

Fabio Braga de Oliveira

Relações entre o PMBOK e metodologias ágeis de desenvolvimento de software

Campinas – SP

Dezembro / 2008

Fabio Braga de Oliveira

Relações entre o PMBOK e metodologias ágeis de desenvolvimento de software

Trabalho apresentado ao curso MBA em Gerenciamento de Projetos, Pós-Graduação *lato sensu*, da Fundação Getúlio Vargas como requisito parcial para a obtenção do Grau de Especialista em Gerenciamento de Projetos.

Orientador:
Prof. André Valle

FUNDAÇÃO GETÚLIO VARGAS

Campinas – SP

Dezembro / 2008

Fundação Getúlio Vargas

Programa FGV Management

MBA em gerenciamento de projetos

O trabalho de conclusão de curso “*Relações entre o PMBOK e metodologias ágeis de desenvolvimento de software*”, elaborado por Fabio Braga de Oliveira e aprovado pela Coordenação Acadêmica do curso MBA em Gerenciamento de Projetos em 1 de dezembro de 2008, foi aceito como requisito parcial para a obtenção do certificado do curso de pós-graduação, nível de especialização do Programa FGV Management.

André Bittencourt do Valle
Coordenador Acadêmico Executivo

Termo de compromisso

O aluno Fabio Braga de Oliveira, abaixo assinado, do curso MBA em gerenciamento de projetos, turma PROJ15 do programa FGV Management, realizado nas dependências da BI Campinas, no período de 01/01/2007 a 01/07/2008, declara que o conteúdo do Trabalho de Conclusão de Curso intitulado “Relações entre o PMBOK e metodologias ágeis de desenvolvimento de software” é autêntico, original e de sua autoria exclusiva.

Campinas, 01 de dezembro de 2008.

Fabio Braga de Oliveira

Dedicatória

Dedico este trabalho a minha esposa,

Que com carinho e paciência tem me ensinado os verdadeiros valores da vida.

Agradecimentos

Dedico meus sinceros agradecimentos para:

- o professor André Bittencourt do Valle, pela orientação e incentivo;
- o amigo Guilherme O'Connor, por sua revisão, influência, sugestões e amizade;
- ao projeto abnTex (ABNTEX, 2008) e sua comunidade, praticamente responsável pela bela formatação gráfica deste trabalho;
- a todos os colegas da turma PROJ15 do curso MBA em gerenciamento de projetos da Fundação Getúlio Vargas.

“The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds castles in the air, from air, creating by exertion of the imagination. [...] Yet the program construct, unlike the poet’s words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. [...] The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard , and a display screen comes to life, showing things that never were nor could be.”

Brooks

Resumo

Neste trabalho foi realizada uma pesquisa bibliográfica buscando as relações entre metodologias modernas de desenvolvimento de software ditas ágeis com o que é advogado como boas práticas pelo *Project Management Institute (PMI)* através do seu trabalho *A Guide to the Project Management Body of Knowledge(PMBOK)* (PMI, 2004).

Palavras-chave: engenharia de software, metodologias ágeis, PMBOK, Project Management Institute.

Abstract

In this work was realized a bibliography research looking for the relation between modern software methodologies said agiles with what is advocated as good practices by the Project Management Institute(PMI) in its work A Guide to the Project Management Body of Knowledge (PMI, 2004).

Key Words: software engineering, agile methodologies, PMBOK, Project Management Institute.

Sumário

Lista de Figuras

Lista de Tabelas

1	Introdução	p. 13
1.1	Considerações iniciais	p. 13
1.2	Objetivos	p. 18
1.3	Definição do escopo	p. 18
1.4	Metodologia científica	p. 19
2	Contexto histórico	p. 20
2.1	A história do conhecimento em gerenciamento de projetos	p. 20
2.1.1	Antes de 1940	p. 21
2.1.2	A segunda guerra mundial	p. 24
2.1.3	Os anos 50, o desenvolvimento do gerenciamento de sistemas	p. 25
2.1.4	Os anos 60, Apollo e a década do gerenciamento de sistemas	p. 28
2.2	A curta história da engenharia de software	p. 29
2.2.1	Pré-história	p. 29
2.2.2	Os anos 50	p. 30
2.2.3	Os anos 60	p. 30
2.2.4	Os anos 70	p. 32
2.2.5	Os anos 80	p. 33
2.2.6	Os anos 90	p. 34

2.2.7	Os anos 2000	p. 34
3	A teoria por trás da metodologia	p. 38
3.1	A anatomia de uma metodologia	p. 38
3.1.1	Estrutura de uma metodologia	p. 38
3.1.2	Tipos de metodologias	p. 39
3.1.3	Escopo	p. 40
3.1.4	Conceitos do desenho de metodologias	p. 41
4	Um vôo rápido sobre as diversas metodologias	p. 43
4.1	<i>Extreme Programming</i>	p. 43
4.2	<i>Crystal Clear</i>	p. 45
4.3	<i>Scrum</i>	p. 48
4.4	<i>Feature Driven Development</i>	p. 50
4.5	O PMBOK	p. 53
5	Conclusões	p. 56
	Referências Bibliográficas	p. 59

Lista de Figuras

2.1	Pirâmides de Giza, no Egito (EGYPTIAN..., 2008)	p. 21
2.2	York Minster, catedral de York, Inglaterra (CASTLES..., 2008)	p. 22
2.3	Trabalhadores de ferrovias no nordeste de Utah, 1869 (RAILROADS..., 2008)	p. 23
2.4	Rotas de invasão na Normandia (NORMANDY..., 2008)	p. 25
2.5	Dispositivo nuclear “Jumbo” sendo posicionado para o teste “Trinity” em Alamogordo, Novo México (NEIGHBORHOOD..., 2008)	p. 26
2.6	Primeiro lançamento do míssil Atlas do Cabo Canaveral em 1957 (DECEM- BER..., 2008)	p. 27
2.7	Processo formal de desenvolvimento do projeto SAGE	p. 30
2.8	Modelo cascata de desenvolvimento de software	p. 32
2.9	Tendências de software (BOEHM, 2006)	p. 37
3.1	As três dimensões do escopo de uma metodologia (COCKBURN, 2001) . . .	p. 40
4.1	Processo da metodologia Scrum (SCRUM, 2008)	p. 48
4.2	Diagrama de processos da metodologia FDD (FEATURE..., 2008)	p. 51
4.3	Iteração entre os grupos de processos do PMBOK (PMI, 2004)	p. 54

Lista de Tabelas

1.1	<i>The Standish Group Chaos Report: fatores de sucesso</i>	p. 15
1.2	<i>The Standish Group Chaos Report: estudo de casos</i>	p. 15
1.3	Taxas de sucesso relatadas pelo <i>Standish Group</i>	p. 16
4.1	Marcos na metodologia FDD	p. 52

1 *Introdução*

1.1 Considerações iniciais

Não são necessários muitos anos de experiência na indústria de software para já se colecionar histórias de projetos que terminaram muito longe de suas estimativas iniciais de custos e prazos. Apesar dos grandes esforços já feitos na área, ao que parece algo ainda foge aos olhos mais cuidadosos.

A bibliografia é vasta quando se trata do assunto engenharia de software e metodologias de desenvolvimento de software. Ao longo do tempo, muitos autores e organizações lançaram mão de recursos para melhor compreender este fenômeno.

Em 1975, o célebre livro *The Mythical Man-Month* (BROOKS, 1995) já retratava a realidade dura que iria ser vivida pelas gerações vindouras. Seu autor, Frederick P. Brooks, cunhou a famosa frase:

Adding manpower to a late software project makes it later.¹ (BROOKS, 1995)

Além desta famosa frase, sua obra contém outras tantas igualmente famosas, e apesar de ser uma obra sobre engenharia de software com mais de 30 anos de idade, é considerada muito atual (REVIEW..., 2005). A tendência dos gerentes de projetos de software ainda repetirem os mesmos erros fez com que seu autor, jocosamente, declarasse que seu livro é chamado “A Bíblia da Engenharia de Software” porque “todos lêem, mas ninguém faz nada de acordo com ela!”.

Vinte anos após estes trabalhos, em 1995 o *The Standish Group* fez uma grande pesquisa junto a 365 empresas de software das mais diversas áreas e compilou seu resultado no relatório *Chaos* (THE STANDISH GROUP, 1995) que trazia os seguintes dados:

- Os Estados Unidos despendia mais do que 250 bilhões de dólares por ano em desenvolvimento de software em aproximadamente 175000 projetos. O custo médio de desenvol-

¹Tradução: Adicionar pessoas a um projeto atrasado faz ele mais atrasado.

vimento para uma empresa grande era de US\$2.322.000; para uma empresa média era de US\$1.331.000; e para uma empresa pequena era de US\$434.000.

- 31,1% dos projetos eram cancelados antes deles serem concluídos.
- 52,7% custaram 189% a sua estimativa inicial.

E ainda de acordo com a seguinte classificação de projetos:

- Tipo 1, projeto bem sucedido: o projeto foi terminado no prazo e no custo estimado, com todas as características e funções especificadas inicialmente.
- Tipo 2, projeto posto em risco: o projeto foi terminado e seu produto é operacional, mas passou seu prazo ou seu custo, e oferece bem menos características e funções do que originalmente especificado.
- Tipo 3, projeto cancelado: o projeto foi cancelado em algum ponto durante seu desenvolvimento.

Tem-se que as falhas em projetos não eram igualmente distribuídas de acordo com o tamanho da organização. Contrariando o que poderia ser o senso comum, somente 9% dos projetos em grandes empresas eram bem sucedidos (tipo 1), enquanto as empresas pequenas e médias tinham uma taxa de sucesso de 28% e 16,2% respectivamente.

De acordo com o relatório, um dos principais resultados da pesquisa seria descobrir as causas de tantas falhas. Para tanto, foram entrevistados executivos de TI sobre suas opiniões do que faria um projeto ser bem sucedido. As três maiores razões foram o envolvimento do usuário, suporte do gerenciamento executivo e requisitos claros. Há outros fatores, como pode ser visto na tabela 1.1, mas com estes elementos a chance de sucesso aumentaria enormemente.

Das empresas pesquisadas, foram escolhidos quatro casos para estudo, representativos entre empresas pequenas, médias e grandes, e entre diversas indústrias, duas do tipo 3 e duas do tipo 1 conforme o critério apresentado anteriormente. Os projetos escolhidos foram o DMV, o CONFIRM, o HYATT e o ITAMARATI.

O projeto DMV foi iniciado pelo departamento de trânsito da Califórnia, em 1987, para revitalizar os sistemas responsáveis pelo registro de licenças. Em 1993, depois de terem sido gastos 45 milhões de dólares, o projeto foi cancelado.

Fatores de sucesso do projeto	% de respostas
1. Envolvimento do usuário	15,9%
2. Suporte do gerenciamento executivo	13,9%
3. Requisitos claros	13,0%
4. Planejamento apropriado	13,0%
5. Expectativas realistas	8,2%
6. Marcos menores	7,7%
7. Equipe competente	7,2%
8. Senso de propriedade	5,3%
9. Visão e objetivos claros	2,9%
10. Trabalho duro, equipe focada	2,4%
Outros	13,9%

Tabela 1.1: *The Standish Group Chaos Report*: fatores de sucesso

O projeto CONFIRM foi um projeto iniciado em 1994 pelas empresas *American Airlines*, *Budget Rent-A-Car*, *Marriot Corp.* e *Hilton Hotels*, que depois de 165 milhões de dólares cancelaram seu projeto de reserva de viagens, hotéis e aluguel de veículos.

Enquanto o projeto CONFIRM falhava, o projeto de reserva HYATT do *Hyatt Hotels* foi muito bem sucedido. Hoje, pode-se reservar um quarto através do celular, pedir que o ônibus de cortesia passe no aeroporto e ter suas chaves esperando você no balcão, tudo isso com um prazo e custos menores que o estimado, e com funcionalidades extras.

O projeto ITAMARATI do Banco Itamarati também foi um grande sucesso. O seu projeto para melhoria do atendimento ao cliente teve muitos dos ingredientes-chaves que contribuíram para seu fim no custo e no prazo estimados.

Ao cruzar os dados de fatores de sucesso com os casos de estudo, dando-se um peso adequado aos itens ditos mais influentes no sucesso de um projeto, obteve-se a tabela 1.2 como resultado.

Critério de Sucesso	Pontos	DMV	CONFIRM	HYATT	ITAMARATI
1. Envolvimento do usuário	19	não(0)	não(0)	sim(19)	sim(19)
2. Suporte do Gerenciamento Executivo	16	não(0)	sim(16)	sim (16)	sim(16)
3. Requisitos claros	15	não(0)	não(0)	sim(15)	não(0)
4. Planejamento apropriado	11	não(0)	não(0)	sim(11)	sim(11)
5. Expectativas realistas	10	sim(10)	sim(10)	sim(10)	sim(10)
6. Marcos menores	9	não(0)	não(0)	sim(9)	sim(9)
7. Equipe competente	8	não(0)	não(0)	sim(8)	sim(8)
8. Senso de propriedade	6	não(0)	não(0)	sim(6)	sim(6)
9. Visão e objetivos claros	3	não(0)	não(0)	sim(3)	sim (3)
10. Trabalho duro, equipe focada	3	não(0)	sim(3)	sim(3)	sim(3)
Total	100	10	29	100	85

Tabela 1.2: *The Standish Group Chaos Report*: estudo de casos

Como pode ser visto, pelos dados levantados no próprio relatório, os projetos DMV e CONFIRM tinham pequena chance de sucesso, enquanto o HYATT e ITAMARATI possuíam os ingredientes certos.

O resultado final da pesquisa foi que os projetos de desenvolvimento de software estavam num estado caótico. Uma lista de sugestões e fatores de sucesso para projetos foi elaborada a partir desta pesquisa, e publicada.

