REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA

GESTÃO ÁGIL DE PROJETOS DE SOFTWARE VERSUS PMBOK

Rafael Serra – *Faculdade Anhanguera de Taubaté - unidade 2*

RESUMO: Ao longo dos tempos, muito s foi falado sobre o desenvolvimento de software. Diversas técnicas foram desenvolvidas, estudadas e aprimoradas, a fim de mitigar os resultados negativos, ocasionados pela falta de gestão de projetos. Observa- se que o PMBOK (Project Manager Body Knowledge), guia de boas práticas para gestão de projetos, não é suficiente para atender às necessidades da indústria de software. Compreender o projeto, dividir suas etapas e identificar as necessidades, são meios que ajudam a escolher a melhor forma de gerenciar um projeto de software. O método Scrum, utilizado na gestão ágil de projetos de software, quando usado em paralelo ao guia PMBOK, traz recursos suficientes para atender satisfatoriamente todas as áreas do gerenciamento de projetos, desde sua concepção até a entrega final, focando principalmente a interação entre clientes e a equipe do projeto, apresentando uma tratativa segura para os imprevistos ocasionados por uma gestão instável de requisitos.

PALAVRAS-CHAVE:
Métodos de desenvolvimento ágil, gestão de projetos, PMBOK, Scrum.

ABSTRACT: In the course of times, the development of software has been a lot persecuted. Many techniques were developed and studied in order to mitigate negative results caused by lacking of project management. It is observed that the PMBOK guide to good practices for project management is not enough to meet the needs of the software industry. Understanding the project, identifying the needs, are means that help to choose the best way to manage a software project. The Scrum method, used in software engineering, when used in parallel to the PMBOK, provides sufficient resources to meet satisfactorily all areas of project management, focusing primarily on the interaction between clients and project team.

KEYWORDS:
Agile development methods, project management, PMBOK, Scrum.

Artigo Original
Recebido em: 08/05/2012
Avaliado em: 14/02/2012
Publicado em: 23/05/2014

Publicação
Anhanguera Educacional Ltda.

Coordenação
Instituto de Pesquisas Aplicadas e Desenvolvimento Educacional - IPADE

Correspondência
Sistema Anhanguera de Revistas Eletrônicas - SARE
rc.ipade@anhanguera.com

1. INTRODUÇÃO

Desde o início da década de 50, muito se falou sobre desenvolvimento de *software*. Isso se deu ao fato de que, no início, o desenvolvimento era tratado como "estado da arte", onde o desenvolvimento era baseado em tentativa e erro (PRESSMAN, 2006).

A partir da década de 60, o conceito de desenvolvimento de *software* passou a sofrer mudanças. Com isso, as técnicas utilizadas para o desenvolvimento, como o processamento em lote (*Batch*), foram substituídas por conceitos estruturados e sistematizados, onde o *software* passou a ser considerado o resultado de um conjunto de etapas a serem seguidas, nascendo assim Engenharia de *Software* (ROUILLER, 2008).

Grandes mudanças no cenário da indústria de *software* foram percebidas a partir de 2002. A popularização do conhecimento, através da facilidade de acesso a informação, trouxe a necessidade de desenvolver sistemas computacionais com qualidade, em curto intervalo de tempo e com custo competitivo (BURNETT; MACHADO, 2002). Se somado às constantes mudanças legislativas dos Municípios, Estados e União, com o real crescimento global, o mercado de *software* torna-se altamente competitivo e em evidente expansão.

Todas essas necessidades fizeram com que surgissem diversos métodos de desenvolvimento, com o foco no aprimoramento da gestão de projeto e relação entre equipe de projeto e cliente, sistematizando assim, os processos de criação, visando à produtividade (SOARES, 2004).

Uma nova onda de relacionamento invadiu o mundo da gestão de projetos de sistemas computacionais, pois, não mais importava apenas o *software* em si, mas a satisfação do cliente em receber um produto, que condiz com sua necessidade, surgindo assim, uma relação de parceria entre cliente e fornecedor (REZENDE, 2005).

Estudos realizados pelo *Standish Group* (*CHAOS SUMARY*, 2011) mostram como a aplicação de práticas de gestão de projetos podem mudar os resultados das empresas.

Tabela 1. O Relatório do Caos (STANDISH GROUP, 2011).

ANO	FRACASSO/FALHA	ATRASO/PREJUIZO	SUCESSO
2010	21%	42%	37%
2008	24%	44%	32%
2006	19%	46%	35%
2004	18%	53%	29%
2002	15%	51%	34%
2000	23%	49%	28%
1998	28%	46%	26%
1996	40%	33%	27%
1994	31%	53%	16%

Pressman (2006) deixa claro que a ausência de métodos de gestão de projetos é o principal responsável pelo *software* legado, que são os sistemas desenvolvidos décadas atrás e que necessitam continuamente de melhorias.

