2
Revisão Bibliográfica

2.1.
Considerações Iniciais

Neste capítulo é feita a revisão da literatura sobre o tema da pesquisa. É abordada uma análise preliminar das empresas dedicadas a desenvolver projetos de *software*, considerando estudos feitos pelo PMI em 2009 e pela *Standish Group* que avalia os principais fatores de sucesso e fracasso em projetos de *software*.

O capítulo também apresenta como marco teórico o conceito geral de gerenciamento de projetos, metodologias tradicionais e ágeis de desenvolvimento de *software*: *Scrum*, *Extremme Programming, Feature Drive Development*, Produção Enxuta de *Software* e as características principais dos processos de desenvolvimento de *software* com a metodologia CMMI (*Capability Maturity Model Integration*).

2.2.
Contextualização do Problema

O setor dedicado a desenvolver projetos de *software* é denominado pelo PMI como setor de tecnologia de informação, setor em crescimento para a economia do Brasil segundo o Estudo de Benchmarking em Gerenciamento de Projetos do Brasil, *Chapters* Brasileiros, feito pelo PMI em 2009 com a participação de 300 empresas brasileiras. Este estudo de Benchmarking apresentou resultados interessantes e tem por objetivo apresentar um perfil de importantes setores da economia, no que diz respeito ao Gerenciamento de Projetos, oferecendo:

- ✓ Estatísticas sobre práticas de Gerenciamento de Projetos utilizadas;
- ✓ Nível de adequação dos setores da economia às melhores práticas;
- ✓ Ferramentas e técnicas mais utilizadas;
- ✓ Perspectivas e tendências em Gerenciamento de Projetos.

Neste estudo, o PMI, conclui que cerca de 21% dos setores econômicos avaliados são dedicados a desenvolver atividades de tecnologia de informação. (Ver Tabela 2.1).

Tabela 2.1: Distribuição por Setor da Economia

Setor da Economia	**%**
Tecnologia da Informação	21
Consultoria	13
Serviços	10
Industria	9
Engenharia e EPC	7
Governo - Administração Indireta	5
Petróleo- Petroquímica e Gás	4
Governo - Administração Direta	4
Serviços Financeiros	3
Telecomunicações	3
Automobilística	3
Outros	18

Fonte: PMI, 2009

O PMI analisa o setor da tecnologia da informação e na Tabela 2.2 resume os problemas que ocorrem com mais frequência no gerenciamento dos projetos deste setor, destacando: “Problemas de Comunicação, Não Cumprimento do Orçamento, Mudanças de Escopo, Escopo Não Definido Adequadamente Entre Outros”.

Tabela 2.2: Problemas que ocorrem com mais frequência nos projetos na organização

Problemas em Projetos de Tecnologia de Informação	%
Não cumprimento de prazos	9
Não cumprimento de orçamento	5
Problemas de comunicação	11
Escopo não definido adequadamente	8
Mudanças de escopo constantes	9
Falta de apoio da administração	2
Riscos não avaliados corretamente	8
Estimativas incorretas ou sem fundamento	5
Recursos Humanos insuficientes	8
Mudanças de prioridade constantes	7
Outros	28

Fonte: PMI, 2009

Na década de 1970 foi proposto um modelo para desenvolver e gerenciar *software*, conhecido como "Modelo de Desenvolvimento em Cascata", que é estudado no ponto 2.4 desta pesquisa. Neste modelo, muitos problemas em projetos de *software* foram identificados ao longo dos últimos anos, e diversos estudos demostraram que os principais motivos nas falhas nos projetos de *software* são os métodos de gerenciamento. The *Standish Group* (1994, 2006) fez uma pesquisa, e avaliou os principais fatores de sucesso e fracasso em projetos de desenvolvimento de *software* resumidos nas Tabelas 2.3, e 2.4.

De acordo com a análise da *Standish Group* pode-se notar que o envolvimento do usuário, a clara definição dos requisitos, o suporte administrativo e o planejamento apropriado são fatores de grande importância para o sucesso dos projetos de *software*, é assim que surgem propostas de novos modelos de gerenciamento de desenvolvimento de *software* que serão estudados ao longo desta pesquisa.

Tabela 2.3: Fatores de Sucesso em Projetos de *Software*

Fatores de Sucesso em Projetos de *Software*	**%**
Envolvimento do usuário	15,9
Suporte da alta administração	13,9
Clara definição dos requisitos	13
Planejamento Apropriado	9,6
Expectativas Realísticas	8,2
Pontos de verificação dos projetos menores	7,7
Competência da equipe	7,2
Propriedade do projeto	5,3
Clara visão e objetivos	2,9
Trabalho intenso	2,4
Outros	13,9

Fonte: "The Chaos Report(2006)"

Tabela 2.4: Fatores de Fracasso em Projetos *Software*

Fatores de Fracasso em Projetos de *Software*	**%**
Requisitos e especificações incompletos	13,1
Falta de participação dos usuários	12,4
Falta de recursos	10,6
Expectativas irreais	9,9
Ausência De suporte de alta administração	9,3
Volatilidade de requisitos e especificações	8,7
Falta de planejamento	8,1
Obsolescência do Projeto	7,5
Ausência de gerencia de Tecnologia da Informação	6,2
Problemas com a tecnologia empregada	4,3
Outros	9,9

Fonte: "The Chaos Report (2006)"

Segundo Fiorini (1988) o processo de desenvolvimento de *software* é um conjunto de atividades, métodos, práticas e transformações que são usadas para desenvolver, manter, e evoluir *software* e seus artefatos associados.

Humphrey (1995) assinala que um processo de desenvolvimento de *software* é uma sequência de passos necessários para desenvolver e manter *software*.

Santos Soares (2004) assinala que um processo ou metodologia de desenvolvimento de *software* é um conjunto de atividades e resultados

associados que auxiliam na produção de *software,* que tem como resultado do processo um produto que reflete a forma como o processo foi conduzido. Já Sommerville (2001, *apud* Daflon ,2004) define um processo de desenvolvimento de *software* como um conjunto de atividades e resultados associados que levam à produção de um produto de *software*.

Sommerville (2003, *apud* Santos Soares, 2004) apresenta as etapas comuns a todos os processos de desenvolvimento de *software,* que são explicados a seguir:

Especificação de *Software*: É a etapa que define as funcionalidades (requisitos) e restrições do produto de *software*. Nesta etapa o desenvolvedor geralmente conversa com o cliente para definir as características do produto de *software* a desenvolver.

Projeto e Implementação de *Software*: O *software* é produzido de acordo com as especificações da etapa anterior. São propostos modelos através de diagramas e estes modelos são implementados em alguma linguagem de programação.

Validação de *Software*: O *software* é validado para garantir que todas as funcionalidades especificadas sejam implementadas no projeto.

Evolução de *Software*: Refere-se a que a evolução do *software* é importante continuar sendo útil ao cliente.

2.3. Metodologias Tradicionais de Desenvolvimento de Software

As metodologias tradicionais são conhecidas também como metodologias pesadas ou orientadas à documentação. Antigamente, com o uso destas metodologias, o custo de corrigir e fazer alterações era muito alto, já que o acesso aos computadores era limitado, além do fato que não existiam ferramentas modernas, como depuradores e analisadores de código que pudessem ajudar ao desenvolvimento do *software*. Por isso, o *software* era todo planejado e documentado antes de ser implementado. (SANTOS SOARES ,2004).

Teles (2004) usa o termo de metodologia tradicional, ou desenvolvimento tradicional, para se referir aos projetos de *software* que se baseiam no desenvolvimento em cascata.

O modelo em cascata foi proposto na década de 1970 por Winston Royce, num artigo chamado "*Managing the development of large software systems*"

(ROYCE, 1970), e surgiu pela necessidade de controlar grandes projetos de *software*. O modelo em cascata é estruturado, seqüenciado e direcionado á documentação em todas as atividades realizadas ao longo do desenvolvimento.

A Figura 2.1 ilustra uma representação do modelo cascata.

