanto nas áreas ágeis como nas áreas menos ágeis (TSP/PSP, Scrum e XP), conjugada com modelos complementares mais orientados à gestão da organização (CMMI) ou mais à gestão de projectos (PMBOK). Esta análise revelou que mais do que as diferenças que os distinguem, existem objectivos e contextos de aplicabilidade distintos.

Do trabalho já realizado pela comunidade científica e empresarial, foi escolhido um método base para uma evolução, que fizesse emergir um novo método que contempla critérios e pressupostos de decisão consoante o tipo de projecto, organização e cliente em causa.

Este método utiliza os modelos aqui referidos para adquirir uma abrangência que trata a área de gestão dos processos da organização, a gestão dos projectos e as práticas de engenharia. Este método salientou também as características que são transversais a qualquer modelo seleccionado e que passam por uma evolução iterativa dos projectos, uma gestão cuidada das equipas, através do envolvimento, comprometimento, autonomia, motivação e *coaching*, e por um foco no cliente, uma relação próxima, alinhada e de confiança.

A conjugação do movimento gerado pelos métodos ágeis e pelos métodos menos ágeis pode trazer uma força virtuosa à evolução engenharia de software – uma constante preocupação pela excelência dos produtos entregues, através de processos de desenvolvimento controlados e sustentados aliados a uma capacidade de adaptação e inovação, e a um foco cuidado nas pessoas.

5.2 Trabalho futuro

Existem vários caminhos que podem ser escolhidos como linhas futuras de avanço deste trabalho.

Por uma lado a experimentação do método em organizações que desenvolvem software, percebendo se de facto a sua aplicação propicia os resultados pretendidos. Associada a esta experimentação devem estar previstos ajustes nos requisitos e pressupostos estabelecidos, nomeadamente para a componente de detecção de padrões ágeis e na detecção de padrões transversais de aplicação – aqueles que não podem falhar seja qual for a metodologia.

Tantos os métodos ágeis como os métodos não ágeis têm sofrido evoluções ao longo dos últimos anos, bem como têm surgido novos modelos (por exemplo o método ágil *scrum-ban*). Estes novos métodos devem ser analisados de forma a perceber que contribuições podem trazer para o método definido.

Por outro lado será interessante observar a abrangência das metodologias não só na componente de desenvolvimento mas também nas componentes de gestão de serviço de software, aquisições e segurança. Por exemplo é interessante verificar a evolução do CMMI para as componentes de serviços e aquisições, a evolução de TSP/PSP para aspectos de manutenção de aplicações.

Deve existir uma constante atenção à comunidade científica e à indústria que continuam a produzir contribuições sobre o tema. Não é fácil encontrar experiências sustentadas por dados quantitativos, especialmente nos métodos ágeis, dados que eles assentam precisamente em processos leves que não promovem a recolha de exaustiva de métricas.

Referências

[AgileMan2001] Manifesto for Agile Software Development - http://agilemanifesto.org

[Alegria2006] Implementing CMMI using a combination of agile methods, Joe Alegria, Maria Bastarrica, CLEI Electronics Journal, 2006

[Aoyama1998] Agile Software Process and Its Experience, Mikio Aoyama, IEEE, 1998

[Beck2000] Kent Beck, Extreme Programming Explained: Embrace Change,Addison-Wesley, 2000

[Boehm2001] Balancing Discipline and Flexibility With the Spiral Model and MBASE, Barry Boehm, Daniel Port, Crosstalk 2001

[Boehm2004] Balancing Agility and Discipline, Barry Boehm, Richard Turner, Addison-Wesley, 2004

[Boehm2005] Management challenges to implementing agile processes in traditional development organizations, IEEE Software, Boehm, Turner, 2005

[Cohen2003] Agile Software Development - A DACS State-of-the-Art Report, David Cohen, Mikael Lindvall and Patricia Costa, 2003

[ControlChaos2009] http://www.controlchaos.com, 2009

[Edeen2007] Agility versus Discipline: Is Reconciliation Possible? G. Galal-Edeen, A. Riad, M. Seyan, ICCES Proceedings, 2007

[extremeorg2006] http://www.extremeprogramming.org/, 2006

[Glazer2008] CMMI or Agile - Why not embrace both! Hilel Glazer, Jeff Dalton, David Anderson, Mike Konrada, Sandy Shrum, CMU/SEI-2008-TN-003, SEI, 2008

[Humphrey2000] Introduction to Team Software Process, Watts S. Humphrey, 2000, Addison-Wesley

[Humphrey2005] TSP - Leading a deveopment team, Watts S. Humphrey, 2005, Addison-Wesley

[Jones1996] Software defect-removal efficiency, Jones, 1996

[Kalpana2008] Adopting agile in Distributed development, KalpanaSureshchandra, Jagadish

Shrinivasavadhani, 2008

[Konrad2007] CMMI, 2nd Edition – Guidelines for process integration and product improvement, Mary Beth Chrissis, Mike Konrad and Sandy Shrum, Addison Wesley, 2007

[Livermore2007] Factors that Impact Implementing an Agile Software Development Methodology, Jeffrey A. Livermore, Walsh College, IEEE, 2007

[MBASE2003] Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE), Center for Software Engineering, University of South California, 2003

[MCHale2005] Technical Report Mapping TSP to CMMI – SEI - James McHale, Daniel S. Wall, April 2005

[Mikio1998] Agile Software Process and Its Experience, 1988 IEEE

[Moe2008] Understanding Self-organizing teams in Agile Software Development, Nils Moe, Torgeir Dingsoyr, Tore Dyba19th Australian Conference, 2008

[Nichols2009] Deploying TSP on a National Scale: Na expirience report from Pilot Project in Mexico, Carneggie Mellon 2009

[Paasivaara2008] Distributed Agile Development: Using scrum in a Large project

[PMBOK2008] PMBOK Guide 4th Edition , The Project Management Institute, 2008

[Schwaber2002] Schwaber, K. and Beedle, M., Agile Software Development with SCRUM , Prentice-Hall, 2002

[Schwaber2004] Agile Project Management with Scrum, by Ken Schwaber, Microsoft Press, 2004

[Standish2009] The software development project success -rates The Standish Group – 2009

[Stephens2003] Extreme Programming Refactored: The Case Against XP, Matt Stephens and Doug Rosenberg, Apress, 2003

[Wall2007] Case Study: Accelerating Process Improvement by integrating TSP and CMMI, Daniel Wall, James McHale, Marsha Pomeriy-Huff, 2007, Technical report SEI

[Watts2001] Watts Humphrey, “Comments on eXtreme Programming”, eXtreme Programming Pros and Cons: What Questions Remain?, IEEE Computer Society Dynabook,2001

[Watts2002] Winning with Software – An executive summary, Watts S. Humphrey, Addison-Wesley, 2002

[Young2008] How we did adapt agile processes tou our distributed development?, Cynick Young, Hiroki tereshima, Agile 2008 Conference

6 Mapa conceptual