Poderíamos questionar a validade desta pesquisa para os projetos atuais, argumentar que estes dados estão velhos e defasados, e que a realidade hoje é outra, com as mais novas técnicas e metodologias, e com todo o aprendizado acumulado pela indústria. Infelizmente, como pode ser visto na tabela 1.3, isto não é verdade. Se abusarmos e extrapolarmos estes dados, vemos que a taxa de sucesso para projetos de software em grandes empresas tem crescido de maneira linear a uma razão de 1% ao ano, e que atingirá a marca de 50% no ano de 2014 (MARASCO, 2006). Não houve nenhuma revolução nestes últimos anos, nada que justificasse um salto notável. Ao que parece, estamos evoluindo na área sim, mas a um passo extremamente lento.

Relatório	Ano	% Sucesso
Primeiro relatório <i>Chaos</i>	1994	16%
<i>Extreme Chaos</i>	2001	28%
Relatório <i>Chaos</i> recente	2003	31%

Tabela 1.3: Taxas de sucesso relatadas pelo *Standish Group*

Um tema é comum na literatura que trata de engenharia e projetos de software. Há um item intangível na área de software que ainda não foi domado, a exemplo da indústria da construção civil, naval ou de defesa. Nos artigos *No Silver Bullet* e “*No Silver Bullet*” *Refired* (BROOKS, 1995), o primeiro com dez anos de intervalo com relação ao célebre livro, e o segundo com mais dez anos em relação ao primeiro artigo, fazem menções ao que o autor chama de complexidade essencial e complexidade acidental. Segundo Brooks, o que fizemos estes anos todos foi atacar a dificuldade acidental existente na produção de um sistema de software, que são todas as dificuldades criadas por nós mesmos, com excesso de documentação, ou ferramentas ruins ou excessivos pontos de controle. Mas ainda não atacamos a dificuldade essencial de como facilitar a criação do modelo mental do problema a ser atacado pela equipe do projeto. Criar este modelo, como Cockburn também coloca (COCKBURN, 2001), é uma atividade intrinsecamente humana, e como tal passa pelas áreas da política, sociologia, psicologia e comunicação. Ainda segundo Cockburn, um projeto de software é uma atividade social, onde o ambiente deve ser favorável para que exista a troca de conhecimento e experiências, para que este modelo mental do que deverá ser o sistema tenha condições de surgir. Os dois autores defendem a posição que esta é uma resposta melhor de onde se encontra a complexidade de todo projeto de software.

Vemos então um cenário onde é mais comum o fracasso de um projeto de software do que seu sucesso. Pelas estatísticas até o momento, um gerente de projetos na área pode ter como certo que o projeto que ele começa hoje terá uma chance de aproximadamente 70% de não ser concluído ou então de ultrapassar em muito suas estimativas de tempo e custos. Temos também que qualquer solução do problema deve passar pela adequada compreensão e habilidosa manipulação da sua natureza humana.

Mas desde 1969 o *Project Management Institute (PMI)* vem estabelecendo as bases para a profissão de gerente de projetos, tendo como uma de suas principais obras de referência o *Project Management Body of Knowledge (PMBOK)*. Com ele, tenta-se estabelecer “a soma do conhecimento na profissão de gerenciamento de projetos” e “identificar que parte do conhecimento de gerenciamento de projetos é reconhecido como boa prática” (PMI, 2004).

*The primary purpose of the PMBOK Guide is to identify that subset of the Project Management Body of Knowledge that is generally recognized as good practice. “Identify” means to provide a general overview as opposed to an exhaustive description. “Generally recognized” means that the knowledge and practices described are applicable to most projects most of the time, and that there is widespread consensus about their value and usefulness. “Good practice” means that there is general agreement that the correct application of these skills, tools, and techniques can enhance the chances of success over a wide range of different projects. Good practice does not mean that the knowledge described should always be applied uniformly on all projects; the project management team is responsible for determining what is appropriate for any given project.*² (PMI, 2004)

Porém, se refletirmos por alguns instantes, vemos que ou estamos falhando em aplicar um conhecimento que hoje está em sua fase de amadurecimento, com quase 40 anos de idade, ou então ele também não responde completamente aos problemas particulares da área de engenharia de software. Tanto é que grupos de cientistas, pesquisadores e praticantes na área de engenharia de software vêm advogando metodologias ditas ágeis que parecem ser, em alguns aspectos, contrárias à visão do PMI em como se gerenciar um projeto nesta indústria. Como

²Tradução: O principal objetivo do Guia PMBOK é identificar o subconjunto do conjunto de conhecimentos em gerenciamento de projetos que é amplamente reconhecido como boa prática. “Identificar” significa fornecer uma visão geral, e não uma descrição completa. “Amplamente reconhecido” significa que o conhecimento e as práticas descritas são aplicáveis à maioria dos projetos na maior parte do tempo, e que existe um consenso geral em relação ao seu valor e sua utilidade. “Boa prática” significa que existe um consenso geral de que a aplicação correta dessas habilidades, ferramentas e técnicas podem aumentar as chances de sucesso em uma ampla gama de projetos diferentes. Ser uma boa prática não implica que o conhecimento descrito deverá sempre ser aplicado uniformemente em todos os projetos; a equipe de gerenciamento de projetos é responsável por determinar o que é adequado para um projeto específico.

J. Davidson, PhD e PMP, argumenta, talvez seja o momento de rever os princípios por trás da maturidade, dos padrões e processos controlados (FRAME, 2008) num mundo em constante mudança, cada vez maiores e mais rápidas. Num mundo onde as empresas desejam ser capazes de aprender e se adaptar ao meio.

Para as organizações da área, este tema é de suma importância, trata-se de uma questão de sobrevivência entender as variáveis deste problema e possíveis soluções.

1.2 Objetivos

Este trabalho tem como objetivo fazer uma revisão do conhecimento acumulado até o momento sobre a área de gerenciamento de projetos de software, sob a perspectiva de metodologias ágeis de desenvolvimento e sob a perspectiva das recomendações feitas pelo PMI através da sua obra de referência, o PMBOK. Verificar as razões históricas por trás de cada perspectiva de gerenciamento, e estabelecer as relações entre as mesmas, se estão realmente em pontos opostos, se tratam de dimensões diferentes do mesmo problema ou se apenas se complementam.

Para tanto, esta obra está dividida em quatro partes: a primeira estabelece o fundo histórico onde se desenvolveu o conhecimento em gerenciamento de projetos; a segunda estabelece alguns conceitos teóricos sobre metodologias de gerenciamento de software, a fim de estabelecer meios de compará-las de forma clara e precisa; a terceira descreve de forma sucinta diversas metodologias de software e as principais boas práticas estabelecidas pelo PMBOK; e finalmente a quarta analisa e conclui sobre o tema proposto.

1.3 Definição do escopo

Apesar da aplicabilidade das recomendações do PMBOK serem mais extensas do que a área de desenvolvimento de software, esta extensão e suas consequências não são analisadas neste trabalho.

Por outro lado, por não existir uma definição precisa do que é uma metodologia de desenvolvimento de software ágil, este trabalho atem-se a definição e aos trabalhos dos especialistas e pesquisadores que iniciaram a sua divulgação, e que assinaram em conjunto o “Manifesto pelo Desenvolvimento Ágil de Software” (MANIFESTO. . . , 2001).

1.4 Metodologia científica

O trabalho apresentado utiliza a modalidade de pesquisa revisão bibliográfica. Foram utilizados como fontes de pesquisa bibliotecas e sítios na rede global de computadores. Os autores e instituições foram escolhidos de acordo com a relevância e contribuição para a área pesquisada, principalmente cientistas e especialistas nas áreas de gerenciamento de projetos, engenharia de software e metodologias ágeis.

2 *Contexto histórico*

George Santayana, escritor e filósofo, cunhou a famosa frase:

Those who cannot remember the past are condemned to repeat it.¹

Que de tão conhecida se tornou clichê, e ganhou diversas variantes. No entanto, ao se interpretar esta frase na visão do contexto histórico do gerenciamento de projetos, vê-se que ela é somente parcialmente verdadeira. Não só de desastres e falhas é povoado o passado, muitas grandes realizações foram feitas. Mas não é verdade que ao não conhecê-las, estamos condenados a repetir seu sucesso. Muito pelo contrário, como veremos adiante, ainda hoje projetos falham por não seguir a receita de sucesso de projetos anteriores similares. Por outro lado, a história está sim repleta de casos de projetos falhos, que não atingiram seus objetivos, numa razão muito maior do que os bem sucedidos. Analisá-los pode trazer luz sobre o que não deve ser feito. E conhecer a história é o caminho para a melhoria contínua nesta viagem de acumulação de conhecimento.

2.1 **A história do conhecimento em gerenciamento de projetos**

Para entendermos melhor as práticas recomendadas pelo PMBOK, na sua função de repositório do conhecimento da profissão de gerente de projetos, é importante entendermos suas raízes, a origem das suas recomendações e práticas. Para tanto, uma breve revisão histórica se faz necessária. Esta revisão será fortemente baseada nos trabalhos de Peter W. G. Morris em *The Management of Projects* (MORRIS, 1997).

¹Aqueles que não se lembram do passado estão condenados a repetí-lo.

2.1.1 Antes de 1940

Desde as civilizações antigas são realizados grandes projetos, como as pirâmides (ver figura 2.1) ou os aquedutos romanos. Estes primeiros projetos utilizavam uma quantidade enorme de pessoas, e envolviam toda a comunidade, como consequência da atribuição de poderes divinos a autoridade máxima do projeto e, ao mesmo tempo, seu patrocinador. Grandes obras militares gregas, romanas, persas ou chinesas se utilizavam largamente de trabalho escravo, que eram de propriedade do governo ou do empreiteiro. Percebe-se já nesta época a figura do responsável pela obra, como um arquiteto, que delegava o trabalho para empreiteiros, como no caso do *Coliseum* que foi construído por quatro empreiteiras. Os contratos com as empreiteiras já continham detalhes como a especificação do trabalho a ser realizado, o material a ser usado, as garantias e método de pagamento. Já se tinha a consciência da relevância dos prazos.



Figura 2.1: Pirâmides de Giza, no Egito (EGYPTIAN... , 2008)

As catedrais medievais, por sua vez, representam a dominância do sagrado sobre o mundano. Ao contrário de super projetos anteriores, não se dava importância ao tempo necessário para a realização destas obras, levando até mesmo gerações para serem completadas (ver figura 2.2).

Entre os séculos 15 e 17 o número de grandes projetos aumentou, com o surgimento da engenharia como ciência. De novo era enfatizado a importância dos prazos nos projetos. O arquiteto trabalhava não só como projetista, mas como estimador, comprador, organizador, inspetor e responsável financeiro. Conforme as relações comerciais se sofisticavam, as transações contratuais se tornavam parte importante de todo projeto.

Nos séculos 17 e 18 a sofisticação dos projetos de engenharia aumentaram ainda mais.



Figura 2.2: York Minster, catedral de York, Inglaterra (CASTLES. . . , 2008)

Aqueles que constroem os projetos agora estavam organizacionalmente e contratualmente separados daqueles que projetavam e estimavam seu trabalho.

É importante notar que, de todas as áreas que se utilizam das técnicas de gerenciamento de projetos modernas de hoje, a mais antiga é a área de construção e engenharia civil. Mas o desenvolvimento da ciência do gerenciamento de projetos foi realizado principalmente nas áreas de defesa e aeroespacial como veremos adiante, e depois trazido de volta suas contribuições para a área de construção.

Entre a metade do século 19 e a metade do século 20, viu-se o florescer de uma nova era industrial, com avanços em tecnologias relacionadas. Grandes construções como arranha-céus, ferrovias (ver figura 2.3), usinas hidrelétricas, sistemas de tratamento de água e esgoto, entre outros, junto com novas indústrias de produção em massa como a do aço, do petróleo, petroquímica, farmacêutica, energia, telefonia, de aviação, médica e outros setores.

O envolvimento do governo nesta época era baixo. Esta era a Era do empreendedor, do capitalista. Tanto as minas quanto ferrovias e empresas de telefonia eram todas privadas. O capital era abundante e as regulamentações eram simples. As primeiras teorias de gerenciamento científico, com autores como Weber, Taylor, Gilbreth e Gantt, enfatizavam regras simples de administração e organização.

A escola científica deu origem a duas técnicas de gerenciamento de projetos. O gráfico de barras Gantt, produzido por Henry Laurence Gantt para o governo dos EUA para o gerencia-



Figura 2.3: Trabalhadores de ferrovias no nordeste de Utah, 1869 (RAILROADS..., 2008)

mento da produção do *Frankford Arsenal* em 1917. E o não tão famoso *Harmonygraph* criado por Adamiecki por volta de 1896 na Polônia. Este último é o precursor das técnicas de gerenciamento de redes que ficaram famosas nos anos 60 através do CPM (*Critical Path Method*) e do PERT (*Program Evaluation and Review Technique*). Outro precursor das redes foi a análise do caminho, desenvolvido por Wright em 1918 como uma forma de decompor as relações e expressá-las como formas causais estatisticamente.

No final dos anos 20, foi desenvolvido por *Procter and Gamble* o conceito de gerenciamento de produtos. O gerenciamento de produtos era a prática de fazer com que um gerente fosse responsável pela propaganda, planejamento e controle de um produto. Como em gerenciamento de projetos, já era enfatizado a importância da integração entre diversas áreas funcionais da empresa.

Durante os anos 30, a divisão de materiais da *US Air Corps* começou progressivamente a mover-se para uma estrutura de escritório de projetos. Ao mesmo tempo, algumas empresas como a Exxon criaram a figura do engenheiro de projeto, um engenheiro que acompanhava o processo através dos diversos departamentos funcionais. Nesta época a estrutura organizacional que prevalecia era a piramidal ou funcional, mas em 1937, Urwick, da escola de Fayol de administração, edita um livro onde Gulick escreve um artigo que propõe o papel de coordenador, que administraria as tarefas envolvendo diversas áreas funcionais. Esta é a primeira aparição da organização horizontal no meio acadêmico.