O autor também aponta as características da ausência de gestão de projetos:

- As estimativas de prazos e custos são imprecisas;
- Padrões existem, mas não são seguidos;
- Objetivos não definidos;
- Falta de gestão de mudanças;
- A produtividade e o comprometimento das pessoas envolvidas no desenvolvimento não acompanham a demanda de mercado e;
- A qualidade do *software*, às vezes, é menor do que a adequada.

O presente estudo promove meios para desenvolver uma visão sistêmica, de como realizar gestão de projetos para sistemas computacionais. Pretende-se apresentar o desenvolvimento de *software* como ramo da engenharia, com isso, sua gestão deve obedecer a uma sequência sistematizada de processos.

O conhecimento adquirido com as boas práticas da gestão ágil, aliado ao GUIA *PMBOK* (*PMI*, 2008), deve ampliar o conhecimento específico de gestão de projetos de *software*, utilizando-se de forma tácita, as experiências cotidianas.

Desta forma, é possível atender uniformemente todas as áreas envolvidas, de modo a compreender ciclicamente cada processo e atuar separadamente em cada um, seja processo incremental ou iterativo, sem perder a visão sistêmica do projeto.

2. MÉTODOS CLÁSSICOS

Os chamados Métodos Clássicos, ou Métodos Pesados, também conhecidos como Métodos Baseados em Processos Definidos, surgiram junto à engenharia de *software*, por volta da década de 60/70 (MARTINS, 2007).

Podem ser resumidos num conjunto de regras que focam a otimização dos processos, de forma a possuir uma abordagem sistêmica, no qual é adotada uma postura de desenvolvimento estruturado, focado na documentação e no planejamento. O uso deste método é mais aconselhado quando os passos a serem executados são conhecidos (BUENO et al, 2009).

Existem diversos métodos que tratam o desenvolvimento de *software*, entretanto cada um deles possuem suas características e peculiaridades.

Sommerville (2007) relata a importância dos métodos de desenvolvimento de *software* como uma representação abstrata, no qual cada método representa uma determinada perspectiva, obtendo e fornecendo informações sobre o processo. Dentre os métodos clássicos, podem-se destacar:

- Método de Desenvolvimento em Cascata: Utiliza-se de uma rotina de agrupamento de tarefas, que são executas sequencialmente, obedecendo à precedência de cada grupo. Suas disciplinas são classificadas em especificação, desenvolvimento, validação e evolução.

- Método de Desenvolvimento Evolucionário: Intercala as etapas de especificação, desenvolvimento e validação, através de protótipos de sistema. Espera-se, por meio de interações com o cliente, alcançar os requisitos definidos no projeto, com o uso do controle de versões.

- Método de Engenharia de S*oftware* Baseada em Componente: Baseia-se no reuso de componentes. Espera-se que uma quantidade significativa de componentes possa ser reutilizada, diminuindo assim o tempo de desenvolvimento.

Embora sejam métodos distintos, na realidade o uso geralmente é realizado através de práticas combinadas para obtenção do *software* final (PRESSMAN, 2006).

3. MÉTODOS ITERATIVOS

Ao longo do projeto, mudanças de requisitos e prioridades são inevitáveis. O que caracteriza um método como iterativo, é o fato da especificação de requisitos ocorrer de forma combinada com o desenvolvimento do *software*.

Segundo Martins (2007) as causas dos problemas encontrados no decorrer do projeto, na maioria dos casos são similares, tais como:

- Falta de análise;
- Gerenciamento informal de requisitos;
- Falta de capacidade de lidar com as mudanças de escopo e;
- Falta de testes;

Pfleeger (2004) caracteriza a entrega de um projeto de *software* iterativo, como incremental, ou seja, o *software* é partido em uma série de pequenos incrementos e desenvolvido separadamente.

Sommerville (2007) destaca a importância de uma estrutura de desenvolvimento espiral, onde a evolução do *software* é para fora, ou seja, inicia-se com um esboço que é desenvolvido até a entrega final.

3.1. Unified Process

O *Unified Process* (*UP*) é um método de gestão de projetos de *software*, que utiliza como ferramenta para análise de requisitos a *UML* (*Unified Modeling Language*), linguagem de modelagem unificada (PFLEEGER, 2004).