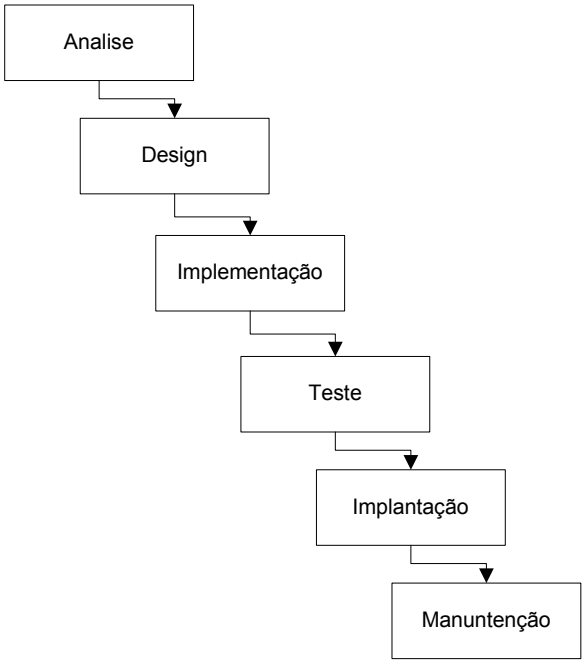

Figura 2.1 – Desenvolvimento tradicional (em cascata)
FONTE: Teles (2004), pág. 31

Royce (1970) e Teles (2004) descrevem as etapas do modelo em cascata:

1. **Análise:** Nesta etapa a equipe faz o levantamento dos requisitos do projeto e busca compreendê-los detalhadamente.
2. **Design:** A equipe projeta a arquitetura do sistema considerando a análise.
3. **Implementação:** A equipe implementa as diferentes partes de *software,* considerando a análise e o *design*.
4. **Teste:** A equipe testa o *software* para verificar se o sistema atende às necessidades especificadas pelo usuário.
5. **Implantação:** O sistema é colocado em produção e os usuários finais passam a utilizá-lo.
6. **Manutenção:** O *software* poderá sofrer diversas alterações, como correções, inclusão de novas funcionalidades ate o final da sua vida.

O desenvolvimento tradicional é sequencial, a equipe deve construir o sistema avançando gradualmente nas diferentes fases de desenvolvimento.

O desenvolvimento em cascata é a base para inúmeros processos de desenvolvimento de *software*, embora seja reconhecidamente ineficaz, mas ainda é o processo mais utilizado para desenvolver sistemas (TELES, 2004).

Santos Soares (2004) menciona que nos inícios da década dos 90 já muitos pesquisadores começaram a advertir que nem sempre o modelo tradicional era o melhor caminho para desenvolver projetos de *software*, já que encontraram várias fraquezas, mas mesmo assim foi o modelo que predominou nessa época, e é na atualidade o mais usado.

2.4. Metodologias Ágeis de Desenvolvimento de *Software*

As metodologias ágeis surgem com o objetivo de melhorar e criar uma alternativa ao processo de desenvolvimento de *software* tradicional (processo em cascata), e podem ser considerados como uma coleção de técnicas de melhora iterativa que compartilham princípios e valores básicos (COHEN *et al*, 2003, p.2) .

A Agile Aliance (2009) descreve as metodologias ágeis como uma forte colaboração entre a equipe de programadores e os clientes; uma comunicação face-a-face (considerada mais eficiente do que documentação escrita); entrega de *software* frequente que agregue valor; equipes com um bom relacionamento e auto-organizáveis; e formas de manter cuidadosamente o código e a equipe, de modo tal que modificações drásticas nos requisitos não sejam consideradas uma crise para o projeto em desenvolvimento.

Para Santos Soares (2004), as metodologias ágeis produzem um melhor resultado para sistemas com requisitos variáveis, já que estas metodologias propõem desenvolvimento de *software* iterativo, disciplinado e criativo, com entregas rápidas.

Martins (2007) assinala que num processo ágil é raramente possível fazer uma especificação inicial que seja detalhada e permaneça imutável, já que afirma "**No inicio é quase impossível estimar detalhadamente todo o projeto**".

Isotton Neto (2008) menciona que as metodologias ágeis trabalham com mais ênfase na implementação do que na geração de documentação. Ele faz uma comparação com as metodologias tradicionais e conclui que estas

metodologias têm como característica principal o fato de serem mais adaptativas do que preditivas. Por isso, as metodologias ágeis têm a facilidade de se adaptar a novos requisitos do projeto, ao contrario das metodologias tradicionais que procuram analisar tudo antes de desenvolver o *software*.

Na atualidade as metodologias ágeis estão tentando verificar vantagens reais frente ao uso de metodologias tradicionais. Segundo o Manifesto Ágil (2009) as metodologias ágeis tem suas bases em 13 princípios:

1. A maior prioridade é satisfazer ao cliente, com entrega adiantada e contínua de *software* de valor.
2. Aceitam-se mudanças de requisitos, mesmo no fim do desenvolvimento.
3. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.
4. Entrega de *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
5. Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
6. Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
7. O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
8. O *software* funcional é a medida primária de progresso.
9. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter indefinidamente, passos constantes.
10. A contínua atenção à excelência técnica e o bom design aumentam a agilidade.
11. A simplicidade é a arte de maximizar a quantidade de trabalho que não precisou ser feito.
12. As melhores arquiteturas, requisitos e design emergem de times auto-organizáveis.
13. Em intervalos regulares, o time reflete sobre ficar mais efetivo, então, se ajustam e aperfeiçoam os comportamentos.

O Manifesto Ágil (2009) destaca como as principais metodologias ágeis de gerenciamento de projetos de software a *Extreme Programming*, *Scrum* e *Feature Drive Development.*

2.4.1.
Extreme Programming

O principal fundador da metodologia *Extreme Programming* chama-se Kent Beck. Esta metodologia é conhecida como XP e começou a ser concebida na metade da década de 1980, quando *Beck e Ward Cunninghan* trabalharam juntos num grupo de pesquisa na *Tektronix*. Anos depois Beck trabalhou como consultor, e foi definindo as praticas que hoje são associadas ao XP. (Fowler, 2005; Goldman *et al*. , 2004)

Segundo Beck (1999), a XP é uma metodologia que procura resolver as limitações do processo de desenvolvimento de *software*, mas não aborda o processo de gerenciamento do projeto, análise financeira, marketing ou vendas. Em relação aos custos, o custo de mudança não aumenta exponencialmente com o avanço do projeto, como o ilustra a Figura 2,2.

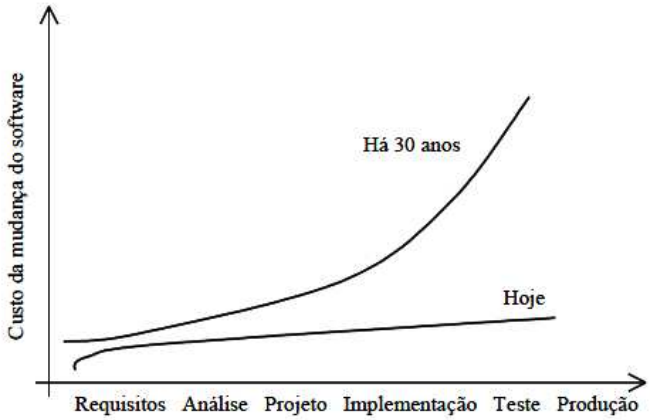

Figura 2.2 – Custo de mudança do *Software*

FONTE: Bôas *et al* (2005) pag 66

De acordo com Astels (2002), XP é um processo de desenvolvimento proposto para projetos que possuam uma equipe de dois a dez desenvolvedores, e trabalhem com funcionalidades não muito bem definidas ou que mudam rapidamente em seu decorrer. A Figura 2.3 descreve as principais praticas XP de desenvolvimento de *software*.

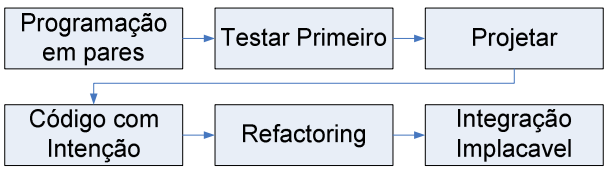

Figura 2.3 – Desenvolvimento de *Software* com XP

FONTE: Adaptado de Extreme Programming Guia Pratica (2002)

2.4.1.1.
Princípios, Responsabilidades e Etapas XP

Astels (2002) resume a metodologia XP em 14 princípios, descritos no Apêndice I. Em relação às responsabilidades, a metodologia XP propõe duas equipes de trabalho: A equipe do cliente e a equipe do desenvolvimento, descritos no Apêndice II. A figura 2.4, descreve as etapas da XP (planejamento inicial, planejamento das iterações, e implementação) descritas com maior detalhe no Apêndice III.