2.1.2 A segunda guerra mundial

Operações militares tem uma estrutura muito parecida com a de um projeto. Elas normalmente têm objetivos claros, precisam de planejamento cuidadoso, dependem de uma boa liderança, comunicação e controle.

Na segunda guerra mundial, viu-se o surgimento da ciência como grande aliada no gerenciamento de um grande número de operações. Apesar disto, viu-se poucos dos mecanismos formais e técnica que hoje encontramos no gerenciamento moderno.

No entanto, a segunda guerra mundial deu três contribuições importantes para o estudo do gerenciamento de projetos moderno, a saber: a Pesquisa Operacional, o Dia D e o *Manhattan Project*.

Pesquisa Operacional

A Pesquisa Operacional é a coleta e análise de informações do cotidiano usando princípios científicos para a otimização de processos. Foi muito utilizada para a otimização do trabalho e logística durante a segunda grande guerra.

O Dia D

O Dia D foi uma enorme ação de invasão coordenada em várias frentes (ver figura 2.4), que exigiram um enorme planejamento, tinha um objetivo bem definido e um Comandante Supremo. Mas apesar destas características, utilizou-se de técnicas tradicionais, sem grandes contribuições para o gerenciamento de projetos moderno.

O *Manhattan Project*

O desenvolvimento da primeira bomba atômica foi um dos maiores projetos industriais de pesquisa já realizados. Ele envolveu 600000 pessoas, custou 2 bilhões de dólares, era de alto risco e grande complexidade.

Quando o projeto em 1942 formalmente teve início, a teoria física de como controlar uma reação nuclear em cadeia ainda não era completamente compreendida. Somente quantidades microscópicas de material radioativo estavam disponíveis. Em 1944, dois anos e meio depois, grandes usinas de beneficiamento de material radioativo, e as duas bombas, já estavam construídas. Tudo isso foi realizado sobre grande segredo e autoridade direta do presidente

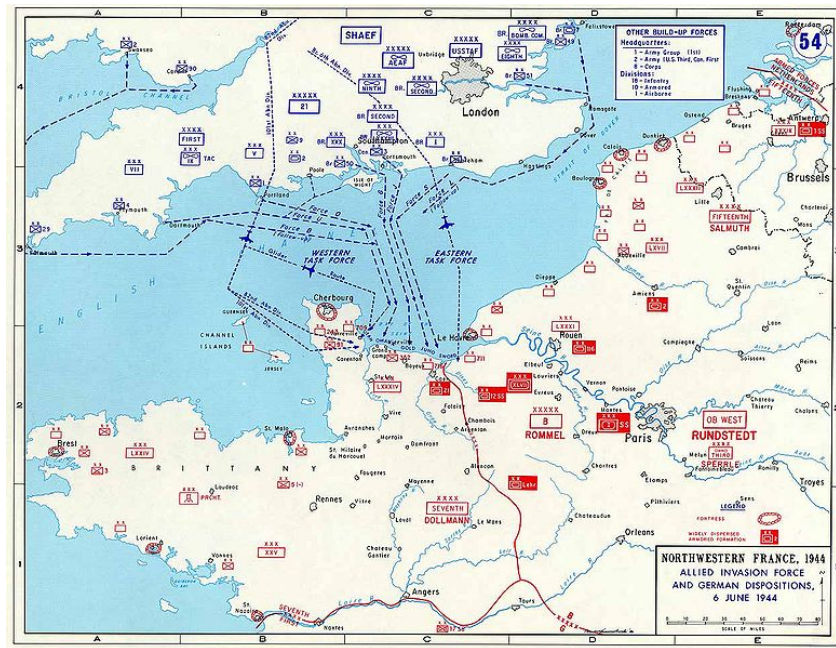


Figura 2.4: Rotas de invasão na Normandia (NORMANDY... , 2008)

Roosevelt.

A essência do projeto foi a urgência e a incerteza. Era preocupação de todos o fato de a Alemanha nazista estar no caminho de criar sua própria bomba, e mesmo não compreendendo completamente os princípios da reação nuclear, era consenso o fato de que os maiores desafios para a construção da bomba eram organizacionais e de engenharia.

2.1.3 Os anos 50, o desenvolvimento do gerenciamento de sistemas

Nos anos 50, tanto as técnicas CPM quanto PERT foram desenvolvidas, e o gerenciamento de sistemas se torna um jargão comum, principalmente pela necessidade de desenvolvimento de aeronaves e mísseis de longo alcance em curtíssimo espaço de tempo pela USAF (*US Air Force*).

Em 1950, com a guerra da Coreia, houve um aumento da necessidade de bombardeiros B47. Este aumento levou a necessidade de melhor coordenação entre engenharia e produção. Como resultado, projetos conjuntos foram criados, e em 1952 isto já era comum.

Com a ameaça de mísseis intercontinentais contendo ogivas nucleares em posse da antiga União Soviética, foi criado um grupo especial com a função de coordenar os esforços na criação de contra medidas. Este grupo teria como função coordenar os esforços de vários grupos no projeto do míssil Atlas (ver figura 2.6). Um comitê dirigido por John von Neumann emitiu um relatório que considerava esta coordenação mais crítica ao sucesso do projeto do que qualquer



Figura 2.5: Dispositivo nuclear “Jumbo” sendo posicionado para o teste “Trinity” em Alamo-gordo, Novo México (NEIGHBORHOOD..., 2008)

dificuldade técnica.



Figura 2.6: Primeiro lançamento do míssil Atlas do Cabo Canaveral em 1957 (DECEMBER... , 2008)

Nesta época já se vê a necessidade da gerência do projeto de um produto como uma entidade única, desde o projeto, passando pela implementação até seu uso. Mas já nessa época, e em projetos subsequentes, vê-se o surgimento de práticas não tão boas, como a excessiva burocracia e padrões, enfatizando o processo de gerenciar outros e não o de se fazer a engenharia e integração por si mesmo. Pior ainda, como quem especificava o trabalho não trabalhava em conjunto com quem produzia, o ambiente para requisitos irrealistas ou inferiores estava configurado.

O projeto Atlas foi o precursor da técnica de compressão de tempo do início de atividades dependentes de forma concorrente, e não sequencial como era até então. O uso desta técnica aumenta o risco do projeto, e foi dada ênfase na melhoria das técnicas de controle de qualidade dos fornecedores contratados.

Mais do que o programa Atlas, o programa Polaris aumentou a orientação dos departamentos por projetos. Também fez surgir uma nova técnica de gerência de projetos, o PERT.

Um time formado por gerentes de projeto da marinha americana e os consultores de gerenciamento Booz, Allen & Hamilton e a *Lookhead Corporation* levantaram os seguintes requisitos para a criação de um novo sistema de controle de projetos:

- deve conter uma estimativa para cada atividade
- como algumas atividades podem envolver incertezas, deve conter também a função de distribuição da probabilidade do tempo de conclusão.
- conhecimento preciso da ordem das atividades.

Além das técnicas de rede desenvolvidas através do PERT, conceitos como o do caminho crítico, como sendo o caminho da seqüência de eventos que determina o prazo do projeto, foram identificados nessa época.

Técnicas como a de se ter a sala de guerra, uma área com informações gerenciais integradas de todos os projetos em andamento, surgem nessa época, e também reuniões semanais dos gerentes de cada projeto, onde informam dados sobre o andamento das atividades. Medir o desempenho planejado com o real se torna um dos principais objetivos.

Interessante notar que caminhos diferentes levam a resultados diferentes. No mesmo período que foi desenvolvido o PERT, o CPM foi desenvolvido por Du Pont na indústria de construção civil. O CPM é muito similar ao PERT, ambos usam redes e setas para representarem atividades, mas a construção civil é uma indústria muito diferente da indústria de pesquisa militar, sendo suas tecnologias e processos muito bem conhecidos. Esta é a razão da falta de preocupação com a probabilidade de sucesso de uma atividade. Também por esta razão viu-se nos primeiros desenvolvimentos uma preocupação maior no CPM com o custo das atividades, já que o ramo da construção é um ramo comercial competitivo.

2.1.4 Os anos 60, Apollo e a década do gerenciamento de sistemas

Com a eleição de John F. Kennedy, Robert S. McNamara recebe o cargo de Secretário da Defesa, e com ele uma série de reformas no processo de gerenciamento interno da USAF são executadas. Grande ênfase no planejamento e definição de contratos, controle de orçamento e requisitos, controle integrado de logística e qualidade, e o uso da Estrutura Analítica do Projeto (*Work Breakdown Structure*) como padrão na comunicação com fornecedores.

Muitas destas técnicas foram aplicadas nos programas da NASA, e atingiram grande popularidade através do programa Apollo.

Desde muito cedo a NASA já era consciente das deficiências do uso do PERT para controle de custos. Após estudos, passou a utilizar a EAP como forma de controle de custos, com ênfase no controle de marcos principais e interfaces entre fornecedores. No entanto, depois de problemas com a quantidade de formas diferentes com que era apresentado os relatórios de acompanhamento do custo do projeto, foi desenvolvido o sistema de Valor Agregado (*Earned Value system*), que se tornou um item importante no conjunto de ferramentas do gerente de projetos.

O projeto Apollo é considerado na academia, no mundo dos negócios e pelo público comum o paradigma do gerenciamento moderno de projetos. Pela sua complexidade, uma grande importância foi dada no controle de interfaces entre sistemas.

Como sabemos, o projeto Apollo foi um grande sucesso. Mas isso não implica que não foram experimentadas toda a sorte de problemas com seus prazos. Já em relação a seus custos, por ser um projeto governamental na fase da corrida espacial, não teve grandes restrições, e o projeto custou na época o total de 25,5 bilhões de dólares.

2.2 A curta história da engenharia de software

2.2.1 Pré-história

Os pioneiros na história da computação não tinham um computador para si. Tinham que escrever seus programas e executá-los na sala de processamento. O software tinha um custo muito baixo se comparado com o custo do hardware, e todo hardware era de aplicação específica e substituído praticamente a cada dois anos, o que fazia necessário se reescrever todo código. A necessidade de se reescrever o código a cada mudança de hardware fez surgir as primeiras linguagens de alto nível, como FORTRAN, COBOL e ALGOL. Os fornecedores de hardware normalmente forneciam o software gratuitamente, já que o hardware não funcionaria sem o software.

O supervisor de Barry Boehm mostrou-lhe o computador ERA 1103 que ocupava uma grande sala, e disse-lhe, em seu primeiro dia no trabalho:

“Now listen. We are paying \$600 an hour for this computer and \$2 an hour for you, and I want you to act accordingly.”² (BOEHM, 2006)

²Tradução: Agora escute. Nós estamos pagando US\$600 a hora por este computador e US\$2 a hora por você, e eu quero que haja de acordo.

O que era bem de acordo com a economia da época.

A noção de reuso aos poucos surgia, com pequenos grupos de empresas e acadêmicos trocando software gratuito entre si, numa pré-gestação do que viria a ser mais tarde o movimento de software livre.

2.2.2 Os anos 50

O projeto de software mais ambicioso da época foi o desenvolvimento do *Semi-Automated Ground Environment* (SAGE) para o sistema de defesa canadense e norte-americano. Nesta época, a metodologia de desenvolvimento de software era uma cópia do que era a metodologia de desenvolvimento de hardware, com métodos formais de especificação, validação e testes (ver figura 2.7).

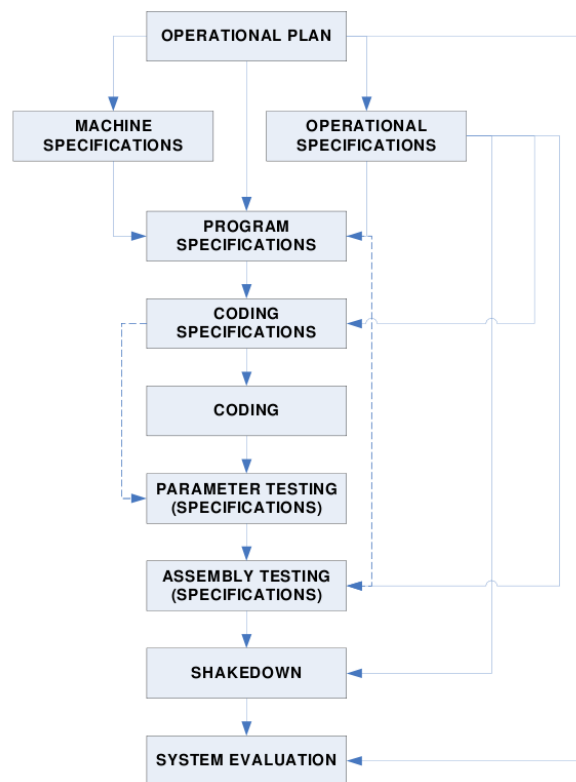


Figura 2.7: Processo formal de desenvolvimento do projeto SAGE

2.2.3 Os anos 60

Nos anos 60 foi-se percebendo que a natureza da produção do software é diferente da natureza da produção do hardware. É muito mais fácil corrigir todas as instalações de um software defeituoso, basta corrigí-lo e realizar cópias a um custo muito baixo, diferente de um hard-

ware que necessitaria de alterações na linha de montagem e substituição mecânica em cada instalação. Por este motivo, muitas empresas começaram a adotar o estilo de programação codificar-e-corriger (*code-and-fix*), ao invés de métodos mais formais e caros.