Martins (2007) aponta o *UP* como um método iterativo dividido em:

- Desenvolvimento;
- Gerenciamento de requisitos;
- Arquitetura baseada em componentes;

- Modelagem do *Software* através da *UML;*
- Verificação constante da qualidade;
- Controle de mudanças;
- Organização estática e dinâmica do sistema e;
- Trabalho focado na arquitetura e nos casos de uso.

O *UP* utiliza uma sequência de diretrizes para alcançar seu objetivo, ou seja, *software* com qualidade, dentro das expectativas do cliente.

Para o caso de haver alterações de requisitos e funcionalidades no decorrer do projeto, que não foram previstas pela equipe de projeto, o *UP* não responde adequadamente.

3.2. Rational Unified Process

O *Rational Unified Process* (*RUP*) é um método de projetos de *software*, originário de estudos realizados a partir da *UML* e do *UP*, pela *Rational Software Corporation.* Traz em sua estrutura, elementos originários dos principais métodos clássicos. (RIBEIRO; SOUZA, 2005).

Sommerville (2007) apresenta três aspectos, que descrevem o *RUP*.

- Dinâmico: Apresenta as fases ao longo do tempo;
- Estático: Apresenta as atividades realizadas e;
- Prático: Trata-se de uma sugestão de boas práticas.

Segundo Ribeiro e Souza (2005), o *RUP* é constituído por quatro fases:

- Concepção: Coleta e dados, definição do plano de negócio;
- Elaboração: Definição do cronograma, arquitetura e ferramentas;
- Construção: Desenvolvimento Incremental;
- Transição: Onde é realizada a implantação e treinamento do sistema.

No *RUP* o foco de cada fase está em produzir os produtos e documentações necessários para atender o objetivo do projeto, ainda sim, existem falhas ao que se refere à gestão e controle de requisitos e não há clareza ao que se refere à documentação.

3.3. Métodos ágeis

Os chamados Métodos Ágeis, também conhecidos como Métodos com Controle Empírico, foram popularizados em 2001, com a reunião de especialistas em engenharia de *software,* no qual foi definido o Manifesto Ágil, onde princípios comuns para todos os métodos ágeis foram definidos (TELES, 2005).

- Indivíduos e interações ao invés de processos e ferramentas;
- *Software* funcional, em vez de documentação abrangente;
- Colaboração do cliente, ao invés de negociação de contratos e;
- Respostas a modificações, em vez de seguir um plano.

Ao estudar as literaturas de Pfleeger (2004), Pressman (2006), Martins (2007) e Sommerville (2007), pode se destacar os Métodos Ágeis, *Extreme Programming*, *Prototipação* e o *Scrum*, este último método é abordado a partir do capitulo 4.

Extreme Programming

Nascido em meados da década de 80, a partir de pesquisas realizadas por Kent Beck e Ward Cunnighan (MARTINS, 2007), o *Extreme Programming* (*XP)* tem seu foco na diminuição dos processos de desenvolvimento, acreditando na excelência da aplicação das técnicas de desenvolvimento, comunicação e trabalho em equipe (PRESSMAN, 2006).

Soares (2004) apresenta o *XP* da seguinte forma:

- Filosofia de desenvolvimento baseado em comunicação transparente;
- Conjunto de práticas de otimização do desenvolvimento;
- Sessões de trabalho com interações de curta duração, resultando em objetividade no retorno das informações;
- Abordagem incremental de planejamento e desenvolvimento, que pode ser alterada ao longo do ciclo de vida do projeto;
- Automatização das rotinas de testes do projeto e;
- Incentivo à comunicação oral entre a equipe do projeto.

O *XP* baseia seu desenvolvimento em um fluxo contínuo de atividades.

As entregas acontecem em grandes blocos, com grandes pacotes de funcionalidades e grandes implementações (SOMMERVILLE, 2007).

De acordo com estudos baseados na literatura de Soares (2004), Martin (2007) e Sommerville (2007), os papéis no *XP* são divididos em 6 fases, que vão da idealização a entrega:

- Exploração: Onde ocorre à identificação de requisitos e funcionalidades;
- Planejamento: Definição das prioridades com o cliente;
- Iterações: Desenvolvimento dos incrementos priorizados;
- Validação: Teste e verificação do *software;*
- Manutenção: Fase de correção do incremento entregue e;
- Morte: Quando não há mais manutenção a ser realizada e o cliente esta satisfeito com o produto entregue.

O *XP* espera alcançar a melhoria dos processos, utilizando recursos de programação em pares, com a utilização de dois programadores etapa de desenvolvimento.