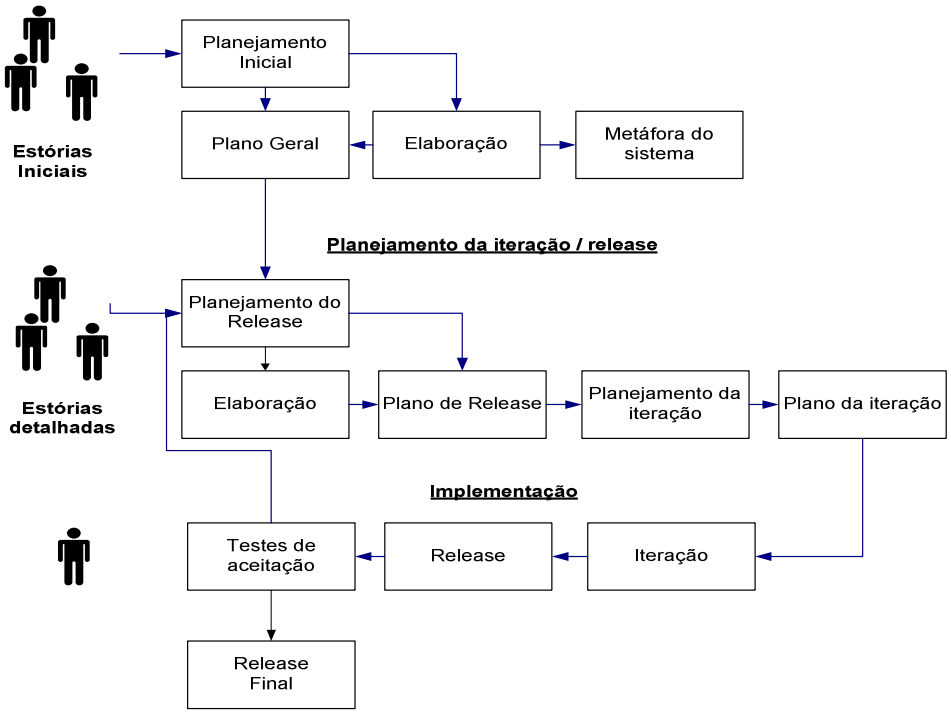

Figura 2.4 – Etapas da XP

FONTE: Martins, Técnicas para o gerenciamento de projetos de *software*, 2007, pag 291

2.4.2.
SCRUM

As raízes do Scrum são encontradas num artigo chamado "O jogo do desenvolvimento de novos produtos" publicado na *Harvard Business Review* em janeiro de 1986 escrito por Takeuchi e Nonuka. O termo *Scrum* é referido ao jogo *Rugby*, que consiste em que os jogadores são organizados em círculos para planejar sua próxima jogada. Dessa forma é mostrado que um projeto se pode ser conduzido em ciclos pequenos, com visão de longo prazo, tendo como principal objetivo ganhar o jogo (MARTINS, 2007).

Scrum foi criado por Ken Schwaber e Jeff Sutherland em 1996, como um método de gerenciamento que aceita que o desenvolvimento de *software* seja imprevisível e aplicável a projetos com muitas mudanças. (SCHWABER, 2004).

Kniberg (2007) e Martins (2007) descrevem a *Scrum* como um *framework* e uma série de práticas que ajudam a gerenciar os projetos de forma visível. Isso permite à equipe saber o que acontece ao longo das etapas do projeto, o qual permite identificar problemas e desvios no desenvolvimento, de modo tal que possam ser tomadas as melhores decisões para direcionar o projeto para o melhor rumo.

2.4.2.1. Fases e Responsabilidades e Elementos no SCRUM

De acordo com Kniberg (2007), Scrum apresenta 3 fases : Pré-jogo, Desenvolvimento e Pós-jogo, ilustradas na Figura 2.5.

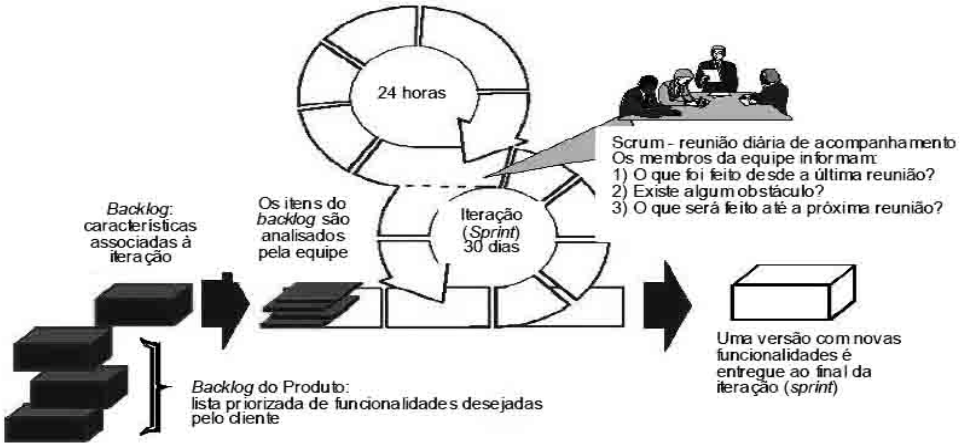

Figura 2.5 – Fases Scrum

FONTE: Agille Aliance (2009)

Na fase do **pré-jogo**, *Scrum* faz um planejamento inicial do projeto, são definidos os padrões, conversões, arquitetura, tecnologia e recursos. No pré-jogo é definido o *Backlog* do Produto, que são os requisitos principais do sistema a desenvolver.

Na fase de **desenvolvimento** são desenvolvidas todas as iterações definidas no *Backlog* do Produto, incrementando gradualmente o produto final, considerando regras de gerenciamento como as **reuniões diárias, lições aprendidas, integração continua do sistema, testes de sistema.**

Na fase do **pós jogo** é integrado em um todo o sistema em desenvolvimento, é feito o teste final, e é gerada a documentação mais relevante para o sistema com a versão final do sistema.

Na Tabela 2.5 se descrevem as responsabilidades da equipe de desenvolvimento *Scrum*.

Tabela 2.5 – Responsabilidades Scrum

Product Owner	Define os requisitos do produto, decide a data de release e o que deve conter nela;
	É responsável pelo Retorno Financeiro (*ROI*) do produto;
	Prioriza os requisitos de acordo com o seu valor de mercado;
	Pode mudar os requisitos e prioridades a cada *Sprint*;
	Aceita ou rejeita o resultado de cada *Sprint*.
SCRUM *Master*	Garante que o time esteja totalmente funcional e produtivo;
	Facilita a colaboração entre as funções da equipe e elimina os impedimentos do time;
	Protege o time de interferências externas;
	Garante que o processo está em andamento;
	Participam das reuniões diárias, revisão da *Sprint* e planejamento.
SCRUM Team	Multifuncional, entre 5-9 membros;
	Seleciona, entre os itens priorizados, os que irão ser executados durante a *Sprint*;
	Tem todo o direito de realizar o que quiser dentro da *Sprint* para cumprir o objetivo da iteração;
	Auto-organizado: organiza o time e o trabalho entre os membros de forma participativa;
	Ao final da *Sprint*, realiza o demo do produto finalizado.

Fonte: Adaptado de Kniberg (2007)

A Tabela 2.6 descreve os elementos, e a linguagem utilizada na metodologia *Scrum*.

Tabela 2.6 – Elementos no Scrum

Sprints	É uma série de iterações bem definidas.
Time-box	E a duração do sprint de duas a quatro semanas.
Sprint Planning Meeting	Realiza-se uma reunião de planejamento em que o time de desenvolvedores tem contato com o cliente.
Product Owner	E o cliente quem prioriza o trabalho que precisa ser feito, seleciona e estima as tarefas que o time pode realizar dentro da Sprint.
Execução da *Sprint*	Fazer o *Sprint*.
Daily Meeting	Reuniões diárias rápidas - não mais de 15 minutos de duração.
Sprint Burndown	Gráfico de avanço de tarefas usado nos *Daily Meeting.*
Sprint Review	E uma Reunião de Revisão em que o time demonstra o produto gerado na *Sprint* e valida se o objetivo foi atingido.
Sprint Retrospective	Reunião de lições aprendidas, com o objetivo de melhorar o processo/time e/ou produto para a próxima *Sprint.*

Fonte: Adaptado Kniberg (2007)

2.4.3. Feature Drive Development

Esta metodologia foi criada por Jeff de Luca, Peter Code e Stephen Palmer, no ano 1999 num grande projeto de desenvolvimento de *software* num banco de Cingapura (Martíns, 2007). O *Feature Drive Development* (FDD) a diferencia dos outros métodos ágeis se baseia em processos definidos e repetitivos. As principais características desta metodologia são:

- É iterativo;
- Enfatiza a qualidade;
- Entrega resultados tangíveis e frequentes;
- Provê relatórios de progresso precisos e significativos;
- É apreciado pelos clientes, gerentes e desenvolvedores.

O criador deste modelo afirma que os clientes apreciam o FDD, porque com ele é possível obter resultados significativos mais cedo, já que o FDD provê meios para acompanhar a evolução do projeto com segurança, a través dos relatórios de progresso. O FDD foi concebido para ser usado em projetos de desenvolvimento de novos *software*s, como em projetos para evoluir um *software* existente. O FDD usa um modelo de desenvolvimento incremental, e

faz uso de diversos mecanismos de controle e divulgação do trabalho realizado no projeto (*Feature - driven development* web site, 2010).

O ciclo de vida desta metodologia possui etapas bem definidas mostradas na Figura 2.6: Desenvolvimento de modelo geral, Construção de lista de funcionalidades, Planejamento de funcionalidades, Projeto e construção por funcionalidades.