Percebeu-se também que toda atividade de produção e manutenção de software não poderia ser modelada através dos modelos de manutenção de hardware. O software é invisível, não tem peso nem ocupa espaço, mas começa a ter um alto custo. É difícil dizer se um projeto de software está dentro do prazo, e se você adiciona mais pessoas ao projeto, ele ficará mais atrasado (como já dizia Brooks). Um software tem muitos estados, modos, e caminhos para serem testados. Winstow Royce, num artigo clássico em 1970, disse:

In order to procure a \$5 million hardware device, I would expect a 30 page specification would provide adequate detail to control the procurement. In order to procure \$5 million worth of software, a 1500 page specification is about right in order to achieve comparable control.³ (ROYCE, 1987)

Um outro grande problema com os métodos de engenharia aplicados a computação foi que, no ritmo de crescimento da demanda por profissionais, começou a faltar mão de obra especializada, e os grandes projetos começaram a contratar pessoas menos especializadas e até de outras áreas, mesmo graduados em ciências humanas e biológicas. Para estes profissionais o estilo de codificação codificar-e-corriger era bem adequado, já que eles podiam utilizar de sua natural criatividade, mas com isso aumentou a quantidade de código espaguete e surgiu a figura do cowboy do teclado, aquele que consegue com seu código incompreensível e muitas horas extras salvar projetos atrasados. Também surgia na mesma época a subcultura hacker.

Mas alguns projetos não sucumbiram ao modo de produção codificar-e-corriger, como o projeto do sistema operacional IBM OS-360 e os projetos Gemini, Mercury e Apollo da NASA. Todos estes foram trabalhados de acordo com as mais modernas ferramentas de engenharia da época, e obtiveram grande êxito, apesar de alguns atrasados ou a um grande custo.

A situação em que se encontrava a indústria de software levou a duas conferências em 1968 e 1969 da *NATO Science Committee* com vários pesquisadores e especialistas da época, que oficialmente deu origem a Engenharia de Software. Foram estabelecidas as práticas que deveriam orientar as agências governamentais e empresas, e que metodologias mais disciplinadas deveriam surgir para suprir a demanda de projetos de maior escala.

³Para adquirir um dispositivo de hardware de US\$5 milhões, eu esperaria que uma especificação com 30 páginas provesse o detalhamento adequado para controlar a aquisição. Para adquirir um software no valor de US\$5 milhões, uma especificação de 1500 páginas é necessária para conseguir controle comparável.

2.2.4 Os anos 70

Houve uma reação ao modo de programar codificar-e-corrigir, surgindo métodos mais organizados de produção, sintetizando o que havia de melhor do desenho e produção de hardware, com as modificações exigidas para a produção de software. Este cuidado pode ser exemplificado pela carta de Dijkstra à *ACM Communications: Go To Statement Considered Harmful* (DIJKSTRA, 1968), e um artigo de Böhm e Jacopini (BÖHM; JACOPINI, 1966), que mostravam que programas sequenciais sempre podiam ser escritos com apenas desvios condicionais e laços, iniciando o movimento da Programação Estruturada.

O resultado deste movimento por maior disciplina resultou em duas vertentes: uma de métodos formais de programação, que focava na corretude do programa via prova matemática ou na construção do programa usando métodos do cálculo; e a outra menos formal, usando programação estruturada e técnicas gerenciais.

O sucesso da programação estruturada trouxe outras técnicas e melhorias, como os princípios da modularidade, coesão, redução da superfície de contato, e tipos de dados abstratos.

Um grande marco histórico foi a criação do modelo cascata de desenvolvimento (ROYCE, 1987). Este modelo enfatizava a produção seqüencial do software, com verificações iterativas fechadas em cada fase para evitar a propagação de erros e o custo da correção tardia (ver figura 2.8).

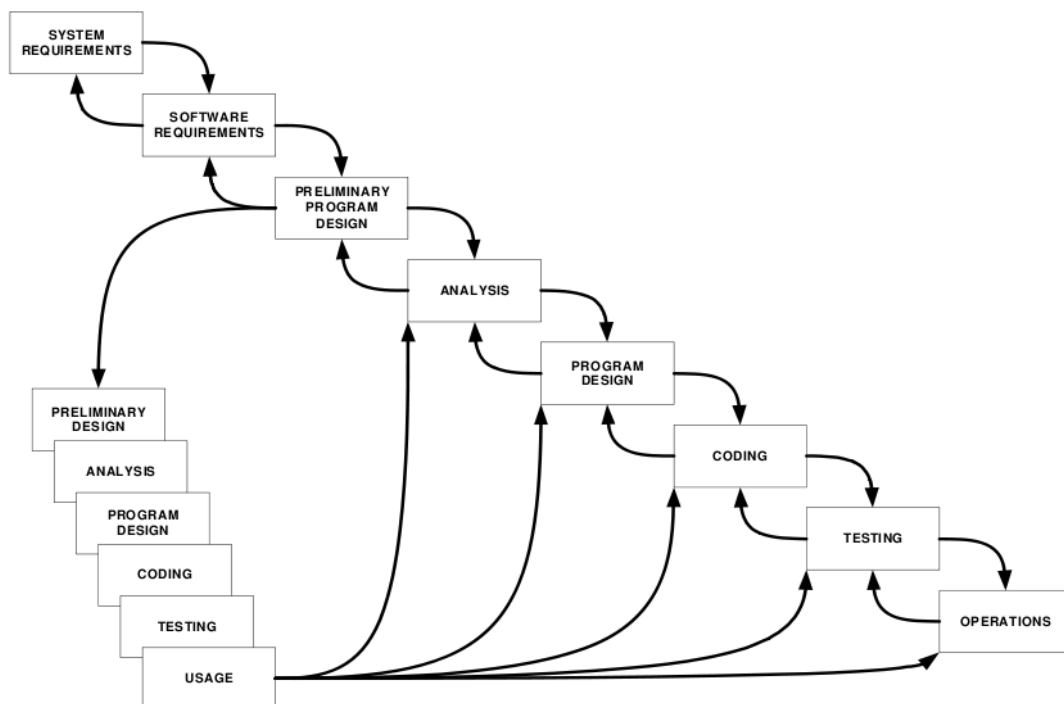


Figura 2.8: Modelo cascata de desenvolvimento de software

Outro efeito positivo de processos mais rígidos foi o estímulo a processos quantitativos mais efetivos na engenharia de software. No entanto, no fim dos anos 70 os problemas voltaram a crescer, com o excesso de formalidade e o uso de processos seqüências. Métodos formais têm problemas com a escalabilidade e a usabilidade pela maioria dos desenvolvedores médios. O modelo seqüencial em cascata se mostrou uma metodologia pesada, orientado a documentos, lenta e cara.

2.2.5 Os anos 80

A crise de software se mostra com força total. Projetos ultrapassam seus custos, há danos financeiros e até perdas de vidas por falhas em softwares. Uma série de iniciativas, utilizando-se dos resultados dos métodos quantitativos empregados anteriormente, começaram a atacar os principais pontos de falhas, como por exemplo o alto custo de testes tardios ou o custo pela falta de requisitos claros.

A falha pelo não cumprimento de técnicas e metodologias já consagradas fez com que o Departamento de Defesa norte-americano financiasse o projeto do *CMU Software Engineering Institute* para o desenvolvimento de um modelo de maturidade de software, o *Capability Maturity Model for Software* (SW-CMM), e desenvolvesse métodos para determinar a maturidade de uma organização. Este foi fortemente influenciado pelas experiência da IBM no desenvolvimento de seus projetos, e apesar de ter sido criado com a intenção de ser independente de metodologias, tem uma forte tendência seqüencial, como o modelo de desenvolvimento em cascata.

O medo de perder licitações fez com que muitas empresas investissem no modelo de maturidade, e muitas relataram benefícios devido ao menor retrabalho. Isto fez com que o modelo de maturidade ganhasse popularidade.

Os anos 80 viram uma série de melhorias na produtividade utilizando-se de sistemas especialistas, linguagens de alto nível, orientação a objetos, estações de trabalho poderosas e programação visual. Todas elas foram colocadas em perspectiva no trabalho de Brooks *No Silver Bullet* (BROOKS, 1995), onde ele separa o que são tarefas acidentais do desenvolvimento de software, tarefas estas repetitivas que podem ser evitadas através da automação, do que são as tarefas essenciais, aquelas que são necessária e que precisam de trabalho intelectual, bom senso e colaboração. Brooks defende em seu trabalho como candidatos para enfrentar as dificuldades essenciais o uso de projetistas de software experientes, prototipagem, desenvolvimento evolucionário e evitar o trabalho via reuso.

O reuso de componentes de software foram um dos maiores agentes do aumento da produtividade nos anos 80. O uso de softwares na infraestrutura (sistemas operacionais, bancos de dados) e ferramentas (construtores de interfaces com o usuário) evitaram muito desenvolvimento e retrabalho.

2.2.6 Os anos 90

O desenvolvimento orientado a objetos ganhou mais força, principalmente com a criação de padrões de desenvolvimento, linguagens de descrição de arquiteturas e o desenvolvimento da *Unified Modeling Language* (UML).

A contínua expansão da rede global de computadores acirrou ainda mais a competição no mercado de software, e com isso a mudança das metodologias de modelos seqüenciais para modelos concorrentes da engenharia de requisitos, desenho, e codificação.

A criação de produtos cada vez mais iterativos fez com que o usuário se tornasse cada vez mais exigente e imprevisível, tornando mais difícil a manutenção de modelos seqüenciais de desenvolvimento.

O modelo espiral de desenvolvimento (BOEHM, 1986) foi criado com o intento de controlar o processo concorrente de construção de software. Foi adotado também pela Rational/IBM no seu *Rational Unified Process*, e como tal foi utilizado em um grande número de projetos.

Outra forma de desenvolvimento concorrente que contribuiu muito nos anos 90 foi o desenvolvimento de software de código aberto. Das raízes da cultura hacker dos anos 60, estabeleceu sua presença institucional com a criação da *Free Software Foundation* e o *GNU General Public License* por Richard Matthew Stallman. Com estas contribuições foram criadas as condições para o surgimento e evolução de inúmeros pacotes de software úteis e gratuitos, como o compilador C GCC e o editor Emacs. Outros grandes acontecimentos no mundo do software livre foram a criação do núcleo do Linux por Linus Torvald (1991), a criação do *World Wide Web Consortium* e o livro *The Cathedral and the Bazaar* (RAYMOND, 1999) com uma análise concisa no movimento de software livre e suas consequências.

2.2.7 Os anos 2000

Os anos 2000 viram um aumento das tendências dos anos 90, com a aceleração da velocidade das mudanças na tecnologia da informação, nas organizações, no ambiente competitivo e global. Estas mudanças rápidas aumentaram a insatisfação com o excesso de planejamento,

especificação, documentação e outros impostos por modelos de maturidade e contratos.

O início dos anos 2000 viram o surgimento do movimento ágil de desenvolvimento de software e diversas metodologias seguindo seus princípios, com representantes como o *Adaptive Software Development*, *Crystal*, *Dynamic Systems Development*, *Extreme Programming*(XP), *Feature Driven Development* e o *Scrum*. Os maiores representantes deste movimento se encontraram em 2001 e escreveram o *Agile Manifesto* (MANIFESTO..., 2001), colocando quatro valores como preferenciais:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.⁴

A mais conhecida e aplicada metodologia dentre as ágeis foi a XP, que se mostrou efetiva em diminuir a curva de custo de mudanças no tempo. Mostrou-se efetiva ao tratar projetos pequenos, com uma equipe bem treinada e requisitos que se alteravam rapidamente.

Os métodos ágeis se encaixam na necessidade de desenvolvimento de incrementos de acordo com a priorização do maior valor para o usuário, e as mudanças em suas prioridades. A tendência é fazer com que a tecnologia se adapte às pessoas, e não o contrário. Isto se reflete cada vez mais nos critérios de seleção de produtos, com ênfase cada vez maior no valor agregado e usabilidade, ao contrário do que se tinha antes, que era o critério do número de características e o custo. Estas tendências norteiam as prioridades na escolha de produtos e processos, estratégias de propaganda e meios de sobrevivência competitiva.

Uma característica dos novos produtos de software que os torna um grande desafio é a que os requisitos não são mais pré-especificados, mas emergem durante o uso e aprendizado por parte do usuário. Estes requisitos seguem uma pirâmide como a pirâmide de necessidades de Maslow, onde após as necessidades básicas são satisfeitas, novas e mais sofisticadas necessidades surgem. Neste ambiente, processos adaptativos e orientados a diminuição dos riscos são mais recomendados, ao invés de processos com excessivo planejamento, processos em cascata,

⁴Tradução:

- Indivíduos e interações sobre processos e ferramentas.
- Software funcionando sobre extensiva documentação.
- Colaboração do cliente sobre negociação de contratos.
- Resposta a mudanças sobre seguir um plano.

modelos enfatizando a repetição e a otimização. Estudos de engenharia de software orientados ao valor agregado (JAIN; BOEHM, 2005) tentam hoje atacar estes novos desafios.