Embora possua uma filosofia iterativa e ágil, o *XP* possui uma desvantagem na entrega do produto, pois trata o desenvolvimento em grandes blocos de incremento, o que gera uma quantidade significativa de ciclos de desenvolvimento.

Prototipação de Software

Um protótipo é a versão inicial de um sistema, utilizado para demonstrações e experimentos (SOMMERVILLE, 2007).

Pressman (2006) indica a *Prototipação*, quando os requisitos de *software* são imprecisos ou inexistentes, no entanto, há claro entendimento do problema.

Segundo Pfleeger (2004) o objetivo da *Prototipação* é reduzir os riscos e as incertezas do desenvolvimento de *software*, utilizando etapas rotativas para o desenvolvimento:

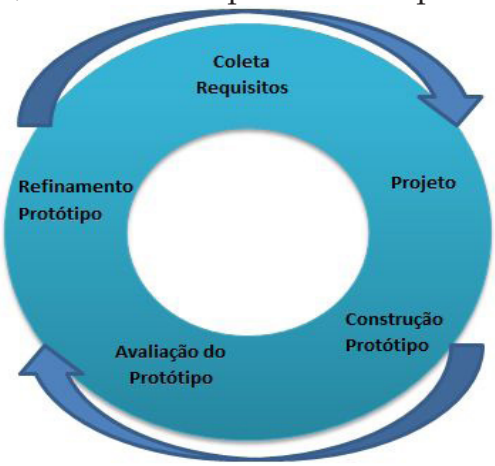

Figura 1. Ciclo de vida da *Prototipação* (PFLEEGER, 2004)

A *Prototipação* é um método indicado para pequenas equipes de projetos, pois permite que os envolvidos com o projeto experimentem, avaliem os requisitos e tracem novas metas.

Este ciclo rotativo de etapas exige um alto grau de controle de versões, pois do contrário pode gerar um sistema de baixa qualidade.

4. SCRUM

Concebido em 1993 por Jeff Sutherland, John Scumiotales e Jeff Mckenna, o *Scrum* possui uma filosofia iterativa, adaptativa e incremental. Este processo de inovação contínua tem o intuito de entregar "valor" ao cliente (MARTINS, 2007).

O *Scrum* é um método baseado em iterações, que, através de reuniões periódicas, são avaliadas, de forma adaptativa e flexível, questões que visam determinar o estado do projeto, em vista das metas da equipe.

Schwaber (2009) descreve os papéis e as responsabilidades do *Scrum*:

- O Dono do Produto (*Product Owner*): Representa os interesses de todos os envolvidos no projeto, requisitos, metas, investimentos, etc;
- Mestre *Scrum* (*Scrum Master*): Gerencia o projeto e serve como facilitador para os demais membros da equipe e;
- Equipe (*Team*): Desenvolve as funcionalidades do produto.

Para que o projeto tenha início, é necessário que o Dono do Produto, apoiado por toda a Equipe, defina e priorize uma lista de itens, com as funcionalidades e requisitos que o *software* irá possuir. Esta lista recebe o nome de *Backlog* do Produto.

Tabela 2. Exemplo de Backlog do produto, (MARTINS, 2007).

Backlog do Produto								Atualização	dd/mm/aaaa
								IED real diário (%)	XX%
ITER#	ID	Assunto	Item	Prior.	Resp.	IED	Risco	Dias estimados	Dias pendentes
1	15	Filas Backup	Mudar a arquitetura para filas de chamadas para processamento	1	AG	4	A	6,7	4,4
5	3	Segurança	Logar as pesquisas de gravação feitas pelos usuários e as reproduções	2	AC	0,5	B	0,8	3,1
3	8	Relatório	relatório de auditoria	2	AG	0,5	B	0,8	???

- Iter*:* Número que a iteração será tratada;
- Id: Código único atribuído a cada item;
- Assunto: Forma de agrupamento dos itens;
- Item: Descrição/definição do item;
- Prior: Prioridade que item possui sobre o *backlog*, no qual a escala é definida por toda a equipe, mas a atribuição cabe apenas ao Dono do Produto;
- Resp: Colaborador de Equipe responsável pelo item;
- IED: Estimativa de prazo ideal considera 100% de dedicação do colaborador por dia de trabalho;
- IED real diário: Porcentagem real que o colaborador poderá dispor no dia de trabalho
- Risco: Nível de incerteza do item, quanto à estimativa;
- Dias estimados: Representado pela fórmula (IED/ (1 - IED real diário)), mostra a quantidade real de dias para conclusão do item e;
- Dias pendentes: Quantos dias faltam para concluir o item.