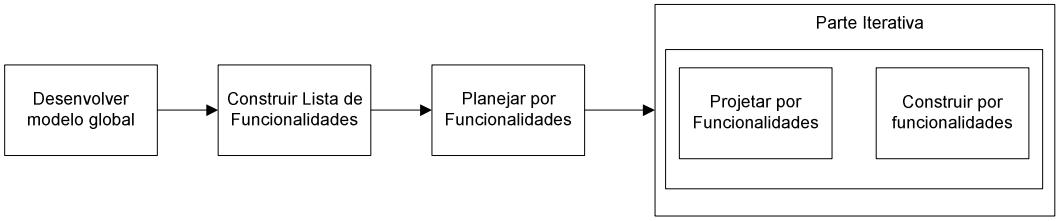

Figura 2.6 – Ciclo de vida FDD

FONTE: Martins (2007)

2.5. A Produção Enxuta de *Software*

A produção enxuta teve seus inícios no Japão na empresa *Toyota Motor Company,* quando a empresa precisava produzir carros em pequenas quantidades, com custos baixos como os da produção em massa.

Na década de 1950, surgiu o Sistema Toyota de Produção, uma forma nova de pensamento na manufatura, tudo o que não agregava valor aos clientes era definido como perda (Womack, Jones e Roos, 1992) . O objetivo é aperfeiçoar os processos e procedimentos, através da redução contínua de desperdícios e tempos elevados de espera entre os processos de produção.

Ghinato (2000) descreve a produção enxuta como uma filosofia de gerenciamento que procura aperfeiçoar a organização de forma a atender as necessidades do cliente no menor prazo possível, com alta qualidade e baixo custo, envolvendo e integrando todas as partes da organização, com o objetivo principal de aumentar a eficiência da produção, eliminando custos desnecessários.

A Produção Enxuta de *Software* teve como inspiração os princípios da produção enxuta da manufatura, originando o termo *Lean Software Development* que explica os princípios adaptados do *Lean* para a área de desenvolvimento de *software* (Poppendieck, M. e Poppendieck, T.,2003). São considerados sete princípios do desenvolvimento enxuto de *software*, explicados a seguir.

1. **Eliminar Perdas:** O pensamento especulativo em projeto de *software* é uma grande fonte de desperdício. A perda ou desperdício é definida como aquilo que não agrega valor ao produto. No caso do desenvolvimento de *software*, quando um desenvolvedor programa mais funcionalidades do que as necessárias, essas são consideradas perdas. E recomendável entregar ao cliente só o necessário no menor tempo possível.
2. **Amplificar o Aprendizado:** Desenvolver *software* é um exercício de aprendizado, os arquitetos de *software* devem acompanhar o trabalho da equipe de desenvolvimento no dia a dia. Uma boa abordagem para melhorar o ambiente de desenvolvimento de *software* é pela amplificação do aprendizado.
3. **Adiar Compromissos e Tomar Decisões o mais tarde possível**: E recomendável que as tarefas nos projetos de *software* sejam trabalhadas em forma evolutiva e iterativa, já que num projeto de *Software* existem decisões que ao longo do tempo são afetadas por diversas mudanças. Quando existem incertezas, é melhor considerar as decisões tardias, já que é melhor retardar as decisões para que estas possam ser tomadas com base nos fatos e não em especulações.
4. **Fazer Entregas Rápidas** Isto com o objetivo de obter uma retroalimentação das informações em forma rápida. E importante considerar que deve-se respeitar a velocidade de desenvolvimento das equipes de trabalho, já que os desenvolvedores que trabalham sob pressão tendem a introduzir mais defeitos no seu trabalho do que desenvolvedores que trabalham em ambiente controlados.
5. **Equipe Responsável:** Isto é, envolver a equipe nas decisões, e detalhes técnicos, já que com a experiência necessária, e guiada por um líder, tomarão boas decisões para o processo de desenvolvimento. O envolvimento da equipe contribui claramente na produção de um projeto mais claro, e elimina retrabalhos.
6. **Construir *software* com integridade:** A integridade no *software* se refere a três atributos: à facilidade de verificação do *software*, à segurança, e controle das informações e à tolerância aos erros, isto é o grau de detecção de erros e possibilidade de funcionar mesmo sob condições adversas. O *software* que possui integridade possui arquitetura coerente, facilidade de uso, atende aos propósitos para o qual foi feito, é manutenível, adaptável e extensível. Segundo POPPENDIECK, M.; POPPENDIECK, T. (2003) a integridade é resultado de uma sábia liderança, experiências relevantes, comunicação

efetiva e disciplinas saudáveis, onde os processos, procedimentos e métricas não são substitutos adequados para eles.

7. **Visualizar o todo:** Procura-se que o desenvolvimento de *software* seja analisado como um todo, já que para obter a integridade é preciso ter conhecimento profundo de todo o produto.

De outro lado na produção enxuta de *software* é usado o conhecido Kanban , similar ao Kanban na produção . Na Figura 2.7 ilustra-se um quadro de cartões de funcionalidades, onde cada coluna representa o trabalho (tarefas ou atividades) segmentado a ser realizado através das iterações, e possui um cartão que corresponde ao objetivo da iteração (Cartão Tema), e abaixo deles são colocados um listado dos cartões correspondentes aos requisitos a serem implementados ao longo do projeto.

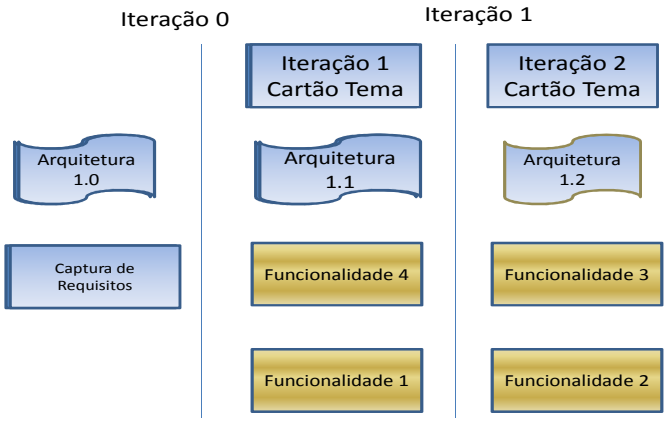

Figura 2.7 – : Quadro de cartões de funcionalidades

Fonte: Adaptado de POPPENDIECK, M.; POPPENDIECK, T., 2003.

Poppendieck, M. e Poppendieck, T. (2003) identificaram os principais desperdícios no desenvolvimento de *software*.

1. **Trabalho Parcialmente Finalizado:** Têm a tendência de se tornarem obsoletos e obstruem o desenvolvimento de outras funcionalidades que precisam ser feitas. Estes trabalhos não permitem verificar se eles irão eventualmente funcionar, consome recursos de investimentos que ainda precisam produzir resultados.
2. **Processo Extra:** A documentação é considerada um artefato necessário e um produto a ser entregue, mas não significa que ela adicione valor. Se for preciso fazer algum documento e este não agrega muito valor para os

clientes é recomendável que eles sejam breves, de alto nível, e de preferência, sem utilizar os desenvolvedores para elaborá-los.

3. **Funcionalidades Extras:** Quando um desenvolvedor adiciona novas funcionalidades como o objetivo de verificar como funcionam as mesmas, é considerado desperdício, já que precisa ser testada, aumenta trabalho em fazer, rastrear, compilar códigos e manter os mesmos. E existe muita probabilidade que este código extra fique obsoleto mesmo ante de ser usado.
4. **Espera:** No desenvolvimento de *software*, as maiores fontes de desperdício são as esperas as ocorrências de eventos, atrasos nos inícios dos projetos, recrutamento do pessoal, atrasos pela documentação dos requisitos, testes, instalação do sistema, todos estes são considerados fontes de perdas. Quando o desenvolvimento é realizado num domínio em expansão, os atrasos representam um problema sério e podem determinar qual será a empresa melhor sucedida.

2.6. Projetos

O PMBOK (Project Management Body of Knowledge) é Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos, e de acordo com este guia um projeto é um esforço temporário empreendido para criar um serviço, produto, ou um resultado exclusivo. (PMBOK, 2008).

Para Kerzner (2002), um projeto é um empreendimento com um objetivo identificável, que consome recursos e opera sob pressões de prazos, custos e qualidade.

De acordo com Pellegrinelli (1997), um projeto tem uma duração fixa, objetivos claros e definidos e é focado na entrega de um único objetivo.

Segundo Nocêra (2009), um projeto é uma ideia que se forma para executar ou realizar algo no futuro; plano; intento; desígnio.

Vargas (2005), afirma que um projeto é um empreendimento não repetitivo, caracterizado pela seqüência clara e lógica de eventos com inicio, meio e fim, e são conduzidos por pessoas dentro de parâmetros predefinidos de tempo, custo, recursos envolvidos e qualidade.

De acordo com o Project Management Institute (2008), as características de um projeto são:

É temporário: Todo projeto possui um início e um final definido. O final é alcançado quando os objetivos tiverem sido atingidos, quando se tornar claro que os objetivos do projeto não serão ou não poderão ser atingidos ou quando não existir mais a necessidade do projeto e ele for encerrado. Temporário não significa necessariamente de curta duração; muitos projetos duram vários anos. Em todos os casos, no entanto, a duração de um projeto é finita. Projetos não são esforços contínuos.