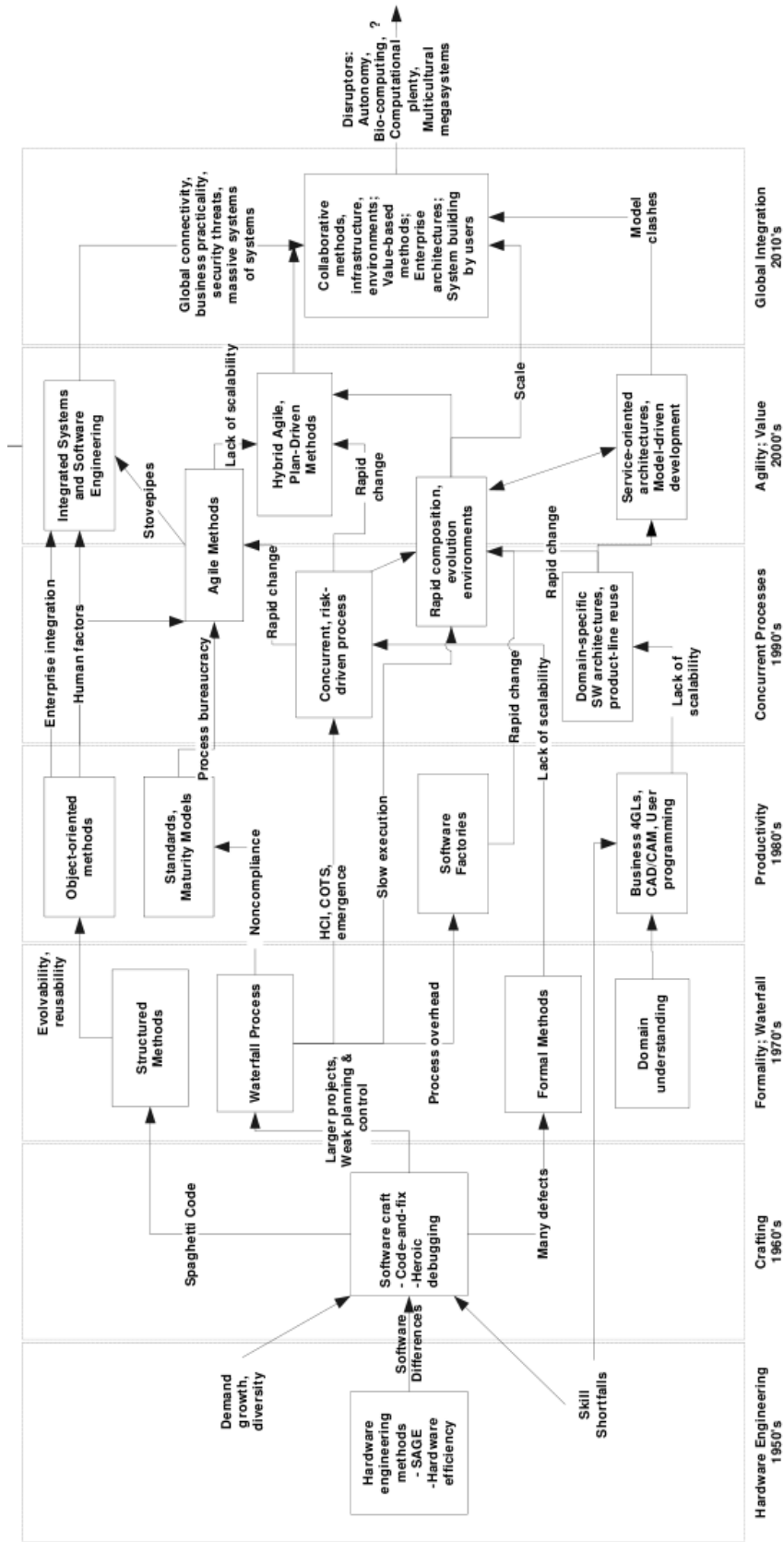


Figura 2.9: Tendências de software (BOEHM, 2006)

3 *A teoria por trás da metodologia*

3.1 A anatomia de uma metodologia

Antes de estudar com mais detalhes algumas metodologias representativas da linha ágil, se faz necessário definir o que é uma metodologia, seus termos e estrutura. Uma metodologia é uma forma de trabalho organizada de um grupo de forma a produzir um resultado. Ou melhor, “A methodology is the conventions your group agrees to”¹ (COCKBURN, 2001). Desta forma, fica evidente a característica social de uma metodologia, e de que mesmo uma empresa pequena possui uma metodologia, mesmo que ela não esteja descrita em algum livro ou tenha recebido o aval de uma grande instituição certificadora. Algumas grandes empresas descrevem as suas metodologias que obtiveram êxito em sua missão, e estas se tornam populares, de forma a serem recebidas pelo público como a forma correta de se obter o mesmo resultado. Mas como cada organização tem seus próprios valores e variações culturais, na maioria dos casos sua implantação como descrita pelo grupo de origem em um outro grupo pode trazer resultados adversos.

3.1.1 Estrutura de uma metodologia

Algumas estruturas se repetem em todas as metodologias, seja ela na área de desenvolvimento de software, construção ou literatura. Elas são (COCKBURN, 2001):

Papéis. Quem você emprega, com qual função, quais as habilidades necessárias.

Habilidades. As competências necessárias para cada papel. Normalmente a habilidade de uma pessoa em sua função é um produto do seu treinamento pelo seu talento.

Equipes. Os papéis que trabalham juntos para obterem um resultado. Em um pequeno projeto, pode haver apenas uma equipe, mas em grandes projetos podem existir diversas equipes coordenadas.

¹Uma metodologia é as convenções acordadas pelo seu grupo

Técnicas. Os procedimentos que as pessoas realizam para cumprir suas tarefas.

Atividades. Como as pessoas utilizam seu tempo. Planejando, programando, testando ou em reuniões são exemplos de atividades.

Processos. Como as atividades trabalham juntas no tempo, com possíveis pré e pós condições entre atividades. Algumas metodologias têm seu foco em processos, em como o trabalho flui entre os membros do projeto.

Produtos. O que o projeto constrói. Alguns produtos podem ter um tempo de vida curto, ou mais longo que o próprio projeto.

Marcos. Eventos que marcam o progresso ou fim das atividades. Um marco tem duas características: ocorre num instante do tempo e ou ocorre completamente ou não ocorre, nunca pode ocorrer parcialmente.

Padrões. As convenções que uma equipe adota em relação a ferramentas, produtos e políticas de trabalho.

Qualidade. As características desejadas para atividades e produtos.

Valores da equipe. Muitos itens da metodologia trabalham de acordo com os valores da equipe. Uma equipe agressiva que valoriza retorno rápido trabalha de forma diferente do que uma que valorize estar com a família e sair do trabalho no horário.

3.1.2 Tipos de metodologias

(MAIER; RECHTIN, 2000) categoriza as metodologias como sendo normativas, racionais, participativas ou heurísticas:

Normativas são aquelas baseadas em soluções ou seqüências de passos conhecidos por funcionar graças ao uso de disciplina.

Racionais são baseadas em métodos e técnicas.

Participativas buscam o envolvimento do cliente na elaboração da solução.

Heurísticas são baseadas em lições aprendidas.

Conforme o conhecimento em uma área cresce, suas metodologias tendem a ir de heurísticas a normativas.

3.1.3 Escopo

O escopo de uma metodologia é o conjunto de atividades e papéis que ela pretende incorporar (COCKBURN, 2001). Neste sentido, muitas metodologias de desenvolvimento de software só estão preocupadas em determinar os papéis e atividades necessários para o desenvolvimento de código fonte, sem preocupação com as atividades comerciais, financeiras e algumas vezes até de recursos humanos necessárias para o projeto.

O escopo de uma metodologia pode ser caracterizado em três eixos: cobertura do ciclo de vida, cobertura de papéis e cobertura de atividades (ver figura 3.1).

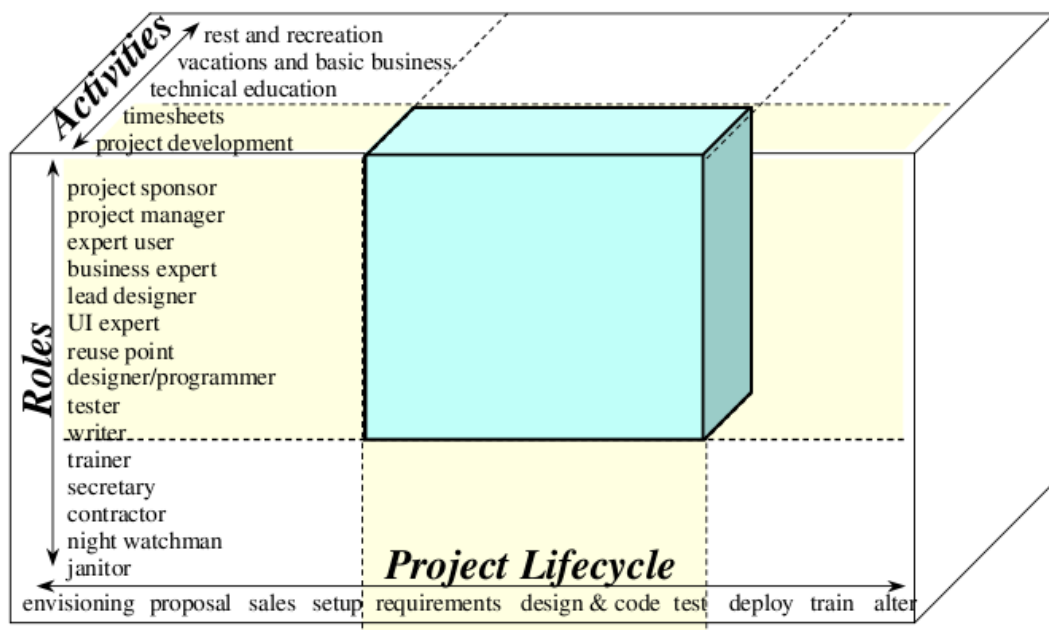


Figura 3.1: As três dimensões do escopo de uma metodologia (COCKBURN, 2001)

A cobertura do ciclo de vida indica quando a metodologia começa seu trabalho no projeto, e quando termina.

A cobertura de papéis refere-se a quais papéis estão sob o seu domínio.

A cobertura de atividades define quais atividades estão sob discussão.

Sob este prisma, vê-se que definir o escopo de metodologias faz com que metodologias que a primeira vista parecem ser contraditórias na verdade apenas têm escopos diferentes. Algumas metodologias do início da história da engenharia de software tinham um escopo muito pequeno, talvez até mesmo insuficiente para o seu trabalho. As metodologias ágeis estão preocupadas com o desenvolvimento de código fonte e o que pode influenciar de forma direta, enquanto as recomendações do PMBOK têm um escopo muito mais abrangente, como veremos mais

adiante.

3.1.4 Conceitos do desenho de metodologias

De acordo com Cockburn (COCKBURN, 2001), a discussão do desenho de metodologias passa pela definição dos seguintes termos: tamanho da metodologia, cerimônia e peso; tamanho do problema; tamanho do projeto; criticidade do sistema; precisão; acurácia; relevância; tolerância; visibilidade; escala; e estabilidade.

Tamanho da metodologia. O número de elementos de controle na metodologia. Cada padrão, produto, atividade, medida de qualidade e técnica descrita na metodologia é um elemento de controle.

Cerimônia da metodologia. A precisão e o intervalo de tolerância da metodologia. Uma cerimônia muito grande implica em controles rígidos. Uma equipe pode escrever casos de uso em quadros brancos e revisá-los durante o almoço, e outra pode preencher um formulário com três páginas e revisá-los numa reunião que durará uma semana. Ambas estão escrevendo casos de uso e os revisando, uma com mais cerimônia que outra.

Peso da metodologia. O produto do tamanho pela cerimônia. O número de elementos de controle multiplicado pela cerimônia necessária em cada um deles.

Tamanho do problema. O número de elementos do problema e sua correlação. As metodologias em geral se limitam a atacar problemas de um certo tamanho, especificado pelo seu autor, ou então especificam meios do usuário da metodologia de como adaptá-la.

Tamanho do projeto. O número de pessoas cujos esforços precisam ser coordenados.

Criticidade do sistema. O dano causado por defeitos no produto do projeto. Cockburn (COCKBURN, 2001) classifica a criticidade do projeto em perda de conforto, perda de pouco dinheiro, perda de muito dinheiro e perda de vidas.

Precisão. Qual a precisão que se deseja obter de um determinado tópico. Alguns autores de metodologias desejam obter mais precisão o mais cedo possível.

Acurácia. Qual a corretude se deseja obter de um determinado tópico. A maioria das metodologias trabalham no sentido de obter modelos mais precisos e mais corretos com o passar do tempo.

Relevância. Falar ou não sobre um determinado tópico.

Tolerância. O quanto a metodologia é tolerante a falhas em alguns dos seus elementos de controle. Algumas metodologias, como a XP, é pouco tolerante a falhas, talvez por ser uma metodologia minimalista. Metodologias maiores tendem a ser mais tolerantes, com degradação progressiva do resultado.

Visibilidade. Quão fácil um observador pode dizer se uma metodologia está sendo seguida. Alguns padrões, como o ISO9001 foca em problemas de visibilidade. Como adicionar visibilidade em um projeto adiciona custos, as metodologias ágeis como um grupo têm pouca ênfase em visibilidade.

Escala. Quantos ítems são colocados juntos representando uma única entidade. Algumas metodologias criam entidades agrupadas para melhor visualização do todo, para combater o problema do limite do número de artefatos que a mente humano consegue lidar simultaneamente.

Estabilidade. Qual a capacidade de adaptação a mudanças ambientais e de escopo.

Com as informações deste capítulo, temos melhor embasamento para analisar as metodologias ágeis e as práticas do PMBOK, de forma mais clara e objetiva.

4 *Um vôo rápido sobre as diversas metodologias*

4.1 *Extreme Programming*

Extreme Programming é uma das – ou a mais – conhecida das metodologias ágeis, muito bem documentada (ver (BECK; ANDREAS, 2004), (WAKE, 2001) e (JEFFRIES; ANDERSON; HENDRICKSON, 2000)) e bastante controversa. Criada pelo engenheiro de software norte-americano Kent Beck enquanto era o líder do projeto C3 (acrônimo para *Chrysler Comprehensive Compensation*).

Em seu livro *Extreme Programming Explained* (BECK; ANDREAS, 2004), Kent Beck propõe o retorno de algumas práticas que foram abandonadas para o desenvolvimento de software, por serem consideradas ingênuas ou impraticáveis. Enfatiza o uso da comunicação oral ao invés do uso de documentos, simplicidade no desenho do software, realimentação constante do sistema com as lições aprendidas e problemas encontrados, tanto pelos participantes do projeto quanto pelo cliente. A metodologia XP envolve as seguintes práticas:

O Jogo do Planejamento. O planejamento do projeto se parece com um jogo de mesa, onde o cliente determina as prioridades e os analistas técnicos determinam as estimativas.

Pequenas iterações. Deve ser colocado um sistema em produção simples o mais rápido possível, e liberar novas versões em ciclos curtos.

Metáfora. O desenvolvimento deve ser guiado por uma história simples compartilhada por todos de como o sistema funciona.