No *Scrum*, tudo nasce do b*acklog* do produto. Este *backlog* é desenvolvido, mantido e atualizado, através de pequenas reuniões diárias, com no máximo 15 minutos de duração, podendo ser prorrogada por mais 15 minutos, chamadas de *Scrum Meeting*, onde são divididas as atividades e separadas em *Sprints*.

Segundo Schwaber (2009) *Sprint* é um conjunto de atividades de desenvolvimento. Todo *Sprint* começa com uma reunião de planejamento colaborativo entre o Dono do Produto e a equipe de projeto, onde são definidas as próximas iterações e atualizado o *backlog* do *Sprint*.

Ao longo do projeto, diversos *Sprints* serão definidos a este conjunto de *Sprints* nomeia-se *Release*.

Após análise dos estudos realizados por Martins (2007), Araujo (2009) e Schwaber (2009), as etapas do método *Scrum* são descritas como:

- Pré-*game*: É a etapa onde os requisitos são definidos e priorizados, o esforço necessário é calculado e os possíveis riscos são avaliados;
- *Game*: É a etapa no qual o produto é desenvolvido. Envolve as atividades de análise do b*acklog* de produto, reuniões para rever o planejamento, revisão dos padrões que o sistema precisa ter compatibilidade e execução das atividades dos *Sprints* e;
- Pós-*game*: É a etapa onde ocorrem as entregas do produto, implantação, homologação, testes e treinamento. Nesta etapa o produto é preparado para implantação, treinamento dos usuários e documentação do produto.

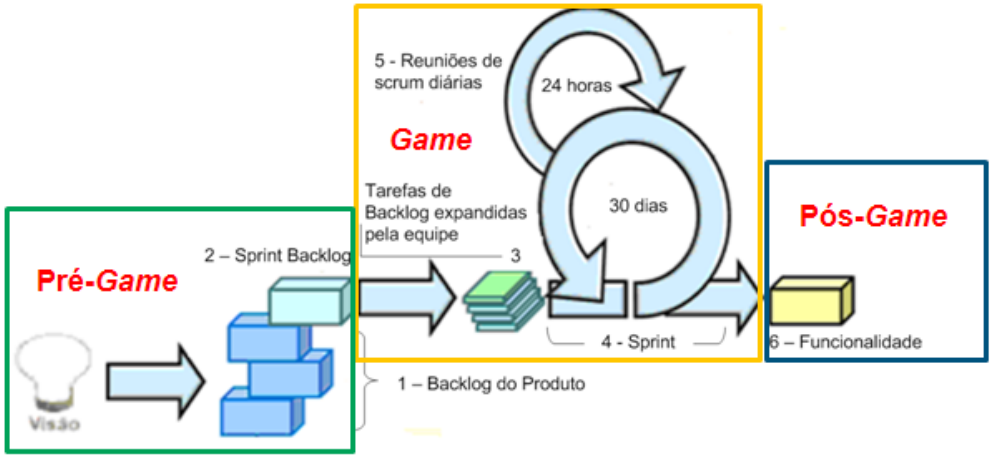

Figura 2. Visão geral adaptada do Método Scrum (BUENO et al, 2009).

O *Scrum* é uma forma de mostrar que qualquer tipo de projeto pode ser conduzido em pequenos ciclos, sem perder a visão ampla do projeto. Em outros termos, o *Scrum*, foca quais são os interesses e as necessidades do cliente.

4.1. Gestão da informação

Uma das principais vantagens do *Scrum* é a representação gráfica. O *backlog* dos *Sprints* pode ser representado através do Gráfico de Tendências dos *Sprints* ou *Burndown Chart*, que mostra quanto de trabalho resta para fazer, como pode ser visto na figura 3.

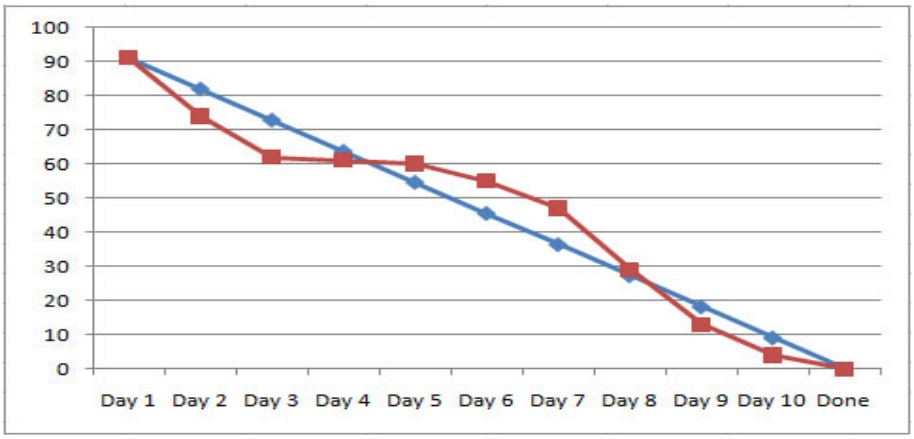

Figura 3. Sprint Burndown. Gráfico de Tendências (MARTINS, 2007).