Produtos, serviços ou resultados exclusivos: Um projeto cria entregas exclusivas, que são produtos, serviços ou resultados. Os projetos podem criar:

- ✓ Um produto ou objeto produzido, quantificável e que pode ser um item final ou um item componente.
- ✓ Uma capacidade de realizar um serviço, como funções de negócios que dão suporte á produção ou á distribuição.
- ✓ Um resultado, como resultados finais ou documentos.

Elaboração progressiva: Elaboração progressiva significa desenvolver em etapas e continuar por incrementos. A elaboração progressiva das especificações de um projeto deve ser cuidadosamente coordenada com a definição adequada do escopo do projeto, particularmente se o projeto for realizado sob contrato. Quando o escopo do projeto é adequadamente definido, o trabalho a ser feito deve ser controlado conforme as especificações do projeto e do produto.

2.6.1.
Gerenciamento de Projetos

A humanidade planeja e gerencia projetos desde o início da civilização. Mesmo sem as ferramentas, técnicas e metodologias que atualmente são conhecidas, as pessoas criavam prazos de projeto, faziam controle de custos, programação de materiais e recursos, e avaliação de riscos. Com o passar do tempo as pessoas perceberam que aquelas técnicas que usavam tanto para o controle de custos, controle de prazos, aquisição de recursos e gerenciamento de riscos poderiam se aplicar a diferentes projetos. (PMI® - SP, 2010).

O gerenciamento de projetos é a aplicação do conhecimento, habilidades, ferramentas e técnicas às atividades do projeto, a fim de atender aos seus requisitos. Os gerentes de projetos fazem isso ao padronizar tarefas rotineiras para obter resultados repetitivos e reduzir o número de tarefas que poderiam ser

negligenciadas ou esquecidas. O gerenciamento de projetos é realizado através da aplicação e da integração dos seguintes processos de gerenciamento de projetos: **Iniciação, Planejamento, Execução, Monitoramento e Controle, e Encerramento**. O gerente de projetos é a pessoa responsável pela realização dos objetivos do projeto. (PMBOK, 2008).

Segundo Kerzner (2002), a maturidade na gerência de projetos pode não ser possível se não existe um processo repetitivo, que possa ser padronizado para outros projetos.

Um projeto bem sucedido segundo PMBOK (2008) deve:

- ✓ Selecionar os processos adequados dentro dos grupos de processos de gerenciamento de projetos necessários para atender aos objetivos do projeto.
- ✓ Usar uma abordagem definida para adaptar os planos e as especificações do produto de forma a atender aos requisitos do produto e do projeto.
- ✓ Atender aos requisitos para satisfazer as necessidades, desejos e expectativas das partes interessadas.
- ✓ Balancear as demandas conflitantes de escopo, tempo, custo, qualidade, recursos e risco para produzir um produto de qualidade.

2.6.2.
Processos de Gerenciamento de Projetos

De acordo com o PMBOK (2008) e Nôcera (2009), o gerenciamento de projetos é realizado por meio da aplicação e integração de cinco processos de gerenciamento apresentados na Figura 2.8.

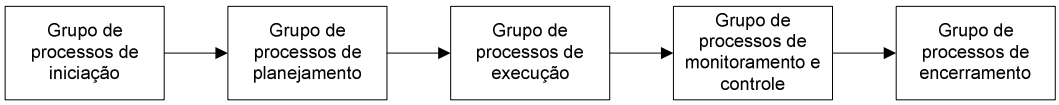

Figura 2.8 Grupos de processo de gerenciamento de projetos.

Fonte: Adaptado PMBOK 2008

- ✓ **Grupo de processos de iniciação.** Define e autoriza o projeto ou uma fase do projeto.
- ✓ **Grupo de processos de planejamento.** Define e refina os objetivos e planeja a ação necessária para alcançar os objetivos e o escopo para os quais o projeto foi realizado.
- ✓ **Grupo de processos de execução.** Integra pessoas e outros recursos para realizar o plano de gerenciamento do projeto para o projeto.

- ✓ **Grupo de processos de monitoramento e controle.** Mede e monitora regularmente o progresso para identificar variações em relação ao plano de gerenciamento do projeto, de forma que possam ser tomadas ações corretivas quando necessário para atender aos objetivos do projeto.
- ✓ **Grupo de processos de encerramento.** Formaliza a aceitação do produto, serviço ou resultado e conduz o projeto ou uma fase do projeto a um final ordenado.

Nocêra (2009) descreve os benefícios do gerenciamento de projetos, e assinala que geralmente numa organização os benefícios estão relacionados aos seguintes aspectos de entrega do produto do projeto:

- **Técnicos:** O resultado do produto de projeto, conforme requerido.
- **Prazo de entrega:** Obtenção do produto do projeto na data requerida.
- **Custo:** Custo final de acordo com o planejado.
- **Satisfação do cliente:** Com o produto final prazo e custo.
- **Satisfação das partes interessadas** e membros da equipe do projeto.

2.6.3. Ciclo de Vida do Projeto

As organizações como os gerentes de projetos, podem dividir projetos em fases com o objetivo de oferecer um melhor controle gerencial com ligações adequadas com as operações em andamento da organização executora (PMBOK, 2008). Estas fases são conhecidas como ciclo de vida do projeto. O ciclo de vida do projeto define as fases que conectam o início de um projeto ao seu final, e este pode ajudar ao gerente de projetos a esclarecer se deve tratar o estudo de viabilidade como a primeira fase do projeto ou como um projeto autônomo separado. Quando o resultado desse esforço preliminar não é claramente identificável, é melhor tratar esses esforços como um projeto separado. As fases do ciclo de vida de um projeto não são iguais aos grupos de processos de gerenciamento de projetos. A Figura 2.9 ilustra o ciclo de vida ao longo do projeto.

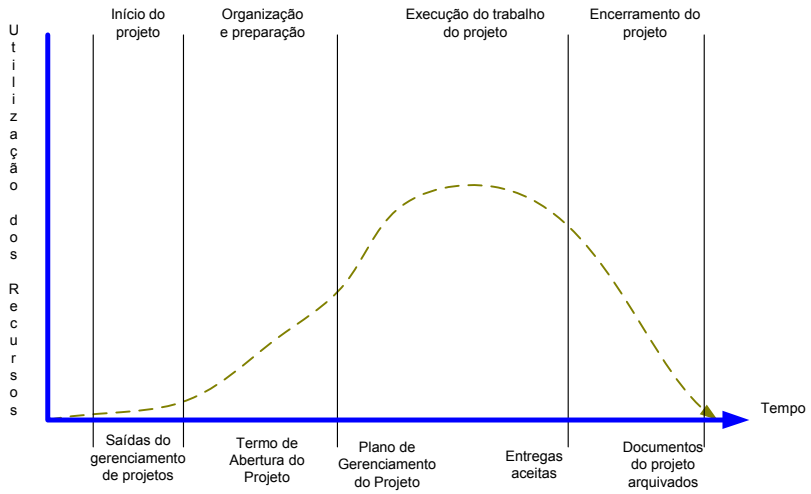

Figura 2.9 – Nível de custos e pessoal no ciclo de vida do projeto

FONTE: PMBOK 2008, pag. 16

O inicio do projeto, geralmente, é definido quando a organização identifica uma oportunidade de negócio que deseja aproveitar e as fases do ciclo de vida são sequenciais.

O risco de não atingir os objetivos é maior no início do projeto, pelo nível de incertezas. A certeza de término geralmente se torna cada vez maior conforme o projeto continua.

A Figura 2.10 apresenta como as partes interessadas têm capacidade de influenciarem as características finais do produto do projeto e o custo final do projeto com maior grau no início e torna-se cada vez menor conforme o projeto continua. Contribui para esse fenômeno o fato de que o custo das mudanças e da correção, conforme o projeto continua.

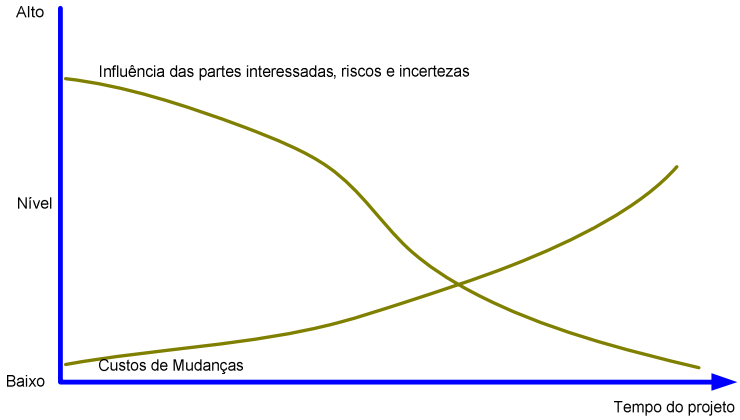

Figura 2.10 – Influência das partes interessadas ao longo do tempo

FONTE: PMBOK 2008, pag. 17

2.7.
Gerenciamento da Qualidade do Projeto

Marshall (2008) afirma que os sistemas de gestão da qualidade inicialmente foram desenvolvidos em organizações com procedimentos padronizados, mas existem muitos setores voltados ao desenvolvimento de produtos e/ou serviços exclusivos, para este caso a organização por projeto é a melhor opção para esse contexto dinâmico.