Desenho simples. O sistema deve ser desenhado o mais simples possível. Qualquer complexidade extra deve ser removida no momento que for descoberta.

Testes. Os programadores devem escrever testes unitários continuamente, que precisam rodar

sem falhas para o desenvolvimento continuar. Clientes devem escrever testes demonstrando que uma característica foi terminada.

Refatoramento. Os programadores devem reestruturar o sistema continuamente, removendo código duplicado, melhorando a comunicação, a simplicidade ou adicionando flexibilidade.

Programação em pares. Todo código deve ser escrito por dois programadores em um único computador. Com isso é garantido a troca de experiências, a revisão constante do código e a transferência do modelo tácito do projeto.

Propriedade comum. Todos podem mudar qualquer parte do código a qualquer momento. Proporciona a melhoria contínua.

Integração contínua. O sistema deve ser integrado e testado muitas vezes num mesmo dia, a cada vez que uma tarefa é completada, utilizando-se os testes unitários escritos.

Semana de 40 horas. A equipe deve ter como meta trabalhar 40 horas por semana. Nunca deve realizar horas extras duas semanas consecutivas. Com isso garante-se a qualidade e o moral elevado da equipe.

Cliente presente. Um representante do cliente deve fazer parte da equipe, em tempo integral, para responder as questões que surgirem.

Padrões de codificação. Os programadores devem escrever o código de acordo com regras que enfatizem a comunicação no projeto.

Ao se analisar as técnicas do XP à luz dos conceitos de metodologias de software elaboradas por Cockburn, vê-se que a metodologia XP constrói um ambiente onde a comunicação pessoal e oral é enfatizada, e os tempos entre as questões e as respostas são curtos. O custo para a descoberta de uma informação é muito baixo. O cliente também tem uma resposta rápida quanto a implicação de suas decisões de implementação nos prazos e custos do projeto.

XP usa a excelente capacidade humana em se comunicar, através da programação em pares e retorno rápido do cliente, para compensar a tendência humana em cometer erros.

Por mais controverso que pareça, XP é uma metodologia muito disciplinada. É necessário alta aderência a padrões de código e de desenho, testes unitários que devem ter sucesso 100% do tempo, testes de aceitação, trabalho em pares, manter o código simples e refatoramento agressivo.

Dois pontos são considerados os mais controversos na metodologia XP: a falta de documentos e o uso de apenas pequenas equipes. No caso da falta de documentação, vê-se que XP enfatiza apenas o objetivo principal da equipe, a entrega do sistema funcionando, e não novos projetos futuros, como alterações e novas versões, onde a documentação seria necessária. Neste caso, o autor conta com o conhecimento tácito da equipe sendo mantido entre projetos, ou melhor dizendo, o capital humano da empresa. No caso de pequenas equipes, se levarmos em conta que uma equipe pequena com uma metodologia leve consegue produzir tanto quanto uma equipe grande com uma metodologia pesada, vemos que uma equipe pequena pode atacar problemas de tamanho razoável e ser muito eficiente para a maioria dos casos.

Usando os critérios estabelecidos no capítulo 3, vemos toda a estrutura de uma metodologia presente, com papéis, técnicas, atividades, processos, padrões, valores e outras. Podemos classificar a metodologia XP como sendo do tipo heurística e participativa, pois foca em processos para facilitar a capacidade de adaptação da equipe, e na participação constante do cliente no projeto. O seu escopo compreende as principais atividades necessárias para a produção de software e coordenação da equipe, deixando de lado preocupações como técnicas de seleção e capacitação de pessoal, gerencia financeira e controle de contratos de terceiros. Como tal foi utilizada em ambientes com requisitos instáveis e pequenas equipes, com grande êxito. A metodologia esforça-se em ser pequena, com baixa cerimônia, por isso mesmo uma metodologia leve. Por trabalhar somente com a comunicação oral e o conhecimento tácito da equipe, é sempre aconselhada para uso com equipes entre três a uma dúzia de pessoas (o suficiente para manter todos numa mesma sala). Fazendo uma reflexão, podemos questionar sua capacidade em atacar problemas com alta criticidade, envolvendo a perda de altas somas em dinheiro ou de vidas, sem o uso de processos mais formais. Uma outra crítica seria no sentido de ser pouco tolerante com relação a algumas fraquezas humanas, já que a metodologia se baseia na construção de testes automatizados e refatoração agressiva, atividades que sofrem resistência a serem adotadas pelo programador médio, um grande desafio para a adoção da metodologia em si.

4.2 *Crystal Clear*

Crystal Clear, mais do que uma metodologia, é o nome de uma família de metodologias elaboradas por Cockburn e descritas nos livros *Agile Software Development* (COCKBURN, 2001) e *Crystal Clear* (COCKBURN, 2005). O autor valoriza o conceito de que toda metodologia de desenvolvimento de software deve estabelecer um ambiente favorável para a produção do seu produto. São estabelecidos sete princípios a serem seguidos, sendo os três primeiros

obrigatórios, e os demais apenas auxiliares para manter o projeto neste ambiente seguro de desenvolvimento.

Os sete princípios são:

Entregas freqüentes. Segundo o autor, são tantas as vantagens de entregas freqüentes que é intrigante pensar que alguns projetos façam diferente. O intervalo entre entregas não deveria ultrapassar quatro meses, com os seguintes benefícios:

- O patrocinador tem retorno real e tangível sobre o progresso da equipe.
- O usuário tem a possibilidade de descobrir se o que ele pediu é o que ele realmente quer e ter isto refletido no projeto.
- Os desenvolvedores mantêm o foco, evitando momentos de congelamento por indecisão.
- A equipe pode depurar seu processo de desenvolvimento e ter sua moral elevada pelo sucesso da entrega.

Evolução reflexiva. A equipe deve se reunir periodicamente e rever seus processos e hábitos de trabalho. A periodicidade deve também ser discutida pelo grupo, uma vez por semana, por mês ou duas vezes por entrega, o que for melhor para permitir correções no rumo do projeto.

Comunicação osmótica. O autor se refere a comunicação osmótica como sendo a criação de um ambiente tal que o fluxo de informação ocorra entre os membros da equipe de forma quase acidental, como ao ouvir a discussão de terceiros e perceber uma informação importante para o projeto, ou uma dúvida da qual se sabe a resposta. Isto normalmente é conseguido ao se alocar a equipe num mesmo ambiente.

Segurança pessoal. Todas as pessoas do projeto devem se sentir seguras em dizer a verdade, sem temer represálias. Poderem dizer para o colega que o seu desenho do projeto é ruim, e para seu chefe que as estimativas que deu estão erradas em 50%. Com este tipo de segurança, segundo o autor, o projeto tem maiores chances de descobrir o que está causando danos e corrigí-lo o mais cedo possível.

Foco. É necessário ter-se bem claro qual é a tarefa prioritária e não ser interrompido. Leva-se horas para recuperar o fluxo de pensamentos ao ser interrompido por colegas, telefonemas ou reuniões. Todas as pessoas do projeto devem ter um período por dia sem interrupções e vários dias na semana sem terem de trocar de tarefas, para poderem ser realmente produtivos.

Acesso fácil aos usuário experientes. Acesso aos usuários proporciona a equipe:

- Um lugar para fazer entregas e testes frequentes.
- Resposta rápida quanto a qualidade do produto.
- Resposta rápida quanto a qualidade do desenho do produto.
- Requisitos sempre atualizados.

Infraestrutura com testes automatizados, gerenciamento de configuração e integração frequente.

A combinação destes três elementos trazem qualidade de vida aos desenvolvedores. Eles podem adicionar funcionalidades sem ter o medo de quebrar algo que funcionava, pois têm os testes na retaguarda, garantindo a consistência do sistema. Se algo der errado, podem voltar a versão anterior. E erros de integração são detectados no momento em que é mais fácil encontrar a área do código onde o erro pode existir.

Esta é uma visão bem superficial dos métodos e valores da metodologia *Crystal*, mas com estas informações já podemos fazer uma análise a luz dos nossos critérios de comparação. Todas as estruturas de uma metodologia estão presentes, mesmo que de forma bem informal. Podemos notar, da mesma forma que na metodologia XP, a ênfase da metodologia na adaptação e participação do cliente, ao invés de se basear em normas de trabalho, processos fixos e repetitivos e documentação. Podemos com isso classificá-la como sendo participativa e heurística. O seu escopo também, seguindo o modelo XP, é bem focado nas atividades principais do desenvolvimento de software, sem menção a atividades secundárias. Quanto a critérios de desenho da metodologia, vê-se que ela tenta ao máximo ser uma metodologia pequena, com baixa cerimônia, ou seja, uma metodologia leve. O autor descreve um arcabouço de criação de metodologias ágeis, ao invés de descrever uma única receita, a fim de proporcionar com isso ferramentas para que um praticante da área possa desenvolver a sua metodologia, com o peso adequado para o tamanho do seu projeto, o tamanho do seu problema, e sua criticidade. Neste sentido, sua obra estabelece uma escala compreendendo o tamanho da equipe e a criticidade do sistema, onde conforme estes parâmetros o praticante deve refletir sobre o uso de maior ou menor cerimônia, mas sempre procurando a quantia mínima e suficiente. Aliás, o autor sempre coloca como fator decisivo para o sucesso de um projeto a reflexão constante sobre métodos e processos de trabalho, sua melhoria contínua, e a compreensão e tolerância com a natureza e fraquezas humanas.

4.3 Scrum

Em 1986, Hirotaka Takeuchi e Ikujiro Nonaka descreveram um novo método para aumentar a velocidade e flexibilidade do desenvolvimento de novos produtos. Eles compararam este novo método holístico com fases que se sobrepõem e com um time multifuncional ao *rugby*, onde o time como um todo tenta avançar, passando a bola para trás e para frente. Em 1991, DeGrace e Stahl em *Wicked Problems, Righteous Solutions* referem-se a este método como *Scrum*, um termo do *rugby*. Em 1990, Ken Schwaber usou estes métodos em sua empresa, *Advanced Development Methods*, e ao mesmo tempo, Jeff Sutherland desenvolveu técnicas parecidas na empresa *Easel Corporation*. Em 1995 Sutherland e Schwaber apresentaram em conjunto um artigo na *OOPSLA'95* onde descreviam suas experiências, e no ano seguinte trabalharam juntos na produção do livro *Agile Software Development with Scrum* (ver (SCHWABER, 2004)).

A metodologia *Scrum* é um esqueleto que inclui uma série de práticas e papéis. O papel principal é o de *ScrumMaster* que é responsável por manter o processo e tem outras responsabilidades semelhantes a um gerente de projetos. O *Product Owner* é quem representa o patrocinador, e o *Team* inclui os desenvolvedores.

Durante o *sprint*, um período de 15-30 dias, é produzido um incremento de um produto de software possível de ser entregue ao usuário. O conjunto de funcionalidades que vão constar neste *sprint* são descritas no *product backlog*, que é uma lista de funcionalidades priorizadas do produto. Quais itens do *backlog* vão constar no *sprint* é decidido num *sprint planning meeting*. Durante esta reunião, o *Product Owner* informa a equipe que itens deseja que sejam inclusos no próximo *sprint*, a equipe então negocia prazos e é então congelado os requisitos durante o *sprint* (ver figura 4.1).

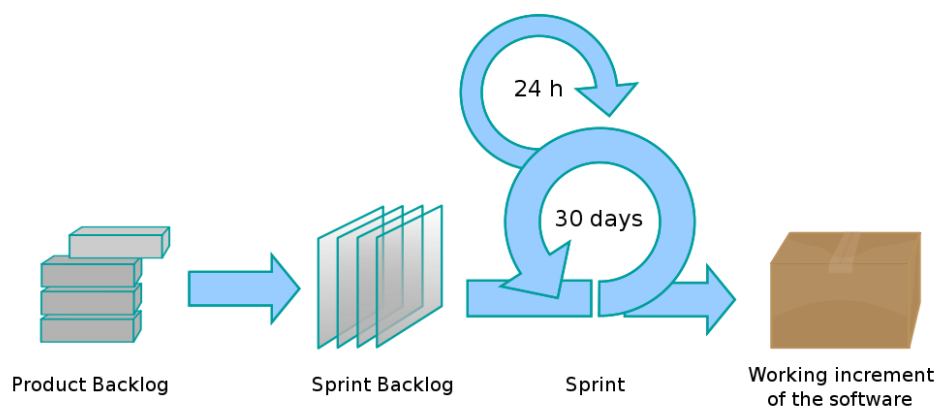


Figura 4.1: Processo da metodologia Scrum (SCRUM, 2008)

A metodologia *Scrum* procura criar equipes auto-adaptativas, encorajando equipe co-locadas e comunicação verbal. Um princípio considerado fundamental na metodologia *Scrum* é o reconhecimento de que o cliente pode mudar de idéia sobre o que precisa a qualquer momento, e que estas mudanças não são facilmente tratadas pelo método tradicional de planejamento. Então, tenta focar em promover a habilidade de responder rapidamente as mudanças.

As práticas gerais do *Scrum* são:

- O cliente deve fazer parte da equipe de desenvolvimento.
- Como muitas metodologias ágeis, *Scrum* também advoga entregas freqüentes do software produzido ao cliente.
- Planos de gerenciamento de riscos e mitigação devem ser desenvolvidos pela equipe. Deve ser feito em todos os estágios e ter seu comprometimento.
- Transparência no planejamento e no desenvolvimento, todos devem saber quem é responsável e quando é responsável.
- Reuniões freqüentes com o patrocinador do projeto, para monitorar o progresso.
- Nenhum problema deve ser varrido para debaixo do tapete. Ninguém deve ser penalizado por reconhecer um erro ou desvio do projeto.
- É desmotivado o uso de horas extras. Trabalhar mais horas não significa produzir mais.