No *Scrum*, o andamento do projeto é transparente para todos os membros da equipe. Outro recurso utilizado para informação do andamento dos *Sprints* é o Quadro de Tarefas do *Sprint*, conforme figura 4.

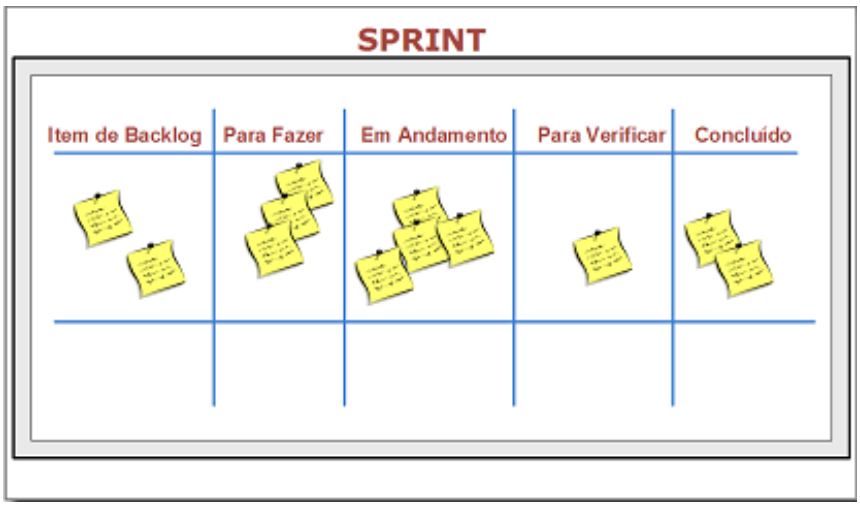

Figura 4. Quadro de Tarefas do Sprint (ARAUJO, 2009).

A escolha do *Scrum* se dá pelo fato de ser um método simples e objetivo, composto por poucas regras e recursos, onde é possível tratar diversos problemas de desenvolvimento, valorizando o conhecimento empírico de cada membro da equipe.

As facilidades do *Scrum* em adaptar regras e práticas de outros métodos, juntamente com a autogestão da equipe, foram às principais qualidades encontradas neste método.

GERENCIA DE PROJETOS DE *SOFTWARE*					
ETAPAS	***SCRUM***	***XP***	***UP***	***RUP***	***PROTOTIPAÇÃO***
ADAPTATIVO	SIM	SIM	SIM	SIM	SIM
ITERATIVO	SIM	SIM	SIM	SIM	SIM
CICLOS	POUCOS CICLOS ROTATIVOS	QUANTIDADE RAZOAVEL DE CICLOS DEPENDENTES	GRANDE QUANTIDADE DE CICLOS DEPENDENTES	POUCOS CICLOS DEPENDENTES	POUCOS CICLOS ROTATIVOS
GESTÃO DA INFORMÇÃO	AUTO GESTÃO	NECESSITA ACOMPANHAMENTO CONSTANTE	NECESSITA ACOMPANHAMENTO CONSTANTE	NECESSITA ACOMPANHAMENTO CONSTANTE	NECESSITA ACOMPANHAMENTO CONSTANTE
DOCUMENTAÇÃO	OBETIVA	OBETIVA	COMPLETA	COMPLETA	OBETIVA
ENTREGA	INCREMETAL DE PEQUENOS BLOCOS	INCREMENTAL DE GRANDES BLOCOS	INCREMETAL DE PEQUENOS BLOCOS	ENTREGA DO PRODUTO COMPLETO.	INCREMETAL DE PEQUENOS BLOCOS

O *Scrum* parte do princípio que se o problema for atacado em conjunto, as soluções serão mais rápidas e criativas, mas para que isso exista é necessário que a equipe tenha comprometimento. Por isso, o *Scrum* é um método de equipes auto organizadas e auto dirigidas.

5. PMBOK

De acordo com *Project Management Institute* (*PMI*), um projeto é um empreendimento temporário, com intuito de desenvolver um produto, serviço ou resultado exclusivo.