PMBOK (2008) analisa o gerenciamento da qualidade com uma abordagem compatível com as normas ISO de qualidade, e com abordagens recomendados por Deming, Juram, Crosby, Gerenciamento da Qualidade Total (GQT), Seis Sigma, Análise de modos e efeitos de falha, Revisões de projeto, Voz do cliente, Custo da qualidade (CDQ) e Melhoria contínua.

O processo para gerenciar a qualidade do projeto inclui diversas atividades das unidades executoras do próprio projeto, e elas serão as responsáveis em determinar o sistema de gerenciamento da qualidade que é implantado por meio das responsabilidades, objetivos e políticas de qualidade, procedimentos e processos de planejamento de qualidade**.**

Gerenciar a qualidade do projeto aborda:

- **Gerenciamento do projeto geral** que se aplica a todos os projetos, independentemente da natureza de seu produto, e consiste no trabalho que deve ser feito com o objetivo de obter um produto de acordo com as funções especificadas.
- **Gerenciamentos do produto do projeto,** que são medidas e técnicas de qualidade do produto, e são específicas do tipo particular de produto produzido pelo projeto.

2.7.1.
Processos de Gerenciamento da Qualidade

Segundo PMBOK (2008), existem três principais processos de gerenciamento da qualidade, mostrados na Figura 2.11.

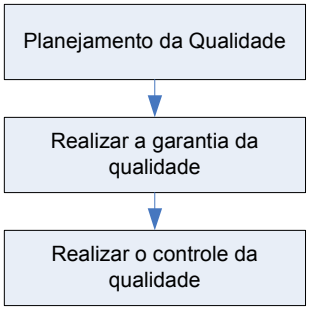

Figura 2.11 – Processos de Gerenciamento da Qualidade
Fonte: Adaptado PMBOK 2008

1. **Planejamento da qualidade:** Identificam-se os padrões de qualidade relevantes para o projeto e determina-se como satisfazer os mesmos**.**
2. **Realizar a garantia da qualidade:** Aplicam-se as atividades de qualidade planejadas e sistemáticas para garantir que o projeto emprega todos os processos necessários para atender aos requisitos**.**
3. **Realizar o controle da qualidade:** Monitoram-se resultados específicos do projeto para determinar se estão de acordo com os padrões relevantes de qualidade e se identificam as maneiras de eliminar as causas dos desempenhos insatisfatórios**.**

Segundo Marshall (2008) e Nôcera (2009), o planejamento da qualidade deve identificar os padrões de qualidade que são relevantes para o projeto e determinar como atingi-los.

Numa organização, o responsável da implementação da política da qualidade é a alta direção, e o gerente de projetos é o responsável pela qualidade do projeto. Para definir a qualidade do projeto, o gerente deve:

- Definir os requisitos que satisfaçam as especificações dadas pelo cliente.
- Definir a equipe com as suas responsabilidades.
- Desenvolver os diversos procedimentos e padrões do projeto, e monitorar o desempenho do projeto no longo da sua duração.

2.7.2. Metas da Qualidade e Ferramentas de Qualidade

Segundo o PMBOK (2008), as metas da qualidade podem estar vinculadas:

- **Aos produtos do projeto:** Neste caso o cliente percebe a qualidade do produto, condições de preço, serviços associados.
- **Ao desempenho da organização no desenvolvimento do produto:** Refere-se à produtividade e à qualidade do processo.

Nôcera (2009) afirma que as ferramentas de qualidade podem ser técnicas, dispositivos, gráficos demonstrativos, esquemas de execução, e diversos procedimentos utilizados para a implantação e verificação da qualidade de um produto ou serviço. Algumas das ferramentas mencionadas no PMBOK para o monitoramento e controle da qualidade são:

1. **Diagrama de causa efeito:** Chamados de diagramas de Ishikawa ou diagramas espinha de peixe, ilustram como diversos fatores podem ser ligados a possíveis problemas ou efeitos.
2. **Gráficos de Controle:** O objetivo de um gráfico de controle é determinar se um processo é ou não estável ou tem desempenho previsível. Podem servir como uma ferramenta de coleta de dados para mostrar quando um processo está sujeito a uma variação de causa especial, que cria uma condição fora de controle.
3. **Elaboração de fluxogramas:** E uma representação gráfica de um processo.
4. **Histograma:** E um gráfico de barras que mostra a distribuição de variáveis. Cada coluna representa um atributo ou uma característica de um problema/situação. A altura de cada coluna representa a frequência relativa da característica. Esta ferramenta ajuda a identificar a causa de problemas em um processo pela forma e amplitude da distribuição.
5. **Diagrama de Pareto:** E um tipo específico de histograma, ordenado por frequência de ocorrência, que mostra quantos defeitos foram gerados por tipo ou categoria de causa identificada.
6. **Gráfico da execução:** E um gráfico de linha que mostra pontos de dados traçados na ordem em que ocorrem.
7. **Diagrama de dispersão:** Mostra o padrão da relação entre duas variáveis. Esta ferramenta permite que a equipe de qualidade estude e identifique a possível relação entre as mudanças observadas em duas variáveis.
8. **Amostragem estatística:** A amostragem estatística envolve a escolha de uma parte de uma população de interesse para inspeção.

9. **Inspeção:** O exame de um produto do trabalho para determinar se ele está de acordo com as normas. Em geral, os resultados de uma inspeção incluem medições.
10. **Revisão de reparo de defeito:** Uma revisão de reparo de defeito é uma ação tomada pelo departamento de controle da qualidade ou por uma organização com nome semelhante para garantir que os defeitos do produto foram reparados e estão em conformidade com os requisitos ou especificações.

2.7.3. Gerência da Qualidade em *Software*

A qualidade em projetos de *software* procura garantir qualidade num produto de *software* a través dos processos de desenvolvimento e assim garantir que satisfaça ao cliente, dentro do que foi acordado na declaração inicial do projeto. Rocha (2001) afirma que ao falar de qualidade em produtos de *software* pode-se distinguir a qualidade desde o ponto de vista do usuário, onde a qualidade tem a ver com a facilidade de uso, desempenho e a confiabilidade dos resultados. Desde o ponto de vista do cliente a qualidade refere-se á taxa de defeitos, facilidade de manutenção e conformidade de acordo aos requisitos iniciais. E a qualidade do ponto de vista da organização refere-se ao cumprimento de prazo, custos e produtividade.

Segundo Rocha (2001), nos processos de desenvolvimento de software, a geração de documentação propicia uma melhor organização durante o desenvolvimento de sistemas, já que isto facilita a manutenção do produto desenvolvido. Afirma que a criação de documentação é tão importante quanto o desenvolvimento do *software*, já que reduz perdas de tempo por parte da equipe para desenvolver as diversas fases do projeto, reduz erros e aumenta a qualidade do projeto.

Mnkandla e Dwolatsky (2006) afirmam que a documentação a ser gerada deve ser discutida no momento do planejamento do projeto de *software*, e deverá definir o que realmente é importante para o projeto o que contribui com o bom desenvolvimento do *software*. A Tabela 2.7 descreve os principais fatores de qualidade que as metodologias ágeis consideram.

Tabela 2.7 – Fatores de Qualidade para as Metodologias Ágeis

Etapa	Pratica	Descrição
Revisão	Manutenibilidade	Facilidade para modificar o *software* para corrigir defeitos ou reunir novos requisitos.
	Verificabilidade e validação	Facilidade em testar o sistema.
	Extendibilidade	Habilidade do sistema em adaptar-se as novas especificações.
Implantação	Portabilidade	Facilidade de instalar o *software* em diferentes plataformas de hardware e *software*.
	Reusabilidade	*Software* que é composto de elementos que podem ser utilizados para construir outras aplicações.
	Compatibilidade	*Software* que é composto de elementos que podem, facilmente, combinar com outros elementos.
	Custo-efetividade	Habilidade do sistema de ser completado com o orçamento previsto.
Operação	Usabilidade	Facilidade com que as pessoas podam aprender a utilizar o *software*.
	Eficiência	Quantidade de recursos de computação e de código exigida para que um programa execute sua função.
	Corretitude	Habilidade do sistema de agir de acordo com a especificação definida.
	Timelines	Entrega de versão do *software* antes ou exatamente quando o cliente precisa.
	Robustez	Execução apropriada do sistema sobre casos não cobertos pela especificação. Esse fator complementa a corretitude.
	Integridade	Quando o acesso ao *software* ou aos dados por pessoas não autorizadas pode ser controlado.