O *Scrum* estabelece as atividades, papéis, processos e demais ítems da estrutura de sua metodologia. Como as metodologias anteriores, podemos agora notar como um padrão a tendência de todas as metodologias ágeis em serem do tipo participativas e heurísticas, o *Scrum* não sendo uma exceção. Também seu escopo é simplificado, mantendo-se no subconjunto das atividades principais necessárias ao desenvolvimento de software. A metodologia tem poucos elementos de controle e baixa cerimônia, e por consequência é uma metodologia leve. Não estabelece técnicas como o refatoramento e programação orientada a testes como obrigatórias, mas vê nestas uma boa prática, e aconselha fortemente seu uso. Exige disciplina e descreve uma receita mais clara de como deve ser o dia a dia do projeto, colocando como item principal da metodologia a comunicação eficiente.

4.4 *Feature Driven Development*

Feature Driven Development ou FDD é uma metodologia incremental e iterativa, desenvolvida por Jeff De Luca durante um projeto para um grande banco em Singapura, envolvendo 50 pessoas e com 15 meses de duração. Une uma série de boas práticas reconhecidas pela indústria para realizar o desenvolvimento de software de uma forma consistente, no prazo, e orientado às características que o cliente acredita mais importantes. A primeira menção da metodologia foi feita na obra *Java Modeling in Color with UML* (COAD; LEFEBVRE; LUCA, 1999) e posteriormente foi lançado um livro mais específico, *A Practical Guide to Feature-Driven Development* (PALMER; FELSING, 2002). Muita informação também pode ser coletada a partir do sítio do autor (LUCA, 2008).

Por causa das experiências e o contexto do autor, a metodologia é representada de forma iconográfica usando a linguagem de modelagem UML como visto na figura 4.2.

A metodologia FDD é composta por cinco atividades básicas:

Desenvolvimento do modelo geral. O projeto se inicia com o desenvolvimento de um modelo geral do sistema. Depois, cada área ou módulo do domínio do projeto é detalhado, passa por uma revisão e é integrado ao modelo geral.

Construção da lista de funcionalidades. O conhecimento adquirido durante o desenvolvimento do modelo inicial é usado para criar uma lista de funcionalidades. O modelo é decomposto em assuntos, cada um contendo uma atividade do negócio. A descrição dos passos de uma atividade do negócio forma uma lista de funcionalidades. Cada funcionalidade deve ser no formato ação - resultado - objeto, como “Calcular o total de vendas” ou “Validar a senha do usuário”. As funcionalidades não podem ter uma estimativa de tempo maior que duas semanas, neste caso deve ser decomposta em funcionalidades mais elementares.

Planejamento por funcionalidade. Depois da lista de funcionalidades completa, é desenvolvido o plano do projeto. Líderes de implementação recebem a posse e responsabilidade sobre conjuntos de funcionalidades que se tornarão classes dentro do paradigma de programação orientada a objetos.

Projeto por funcionalidade. Um pacote do projeto é desenvolvido para cada funcionalidade. Um líder do desenvolvimento seleciona um pequeno grupo de funcionalidades para serem desenvolvidas em duas semanas. Diagramas de seqüência do UML são desenvolvi-

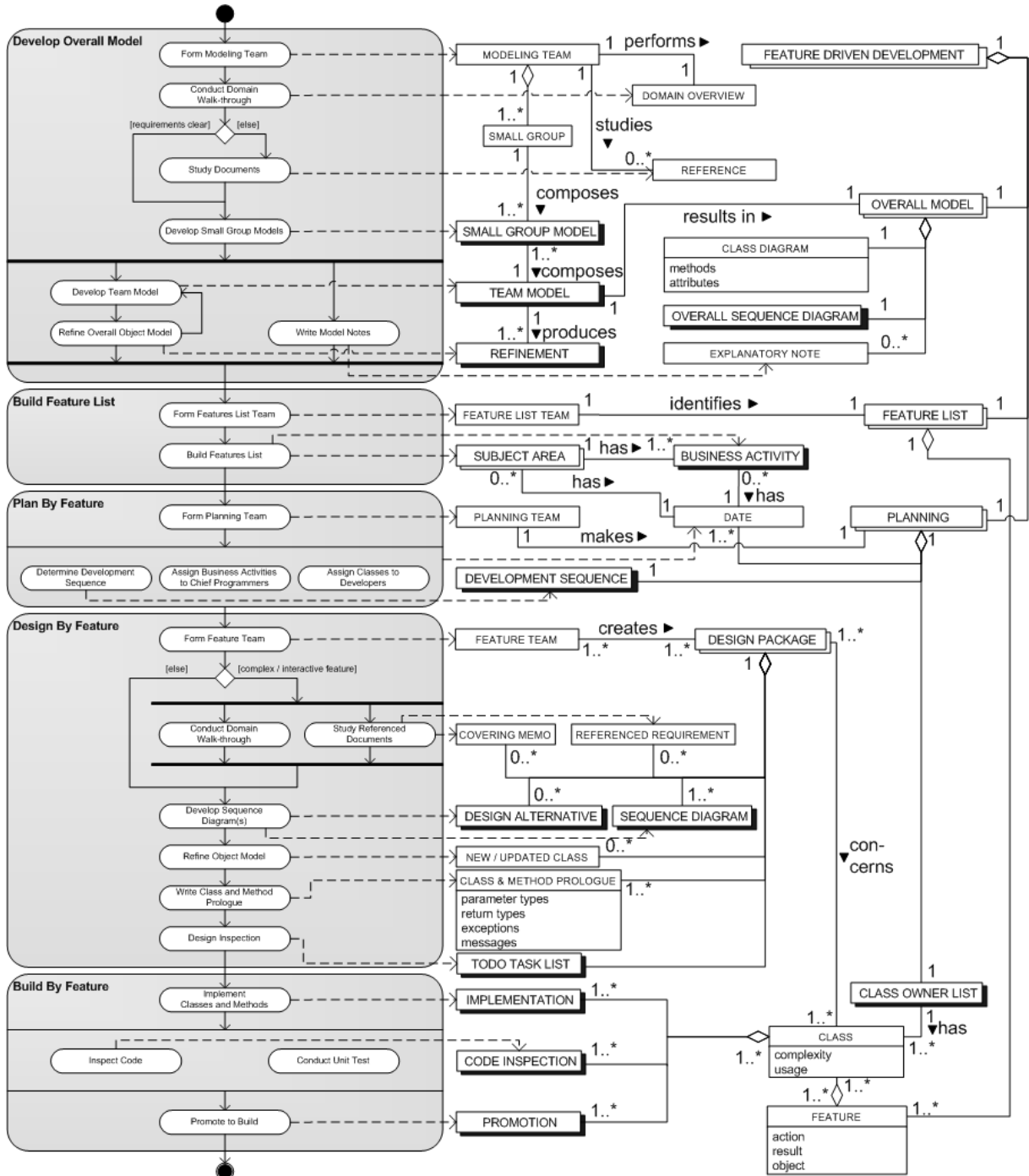


Figura 4.2: Diagrama de processos da metodologia FDD (FEATURE..., 2008)

dos para cada funcionalidade e é refinado o modelo. A assinatura e documentação dos métodos é escrita e uma revisão dos artefatos é feita.

Construção por funcionalidade. Depois do sucesso durante a inspeção dos artefatos da fase anterior, são construídas as funcionalidades especificadas, numa iteração com duração de duas semanas. Depois dos testes unitários e inspeção do código, as funcionalidades são promovidas para o sistema em uso.

Como pode ser visto, cada funcionalidade dentro do projeto é produzida numa iteração curta, sendo então fácil de gerenciar e estimar o tempo de conclusão. Cada funcionalidade passa por estas cinco atividades de forma seqüencial, e marcos são definidos por atividade para seu acompanhamento. Na tabela 4.1 são representadas as atividades e o percentual que representa, por padrão, dentro do total das atividades:

Modelo Geral	Lista	Plano	Projeto	Construção	Promoção
1%	40%	3%	45%	10%	1%

Tabela 4.1: Marcos na metodologia FDD

A metodologia FDD é em muitos aspectos mais cerimoniosa e mais pesada que suas irmãs ágeis, mas ao revermos o contexto onde foi criada, uma equipe de 50 pessoas num projeto de um grande banco, percebemos que ela é ágil sim, mas adaptada as condições de uma equipe maior e numa área mais crítica, onde a perda de dinheiro seria muito relevante. O autor coloca como as principais características incorporadas na metodologia as seguintes:

Modelagem dos objetos do domínio. A modelagem dos objetos do domínio consiste em explorar e explicar o modelo do problema a ser resolvido, de forma a se construir um arcabouço para a construção do sistema.

Desenvolvimento por funcionalidade. Cada funcionalidade é decomposta até que seu tempo de construção seja menor que duas semanas, desta forma é mais fácil construir e controlar cada funcionalidade.

Propriedade individual do código. Cada parte do código tem um único responsável pela sua consistência, desenho e desempenho.

Equipes funcionais. Uma equipe funcional é responsável por uma atividade, de forma que cada decisão de projeto passe pelo aval de muitas pessoas, e várias alternativas sejam consideradas.

Inspeções. São realizadas inspeções após a produção de cada artefato, para se assegurar a qualidade do produto.

Gerenciamento de configuração. Mantêm o histórico de mudanças, o código alterado e permite a recuperação em caso de falhas.

Integrações regulares. Integrações regulares permitem assegurar que o sistema está funcional, e permite capturar erros de integração o mais cedo possível.

Visibilidade do progresso e resultados. Com marcos bem definidos por todo o projeto, o gerente e o patrocinador podem saber a todo momento o progresso e quando intervir para colocar o projeto novamente no rumo.

Sem dúvida, o FDD destoa das metodologias ágeis vistas até o momento. Mais cerimoniosa e com muito mais elementos de controle, é consideravelmente mais pesada, com foco maior na produção de documentos que orientem a construção do sistema de software. Mas mantêm o padrão de ser heurística e participativa. Apesar de destoar, está na classe das metodologias ágeis pela preocupação constante no envolvimento do usuário, iterações curtas e a comunicação, que promove a criação do modelo comum do sistema. Suas diferenças em relação às demais vêm mostrar que mesmo metodologias num ambiente de alta criticidade e com grandes equipes podem ser ágeis, de acordo com a definição dos seus autores, quebrando o mito de que metodologias ágeis estão fadadas a projetos pequenos e triviais. Isto vai ao encontro do trabalho de Cockburn em seu conjunto de metodologias, de tal forma flexíveis que podem ser adaptadas a criticidade e tamanho do projeto do praticante. Luca, no entanto, prefere fornecer ao usuário de sua metodologia não um arcabouço, mas um pacote fechado de práticas com eficácia comprovada.

4.5 O PMBOK

Após a visão geral de diversas metodologias ágeis de desenvolvimento de software, é interessante confrontarmos estas metodologias com as práticas advogadas pelo PMBOK. Apesar do argumento sempre presente de que o PMBOK não é uma metodologia, mas um conjunto de boas práticas que podem ou não serem seguidas, e que o ciclo de vida, técnicas e ferramentas devem ser adequados pela equipe do projeto ao seu projeto em questão, o próprio aconselha fortemente o uso de todos os processos descritos, o que pode confundir o iniciante na área de gerenciamento de projetos, reduzindo o número de metodologias àquelas que sejam coerentes com as recomendações. O PMBOK não é uma metodologia, mas acaba ditando, principalmente

ao ser uma norma do PMI, quais elementos devem estar contidos em uma metodologia válida. Deste modo, a comparação será feita através do ciclo de vida e processos recomendados, em comparação com o ciclo de vida e processos de uma metodologia ágil típica.

O PMBOK descreve 44 processos do gerenciamento de projetos, divididos em cinco grupos e nove áreas do conhecimento. Os cinco grupos são executados na mesma seqüência em todos os projetos, independente da área ou indústria de aplicação (ver figura 4.3):

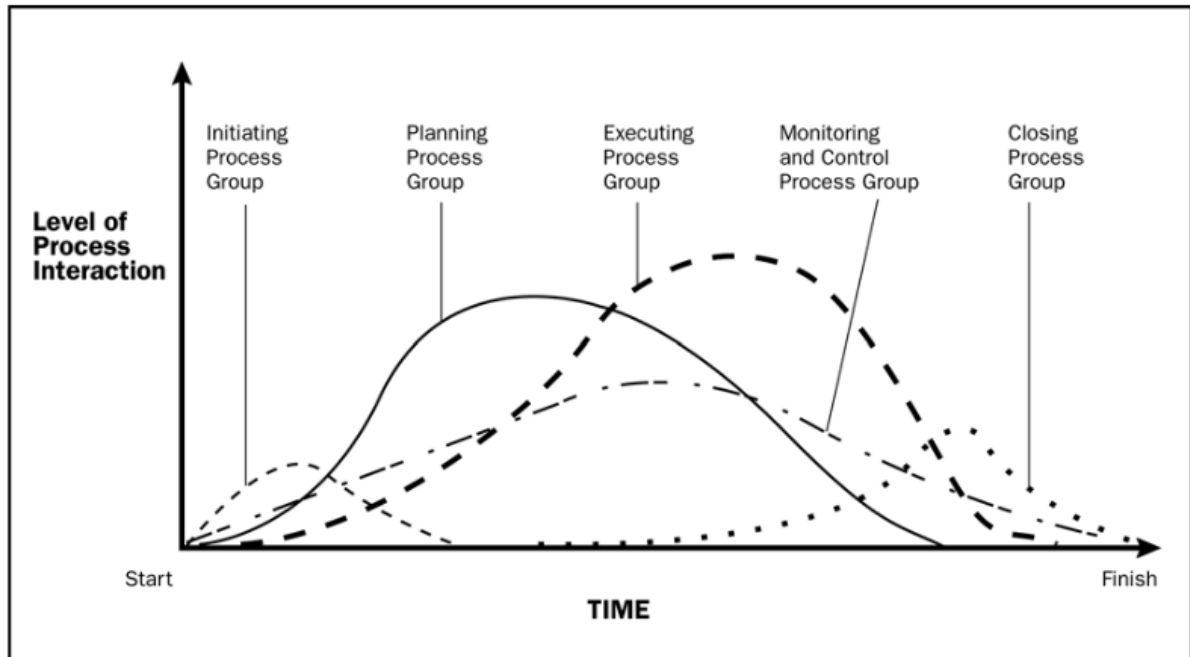


Figura 4.3: Iteração entre os grupos de processos do PMBOK (PMI, 2004)

Grupo de processos de iniciação. Define e autoriza o projeto ou fases do projeto.