Gerenciar projetos é a aplicação do conhecimento, práticas, técnicas, habilidades e ferramentas, a fim de atender aos seus requisitos.

Conforme *PMI* (2008) a gestão de projetos é realizada através da integração e aplicação dos 42 processos, agrupados em 5 fases:

- Iniciação;
- Planejamento;
- Execução;
- Monitoramente e controle e;
- Encerramento

Desde meados de 2002, falhas de projetos de sistemas intensivos de *software* chegam a 75%. Isso se dá, principalmente, ao fato de desenvolvedores e adquirentes não possuírem conhecimento amplo de gestão (BURNETT; MACHADO, 2002).

O *PMBOK* é a união de conhecimentos, habilidades e técnicas de gestão de projetos, reunidos em um guia de boas práticas. O objetivo é disseminar o conhecimento e padronizar o vocabulário entre gerentes e equipes do projeto (*PMI*, 2008).

5.1. O ciclo de vida do projeto

O ciclo de vida do projeto são fases geralmente sequênciais, que podem sobrepor umas às outras, voltadas para o gerenciamento e controle das organizações, natureza do projeto e sua aplicação (*PMI*, 2008).

Muitas vezes, as companhias adotam ciclos de vidas padrões, para serem utilizados em todos os seus projetos (MARTINS, 2007).

O ciclo de vida define as etapas que existem entre o início e o fim do projeto, no geral, classificam os trabalhos a serem realizados, as entregas, o monitoramento e o controle, com variações de acordo com a complexidade e tamanho do projeto (*PMI*, 2008).

Processos estão presentes em todas as áreas de conhecimento do projeto. Um exemplo claro disso é o ciclo *PDCA* (*plan-do-check-act*) que nada mais é do que planejar, fazer, validar e agir, ou ainda na indústria de *software* o *XP*, *RUP*, *UP*, *Prototipação* e o *Scrum* que é o objeto de estudo deste artigo.

5.2. Áreas de conhecimentos

As áreas de conhecimento do projeto são descritas pelo *PMI* (2008), em nove áreas distintas e 42 processos:

- Gerenciamento de Integração: Processos e atividades que compõem os elementos do projeto. São eles: Termo de abertura, escopo preliminar, orientação, mudanças e finalização.

- Gerência de Escopo: Verifica os processos e prevê os trabalhos para término do projeto. Etapa que se desenvolve o escopo, a estrutura analítica do projeto (EAP) e a verificação de controle de escopo.
- Gerência do Tempo: Os processos são definidos com tempo para entrega, os recursos de tempo são estimados e é realizado o controle do cronograma do projeto.
- Gerência de Custos: São definidos os custos e orçamentos dos processos com as devidas estimativas.
- Gerência de Qualidade: Planejamento e qualidade. Estimativas são definidas para garantir os atendimentos das expectativas.
- Gerência de Recursos Humanos (RH): Envolve a organização da equipe do projeto, sendo: planejamento de RH, mobilização de equipe e gerenciamento e desenvolvimento da mesma.
- Gerência de Comunicação: Coleta, disseminação, armazenagem e destino das informações. Classificam-se: Planejamento, distribuição, desempenho e gerenciamento das áreas.
- Gerência de Riscos: Principal gerência. Foca a qualificação e quantificação dos requisitos do projeto. São classificadas em: planejamento e gerenciamento de riscos. Engloba: análise, planejamento, controle e monitoramento.
- Gerência de Aquisições e Contratos: Relaciona-se a compras e contratos. São classificadas em: planos de compras e aquisições, contratações, controle de fornecedores, administração e encerramento de contratos.

No gerenciamento de projetos é importante observar que muitos dos processos são iterativos, por haver a necessidade de uma elaboração progressiva, com isso, o conhecimento da equipe do projeto aumenta conforme o andamento do projeto.

Na gestão ágil de projetos de *software*, fica evidenciado, que, métodos ágeis possuem carências significativas em diversas áreas de conhecimento, como recursos humanos, escopo, comunicação e etc., no qual, espera-se suprir, com os ensinamentos abordados pelo GUIA *PMBOK (PMI,* 2008).

6. GESTÃO ÁGIL

O projeto de *software* nasce a partir da visão de uma necessidade de negócio. A partir desta visão, é definida uma lista inicial de atividades, com requisitos e funcionalidades. Neste momento, toda estrutura do *software* é idealizada informalmente (MARTINS, 2007).

O ambiente moderno de negócios é apressado e sempre mutável. Gerenciar todas as entradas e demandas de projeto não é uma tarefa fácil (PRESSMAN, 2006).