Fonte: Mnkandla e Dwolatsky (2006)

Com o objetivo de ajudar as organizações desenvolvedoras de *software* a definir os processos, existem diversos modelos de qualidade, entre eles ISO 9000, ISO/IEC 12207, CMM, CMMI. Estes modelos têm como objetivo apontar as características que um bom processo de *software* tem que apresentar.

Uma norma derivada da ISO 9000 é a norma ISO 10006 , que é um padrão internacional específico para gerenciamento de projetos . Esta norma é aplicável a projetos de diversas complexidades em diferentes ambientes incluindo hardware, *software*, material processado, serviços ou suas combinações.

2.7.4. CMMI

Na área de desenvolvimento de *software* foi criado o modelo de maturidade CMM onde são aplicados conceitos do TQM ao desenvolvimento de *software*.

Para o CMM a principal causa dos problemas em projetos de *software*, é a falta de um processo claro, definido e efetivo, já que conceber os processos significa conceber como os produtos ou serviços são planejados, produzidos e

entregues. Segundo Fiorini (1998) a Capability Maturity Model (CMM), também conhecido como *Software* CMM (SW-CMM) pode ser definido como soma de "melhores práticas" para diagnóstico e validação de maturidade do desenvolvimento de *software* numa organização. O CMM propõe um caminho gradual que leva as organizações a se aprimorarem continuamente na procura de soluções de problemas inerentes ao desenvolvimento de *software*.

CMM requer documentação, mais ele não tende à burocratização, já que ele propor que o processo documentado seja adaptado às características da empresas e características do *software* que desenvolve.

CMM descreve os principais elementos de um processo de desenvolvimento de *software*, descreve os estágios de maturidade por os quais passam as organizações enquanto evoluem no seu ciclo de desenvolvimento de *software*, através de avaliação contínua, identificação de problemas e ações corretivas, dentro de uma estratégia de melhora de processos.

Para CMM, um nível de maturidade é um patamar evolutivo, bem definido, visando alcançar um processo de *software* maduro. Os níveis são uma forma de priorizar as ações de melhoria para aumentar a maturidade do processo. Cada nível compreende um conjunto de metas, e gera como resultado um aumento na capacitação do processo da organização. Existem cinco níveis de maturidade que caracterizam o nível de capacitação do processo da organização: Inicial, Repetitivo, Definido, Gerenciado e Em Otimização. (FIORINI, 1998). Os cinco níveis de maturidade do CMM são descritos no Apêndice IV, e ilustrados na Figura 2.12.

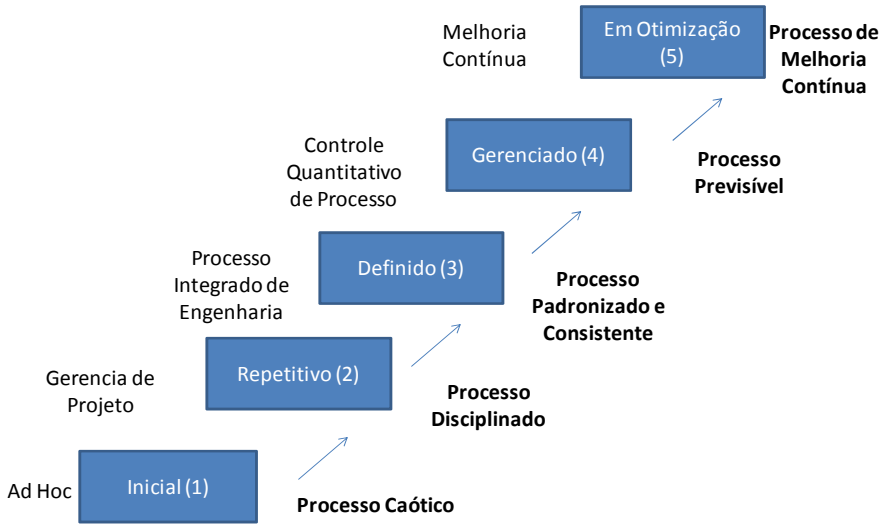

Figura 2.12 – Níveis de Maturidade do CMM , Fiorini 1998

Fonte: Fiorini 1998

Segundo o SEI (2007), uma evolução mais recente ao CMM, é o CMMI (*Capability Maturiry Model Integration*) que consiste num conjunto de boas praticas de atividades de desenvolvimento e manutenção em produtos de *software*. CMMI aborda diversas praticas desde a concepção do produto ate a entrega e manutenção do produto final.

CMMI trabalha com duas representações a contínua e estagiada. A eleição destas representações de caminho de melhoria de processos depende dos interesses das empresas.

- **Representação Contínua:** E aquela que permite á organização melhorar seus processos de trabalho em ritmos variados ou diferentes, é mais recomendável para empresas que buscam melhorias em áreas especificas. Nesta representação a organização trabalha com cinco níveis de capacidade que são descritos na Tabela 2.8

Tabela 2.8 –Níveis de Capacidade CMMI

Nível 0	Incompleto (Ad-hoc)
Nível 1	Executado (Definido), neste nível o processo executado produz unicamente o trabalho necessário.
Nível 2	Gerenciado , neste nível é confrontado o planejado versus o executado.
Nível 3	Definido, este nível trabalha com processos definidos onde são mantidos e descritos os processos existentes.
Nível 4	Quantitativamente gerenciado, neste nível os processos são gerenciados através de diversas técnicas estatísticas.
Nível 5	Em otimização , os processos são alterados com o objetivo de atender ás necessidades de negócios das empresas.

Fonte: SEI ,2006

Nesta representação é possível ter processos com níveis diferentes que variam de acordo aos interesses da empresa. A representação continua no CMMI, está organizado em 4 categorias, como descreve a Tabela 2.9:

Tabela 2.9 –Categorias da Representação Continua CMMI

Categoria	Descrição	Nível de Maturidade
Gerência de Processo	OPF: Foco no Processo da Organização	3
	OPD: Definição do Processo da Organização	3
	OT: Treinamento Organizacional	3
	OPP: Desempenho do Processo Organizacional	4
	OID: Inovação e Melhoria Organizacional	5
Gerência de Projeto	PP: Planejamento de Projeto	2
	PMC: Acompanhamento e Controle de Projeto	2
	SAM: Gerencia de Acordos com Fornecedores	2
	IPM: Gerencia Integrada de Projeto	3
	RSKM: Gerencia de risco	3
	QPM: Gerencia Quantitativa de Projeto	4
Engenharia	REQM: Gerencia de Requisitos	2
	RD: Desenvolvimento de Requisitos	3
	TS: Solução Técnica	3
	PI: Integração de Produto	3
	VER: Verificação	3
	VAL: Validação	3
Suporte	CM: Gerencia de Configuração	2
	PPQA: Garantia da Qualidade de Processo e Produto	2
	MA: Medição e Analise	2
	DAR: Analise de Decisão e Resolução	3
	CAR: Analise Causal e Resolução	5

Fonte: SEI ,2006

- **Representação Estagiada:** E aquela que permite á organização um caminho de melhoria predefinido e testado pelo mercado, e trabalha com processos claros e níveis de maturidade para cada etapa de melhoria de processo. A representação estagiada geralmente é usada quando as empresas já usam algum modelo de maturidade. As categorias desta representação são descritas na Tabela 2.10.

Tabela 2.10 – Categorias da Representação Estagiada CMMI

Área de Processo	Descrição	Nível de Maturidade
Gerenciado	REQM: Gerencia de Requisitos	2
	CM: Gerencia de Configuração	2
	PPQA: Garantia da Qualidade de Processo e Produto	2
	MA: Medição e Analise	2
	PP: Planejamento de Projeto	2
	PMC: Acompanhamento e Controle de Projeto	2
	SAM: Gerencia de Acordos com Fornecedores	2
Definido	OPF: Foco no Processo da Organização	3
	OPD: Definição do Processo da Organização	3
	OT: Treinamento Organizacional	3
	IPM: Gerencia Integrada de Projeto	3
	RSKM: Gerencia de risco	3
	RD: Desenvolvimento de Requisitos	3
	TS: Solução Técnica	3
	PI: Integração de Produto	3
	VER: Verificação	3
	VAL: Validação	3
	DAR: Analise de Decisão e Resolução	3
Gerenciado Quantitativamente	OPP: Desempenho do Processo Organizacional	4
	QPM: Gerencia Quantitativa de Projeto	4
Em Otimização	OID: Inovação e Melhoria Organizacional	5
	CAR: Analise Causal e Resolução	5

Fonte: SEI ,2006

2.8.
Diferenças entre as Metodologias ágeis e tradicionais

Decidir o tipo de abordagem a utilizar no momento de gerenciar um projeto de desenvolvimento de *software*, não é um processo simples, já que a eleição pode mudar a cultura empresarial, tempo, investimentos etc. Nas metodologias tradicionais as decisões são centralizadas, e nas metodologias ágeis as decisões são descentralizadas, MARTIN (2007).