Grupo de processos de planejamento. Define e refina objetivos e curso de ação para atender os requisitos do projeto.

Grupo de processos de execução. Integra pessoas e recursos para efetivar o plano do projeto.

Grupo de processos de monitoramento e controle. Processos que regularmente medem e monitoram o progresso e identificam variações em relação ao plano, de forma que ações corretivas podem ser tomadas.

Grupo de processos de encerramento. Formaliza a aceitação do produto, serviço ou resultado e faz com que o projeto chegue ao seu fim.

Já neste momento fica evidente uma característica forte do PMBOK, que é seu foco em processos. O que nos remete novamente aos trabalhos de (MAIER; RECHTIN, 2000), mostrando que o PMBOK tende a ditar metodologias do tipo racionais, ou até mesmo normativas já que faz parte das normas do PMI. O que revela uma intenção a grande maturidade de suas recomendações.

Todos os processos descritos no PMBOK têm entradas e saídas bem definidas, normalmente descrevendo documentos a serem criados para uso nos processos seguintes. Seu escopo é muito mais abrangente que todas as metodologias ágeis descritas anteriormente, passando por processos de controle do tempo, custos, qualidade, recursos humanos, comunicação, riscos, contratos e outros.

Outra tendência que pode-se notar é que o trabalho de um projeto de acordo com o PMBOK deve seguir a risca o plano estabelecido. Muitos autores separam o mundo das metodologias de desenvolvimento entre as adaptativas e as orientadas a planos, sendo que uma metodologia seguindo o PMBOK estaria no segundo grupo. Mecanismos para adaptar o plano de acordo com mudanças encontradas durante a execução existem, mas estes estão mais para tratar a exceção do que refletir a regra. O gerente do projeto deve concentrar suas energias em elaborar um bom plano, que contenha todos os possíveis eventos e planos de contingência, e guiar o projeto de acordo com ele uma vez que posto em execução.

O PMBOK sugere uma metodologia de trabalho com muitos pontos de controle e cerimoniais, muito provavelmente uma metodologia bem pesada. Com estas características, está muito melhor colocado para gerir projetos com grandes equipes e de grande criticidade. Provavelmente, uma metodologia atuando de acordo com os princípios descritos no PMBOK seria também pouco tolerante a falta de disciplina e a variações nos requisitos, já que isto provocaria uma mudança em cascata numa série de documentos do projeto.

5 *Conclusões*

Na curta revisão histórica realizada neste trabalho, é possível ver que, apesar de possuírem intersecções, a história do gerenciamento de projetos no seu âmbito mais geral e clássico e a história do gerenciamento de software possuem naturezas distintas.

O gerenciamento de projetos clássico sempre teve como produto um item tangível, como estradas, mísseis, usinas hidrelétricas, edifícios, entre outros. Estes projetos em sua maioria permitem a conjectura de terem em comum a linearidade do crescimento de sua complexidade dado o seu tamanho. Mesmo antes das técnicas modernas de gerenciamento de projetos, vários empreendimentos de tamanho considerável foram realizados com grande sucesso.

O software parece ser o primeiro produto intangível com tamanha disseminação e valor econômico da história da humanidade. Quase todas as atividades humanas hoje têm o auxílio de software para serem executadas, e uma grande empresa ou um programador solitário parecem ter a mesma chance de sucesso com seus produtos. Mas pela característica de ser produto da criatividade e gênio humanos, sofre problemas de escalabilidade e sua complexidade tende a crescer de forma exponencial dado o tamanho do problema atacado e o respectivo tamanho da equipe do projeto.

O conhecimento acumulado da coletividade parece seguir um ciclo de formulação de hipóteses seguida de sua comprovação ou refutação. Assim se deu com o conhecimento em gerenciamento de projetos, que seguiu este ciclo e atingiu sua maturidade quase no mesmo instante histórico em que surgiu o software, uma variável nova até então desconhecida. Esta nova variável parece contradizer um modelo já tido como maduro, e esta contradição traz consigo além da defesa quase apaixonada pelos seus adeptos, problemas econômicos para aqueles que investiram neste modelo.

A maturidade de todo processo tem dois lados. Por um lado, a maturidade traz consistência, a capacidade de otimização e processos bem definidos. Por outro lado é o fim da busca por melhorias, é a supremacia da obediência aos processos sobre a inovação. Cada entidade certificadora torna-se então guardião dos processos definidos, retardando sua evolução. Ela enfrenta

o dilema de que as práticas de hoje refletem o passado, mas os padrões devem se adaptar às mudanças, ou se tornam barreiras ao progresso. No entanto, enormes somas de recursos são despendidas por pessoas e organizações para se adequar aos padrões vigentes, e sua mudança constante seria vista como um erro de julgamento da entidade certificadora, o que prejudicaria sua credibilidade, o seu maior produto.

Neste cenário tem-se o conflito entre o que está defendido no PMBOK pelo PMI como boas práticas e as metodologias ágeis. Pelo que foi visto neste trabalho, o software é um novo elemento que possui muitas características únicas, e por isso não deveria ser colocado no mesmo conjunto de produtos clássicos tratados pelo gerenciamento de projetos tradicional. No entanto, não é verdade que nada do conhecimento acumulado da gerência de projetos clássica não se aplica ao gerenciamento de projetos de software, mas é necessário a adição de novos conceitos e técnicas e adaptação dos antigos no conjunto de conhecimentos de forma mais rápida e dinâmica, de forma a acompanhar o passo de evolução da área e do mercado como um todo.

Neste contexto, é importante refletir o valor de entidades que promovem a repetição, processos e padrões num mundo de mudanças cada vez mais rápidas, onde a escola de gerenciamento atual promove *Learning Organizations*¹ (CHAWLA; RENESCH, 1995), pois o aprendizado é a única vantagem competitiva sustentável que qualquer empresa pode ter em qualquer tempo. Neste sentido, as metodologias ágeis estão mais em acordo com esta nova visão, com sua ênfase no aprendizado e adaptação.

Mas vale notar que o processo evolutivo das metodologias ágeis deu seus primeiros passos, mas ainda precisa responder muitas questões, a saber:

- Como lidar com o aumento de escala em projetos ágeis?
- Como gerenciar equipes ágeis geograficamente dispersas num mundo cada vez mais globalizado?
- Como planejar os recursos necessários para um projeto ágil, como estabelecer estimativas de custos e orçamentos?
- Como fazer com que participantes de nível médio sejam produtivos num ambiente ágil?

Finalizando, vemos que a contribuição para a gerência de projetos de software dada hoje pelo PMBOK parece ser mais danosa que benéfica, que suas recomendações só seriam adequadas para projetos de software envolvendo uma grande equipe, para atacar um grande problema

¹Tradução: Organizações que aprendem

numa área de alta criticidade. Como a maioria dos projetos de software não possui estas características acabam sofrendo com a sua influência por ser aplicado na sua totalidade sem a devida reflexão pelos membros da organização. As metodologias ágeis, por outro lado, começam menos ambiciosas quanto ao seu escopo e seu alcance, até mesmo por serem uma resposta contra o modelo anterior, e têm ainda o que evoluir, mas por serem livres de vícios, por terem a adaptação como bandeira e por serem específicas, são mais promissoras num espaço de tempo mais curto como resposta ao problema de como desenvolver software que devolva mais valor ao usuário, de forma mais rápida e mais eficiente.

Referências Bibliográficas

ABNTEX. 2008. Disponível em: <<http://abntex.codigolivre.org.br>>. Acesso em: 23 out. 2008.

BECK, K.; ANDREAS, C. *Extreme Programming Explained: Embrace change*. 2. ed. [S.l.]: Addison-Wesley Professional, 2004.

BOEHM, B. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 11, n. 4, p. 14–24, 1986. ISSN 0163-5948.

BOEHM, B. A view of 20th and 21st century software engineering. In: *ICSE '06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006. p. 12–29. ISBN 1-59593-375-1.

BöHM, C.; JACOPINI, G. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, ACM, New York, NY, USA, v. 9, n. 5, p. 366–371, 1966. ISSN 0001-0782.

BROOKS, J. F. P. *The Mythical Man-Month: Essays on software engineering*. Anniversary. [S.l.]: Addison-Wesley Longman, Inc., 1995. ISBN 0-201-83595-9.

CASTLES & Cathedrals. 2008. Disponível em: <<http://lincolnmotorhomehire.co.uk/castles.htm>>. Acesso em: 20 nov. 2008.

CHAWLA, S.; RENESCH, J. *Learning Organizations*. [S.l.]: Productivity Press, 1995.

COAD, P.; LEFEBVRE, E.; LUCA, J. D. *Java Modeling In Color With UML: Enterprise components and process*. 1. ed. [S.l.]: Prentice Hall PTR, 1999.

COCKBURN, A. *Agile Software Development*. 1. ed. New York: Addison-Wesley Publishing Company, 2001.

COCKBURN, A. *Crystal Clear: A human-powered methodology for small teams*. 1. ed. New York: Addison-Wesley Publishing Company, 2005.

DECEMBER 17, 1957 - first launch of the Atlas rocket by U.S. 2008. Disponível em: <<http://todayinspacehistory.wordpress.com/2007/12/17/december-17-1957-first-launch-of-the-atlas-rocket-by-us/>>. Acesso em: 20 nov. 2008.

DIJKSTRA, E. W. Letters to the editor: go to statement considered harmful. *Commun. ACM*, ACM, New York, NY, USA, v. 11, n. 3, p. 147–148, 1968. ISSN 0001-0782.

EGYPTIAN pyramids. 2008. Disponível em: <http://en.wikipedia.org/wiki/Egyptian_pyramids>. Acesso em: 20 nov. 2008.

- FEATURE Driven Development. 2008. Disponível em: <http://en.wikipedia.org/wiki/Feature_Driven_Development>. Acesso em: 21 nov. 2008.
- FRAME, J. D. Iterative and agile project management: Rethinking pmbok, cmm and iso 9000. 2008. Disponível em: <<http://www.nysforum.org/events/agileprojectmanagement-1-16-08>>. Acesso em: 03 nov. 2008.
- JAIN, A.; BOEHM, B. Developing a theory of value-based software engineering. In: *EDSER '05: Proceedings of the seventh international workshop on Economics-driven software engineering research*. New York, NY, USA: ACM, 2005. p. 1–5. ISBN 1-59593-118-X.
- JEFFRIES, R.; ANDERSON, A.; HENDRICKSON, C. *Extreme Programming Installed*. 1. ed. [S.l.]: Addison-Wesley Professional, 2000.
- LUCA, J. D. *I.T. Solutions That Make A Difference*. 2008. Disponível em: <<http://www.nebulon.com/articles/index.html>>. Acesso em: 21 nov. 2008.
- MAIER, M.; RECHTIN, E. *The Art of Systems Architecting*. 2nd. ed. Boca Raton: CRC Press, 2000.
- MANIFESTO for Agile Software Development. 2001. Disponível em: <<http://agilemanifesto.org>>. Acesso em: 05 nov. 2008.
- MARASCO, J. Software development productivity and project success rates: Are we attacking the right problem? 2006. Disponível em: <<http://www.ibm.com/developerworks/rational/library/feb06/marasco/index.html>>. Acesso em: 02 nov. 2008.
- MORRIS, P. W. G. *The Management of Projects*. 1. ed. [S.l.]: Amer Society of Civil Engineers, 1997. 18-77 p. ISBN 978-0727725936.
- NEIGHBORHOOD Recovery Act. 2008. Disponível em: <<http://virtualology.com/MANHATTENPROJECT.COM/>>. Acesso em: 20 nov. 2008.
- NORMANDY Landings. 2008. Disponível em: <<http://en.wikipedia.org/wiki/D-Day>>. Acesso em: 20 nov. 2008.
- PALMER, S. R.; FELSING, J. M. *A Pratical Guide to Feature-Driven Development*. 1. ed. [S.l.]: Prentice Hall PTR, 2002.
- PROJECT MANAGEMENT INSTITUTE (USA). *A guide to the project management body of knowledge: Pmbok guide*. 3. ed. Pennsylvania: Project Management Institute, Inc., 2004. ISBN 1-930699-45-X.
- RAILROADS in Utah. 2008. Disponível em: <<http://www.media.utah.edu/UHE/r/RAILROAD.html>>. Acesso em: 20 nov. 2008.
- RAYMOND, E. S. *The Cathedral and the Bazaar*. 1999. Disponível em: <<http://www.catb.org/esr/writings/cathedral-bazaar/>>. Acesso em: 14 dez. 2008.
- REVIEW: The Mythical Man Month: Essays on Software Engineering - Anniversary Edition. 2005. Disponível em: <<http://slashdot.org/books/980805/1148235.shtml>>. Acesso em: 08 dez. 2008.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987. p. 328–338. ISBN 0-89791-216-0.

SCHWABER, K. *Agile Project Management with Scrum*. 1. ed. [S.l.]: Microsoft Press, 2004.

SCRUM. 2008. Disponível em: <[http://en.wikipedia.org/wiki/Scrum_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development))>. Acesso em: 22 nov. 2008.

THE STANDISH GROUP. Chaos. 1995. Disponível em: <<http://net.educause.edu/ir/library/pdf/NCP08083B.pdf>>. Acesso em: 01 nov. 2008.

WAKE, W. C. *Extreme Programming Explored*. 1. ed. [S.l.]: Addison-Wesley Professional, 2001.