Numa publicação da revista Pro- Qualití, Qualidade na Produção de *Software*, Magalhães, Vasconcelos e Roullier (2005) descrevem claramente as principais diferenças entre as metodologias ágeis e tradicionais mostradas na Tabela 2.11.

Tabela 2.11 – Diferencias entre as metodologias tradicionais e ágeis

Metodologias Tradicionais	Metodologias Ágeis
São preditivas: Cada etapa de desenvolvimento do projeto é baseada na etapa anterior, só funciona bem se a concepção do sistema não sofre nenhuma alteração, ou seja, parte do princípio que os requisitos são estáveis. Quando se percebe que uma definição não foi a mais acertada, a tendência natural é resistir às mudanças, o que prejudica a evolução do sistema.	**São adaptativas**: Partem do princípio de que o conhecimento sobre o problema aumenta à medida que o sistema vai sendo desenvolvido, levando a uma busca constante por melhores soluções. O desenvolvimento ocorre de forma iterativa e evolutiva, com iterações tão curtas quanto possíveis (a literatura sugere de 3 a 4 semanas). Ao final de cada iteração, obtém-se uma versão com a implementação de um novo subconjunto de funcionalidades, pronta para ser colocada em produção.
São orientadas a processos: Partem do princípio de que processos bem definidos devem ser impostos e executados, para garantir a qualidade do produto resultante. Pessoas são tratadas como recursos alocados, da mesma forma que equipamentos e ferramentas. Desconsidera o talento pessoal, o que acaba afetando o moral e, em conseqüência, a produtividade da equipe.	**São orientadas a pessoas:** Para melhor gerenciar projetos de *software*, pessoas devem ser tratadas como indivíduos e não como recursos, pois cada pessoa apresenta um ritmo de trabalho único e completamente diferente, que é fruto da sua vivência pessoal. Os membros da equipe escolhem as tarefas que irão desenvolver durante a iteração seguinte, além de estimarem o tempo a ser gasto com elas, o que aumenta a motivação, o comprometimento e a produtividade da equipe.
São rígidas: Pressupõem que é possível especificar de antemão todos os requisitos do *software* a ser desenvolvido, dificultando e muitas vezes impedindo realizar alterações e comprometendo a evolução natural da solução. Estão mais voltadas para a obtenção do *software* previamente especificado do que para o sucesso do projeto ou negócio para o qual o *software* foi apontado como solução.	**São flexíveis e iterativas:** Adaptam-se constantemente ao conjunto de requisitos mais atual: a cada iteração, usuários, clientes e desenvolvedores decidem sobre quais características devem ser adicionadas, modificadas e até retiradas do sistema. Além de ser desenvolvido da forma mais iterativa possível, é possível escolher os pontos mais críticos ou que mais agregam valor ao sistema para serem desenvolvidos em primeiro lugar. Estão mais voltadas para o bem do negócio, seu caráter adaptativo aumenta as chances de oferecer um melhor resultado ao projeto e ao negócio.
São burocráticas: Geram muita sobrecarga no desenvolvimento a ser realizado, comprometendo a velocidade de desenvolvimento e, muitas vezes, o sucesso do projeto.	**Buscam constantemente a simplicidade**: Partem do princípio de que é preferível fazer algo simples de imediato e pagar um pouco mais para alterá-lo depois, se necessário, do que fazer algo complicado de imediato e acabar não utilizando depois.

Fonte: Pro- Qualití, Qualidade na Produção de *Software*, Magalhães, Vasconcelos e Roullier (2005)

2.9. Combinando Scrum com XP

Segundo Kniberg (2007), a metodologia Scrum é focada nas práticas de gerenciamento e organização, e XP dá mais atenção às tarefas de programação e desenvolvimento de *software*. Kniberg destaca algumas das melhores práticas XP que são diretamente tratadas pelo Scrum, como a programação em par, desenvolvimento orientado a testes, o design incremental, a integração contínua, a propriedade coletiva do código, ambiente de trabalho informativo, padrão de codificação, ritmo sustentável / trabalho energizado, isto é trabalho de 40 horas semanais.

De acordo com Kniberg (2007), estas duas metodologias em conjunto contribuem claramente no bom gerenciamento de projetos de *software*.

Nascimento (2008) descreve as características em comum entre XP e Scrum que estão descritas na Tabela 2.12:

Tabela 2.12 – Características comuns entre XP e Scrum

Características	XP	SCRUM
Refactoring	x	
Programação em pares	x	
Posse coletiva de código	x	
Testes automatizados	x	
Testes unitários	x	x
Testes funcionais	x	x
Entrega de versões	x	x
Processo de colocação de versões em produção	x	x
Processo de finalização (treinamento e implantação)	x	x
Testes de Integração	x	x
Visão Global do Projeto	x	x
Troca de informações intensa entre o cliente e o desenvolvedor	x	x
Identificação de novos requisitos	x	x
Preparação/ Evolução da documentação		x
Planejamento global do projeto	x	x
Processo de Iniciação do projeto	x	x
Planejamento das iterações	x	x
Estimativa de custo das funcionalidades	x	x
Ciclos de desenvolvimento curtos	x	x
Definição de condições mínimas para o projeto (infraestrutura, equipe, ferramentas)		x
Reuniões regulares	x	x
Revisão do estado do projeto		x
Aprendizagem para a próxima iteração		x
Workshops para refinamento do produto e da metodologia		x

Identificação de riscos	x	x
Equipes multidisciplinares	x	x
Equipes auto- organizáveis	x	x
Equipe de desenvolvimento sempre possui visão global do projeto	x	x
Trabalho em equipe	x	x
Divulgação dos Resultados	x	

Fonte: Adaptado de Nascimento 2008 pag 40, Um modelo de referência para o desenvolvimento ágil de *software*

Com a analise feita por Nascimento (2008), pode-se confirmar a afirmação de Kniberg (2007) e concluir que as duas metodologias são compatíveis para montar um modelo de referência.

2.10. Características Comuns Scrum e CMMI

Considerando que CMMI é um conjunto de "melhores práticas" para diagnóstico e validação de maturidade do desenvolvimento de *software* numa organização, é importante saber o grau de compatibilidade destas práticas com as metodologias ágeis. CMMI soma um conjunto de práticas que são mais adaptadas às metodologias tradicionais, mas tem a característica de serem menos burocráticas do que as metodologias ágeis, pelo que CMMI foi comparado e contrastado com algumas das principais metodologias ágeis de gerenciamento de projetos de *software* com o objetivo de avaliar o grau de compatibilidade entre estas metodologias.

Segundo Shwaber (2004) as práticas de gerenciamento Scrum e as práticas propostas pelo CMMI, têm características em comum, já que Scrum tem a capacidade de mostrar os requisitos de sistema e como são feitas as entregas dos requisitos nos projetos de *software*, Shwaber faz a análise dos requisitos em comum das duas metodologias e conclui que trabalhando com Scrum podem-se cumprir requisitos até um nível três das boas práticas CMMI, como se observa na Tabela 2.13.

Tabela 2.13 – Características comuns entre Scrum e CMMI

Nivel	Área Chave	Cumpre Requisito Scrum	Cumpre Requisito CMM
2	Gestão de requisitos	X	X
2	Planificação de Projetos de *software*	X	X
2	Seguimento e supervisão de projetos de *software*	X	X
2	Subcontratação de *software* de gestão.		
2	Controle de qualidade no *software*	X	X
2	Gestão da configuração no *software*		X
3	Organização focada no proceso		X
3	Organização define o proceso		X
3	Treinamento na programação		X
3	Gestão integrada de *software*		X
3	Engenharia de produto de *software*	X	X
3	Coordenação na equipe de trabalho	X	X
3	Revisão por pares	X	X

Fonte: Shwaber ,2004

O CMMI afirma que as áreas chaves de processo dos níveis de maturidade 2 e 3, trazem maior retorno imediato e são os que estabelecem quase total correspondência com as cláusulas 5.4; 5,5; 5.7; 5.8; e 5.9 de normas ISO 10006 derivada da ISO 9000 quando são aplicadas a *software* .

2.11. Considerações Finais

O capítulo abordou conceitos de processo de desenvolvimento de *software*, metodologias tradicionais e ágeis para o desenvolvimento de projetos de *software*, assim como conceitos de gerenciamento de projetos, produção enxuta, e praticas de qualidade CMMI.

Hoje em dias as mudanças acontecem de forma inevitável, e segundo Vargas (2005), as empresas passam a ser reconhecidas por sua flexibilidade e pela capacidade de atender seus clientes. Afirma que as corporações de sucesso percebem que o uso de conceitos de gerenciamento de projetos hoje é universal, rompe barreiras culturais, e de localidade, onde as necessidades de sobrevivência competitiva são universais.

O material revisado no capítulo permitirá fazer uma análise das principais características destes métodos estudados, para realizar um modelo de referência que possa ajudar ao gerenciamento de projetos de *software*, e ser compatível com as principais praticas de qualidade em projetos de *software*.