Percebe-se então, que em meio a este cenário totalmente incerto, há necessidade de um

método que seja capaz de gerenciar todas as áreas de projeto, com transparência, inspeção e adaptabilidade.

Conforme descrito por Schwaber (2009), o *Scrum* é um método de engenharia de *software* ágil, fundamentado pela Teoria de Controle Empírico, emprega uma abordagem iterativa e incremental, que visa aperfeiçoar a previsibilidade do projeto e mitigar os riscos.

6.1. PMBOK aplicado ao Scrum

O guia *PMBOK* possui uma poderosa base de conhecimento, que pode ser utilizada em parceria com métodos de engenharia de *software* ágeis, garantindo assim, o atendimento de todas as áreas envolvidas no projeto.

As etapas do *Scrum*, quando utilizadas juntamente ao *PMBOK*, tornam a gestão do projeto mais próxima do ideal. As etapas são melhores compreendidas e por haver um padrão em todas as etapas, quando um novo membro é adicionado à equipe ou uma nova funcionalidade, o impacto no projeto é minimizado, pois os processos são todos padronizados.

A aplicação de boas práticas de gestão de projetos na indústria de *software* vem como uma ótima alternativa para sanar lacunas deixadas pela gestão ágil.

Espera-se uma gestão de projetos de *software* com o máximo de controle sobre mudanças e previsibilidade, tornando o andamento do projeto, muito mais seguro.

O Mestre *Scrum*

Cabe ao Mestre *Scrum* controlar todas as entradas e saídas do projeto e fazer a gestão de pessoas da equipe do projeto.

Por agir como facilitador do projeto é necessário possuir conhecimentos nas gerências de integração e Recursos Humanos, onde o GUIA *PMBOK* (*PMI*, 2008) define meios para condução da Equipe, andamento dos *Sprints* e todo o resultado esperado no b*acklog*, através de ferramentas como:

- Termo de abertura;
- Escopo preliminar e;
- Plano de gerenciamento.

Também é parte de seu dever garantir as entregas de cada *Sprint* conforme acordado no *backlog* do produto, além, de garantir a obediência ao que se refere a custos e níveis de qualidade. Neste momento o Mestre *Scrum* pode se valer dos conhecimentos abordados nas gerencias de tempo, custos e qualidade, também apresentados pelo GUIA *PMBOK* (*PMI*, 2008).

O Dono do Produto

Não existe uma regra que auxilie o Dono do Produto na definição do b*acklog* do produto. Com isso, usar recursos da gerência de escopo pode aperfeiçoar os resultados idealizados no início do projeto, através de ferramentas de controle como:

- Escopo do projeto;
- Estrutura analítica do projeto (EAP) e;
- Verificação de controle do escopo.

A utilização destas ferramentas traz ao Dono do Produto confiabilidade para definição de requisitos e funcionalidades, melhor visualização dos objetivos e segurança na definição das estimativas de custo, tempo e etc.

Documentação e armazenamento

Não existe no método *Scrum*, uma regra que trate a documentação do projeto, porém, em muitos casos, se faz necessário, rotinas documentadas e armazenadas de forma adequada visando à preservação da informação.

Para esta situação, a equipe do projeto pode se valer da gerência de comunicação, que visa garantir a documentação do que realmente seja necessário.

O Mestre *Scrum* pode destinar um membro da equipe para ser o responsável pela documentação das atividades, criação e atualização da base de conhecimentos, a este membro é imprescindível possuir conhecimento na gerência de comunicação.

Gestão continuada de riscos

A todo momento o projeto esta exposto a riscos, com isso, se faz necessário que todas as possibilidades de riscos sejam claramente expostas para toda a Equipe, para que nos inícios e términos de cada *Sprint*, a equipe tenha ciência de como deverá caminhar para próxima etapa.

É muito importante que toda a equipe consiga comunicar-se de forma padronizada, utilizando-se de preceitos da gerência de riscos para qualificação e quantificação dos riscos em cada *Sprint*. Essa medida torna a execução do *Sprint* muito mais previsível, tangível e estável.

Resultado das aplicações das técnicas

Espera-se do *Scrum*, um método com aspecto muito mais completo, atendendo de forma padronizada, as áreas que antes não eram possíveis.

Tabela 4. Comparativo do método Scrum após PMBOK

SCRUM		
	ANTES	DEPOIS
RELAÇÃO COM O CLIENTE	Informal	Informal
DESENVOLVIMENTO DO PROJETO	Fechado a equipe inicial do projeto	Aberto para novas contrações
AQUISIÇÃO DE NOVOS RECURSOS DE PESSOAL	Complexa conforme o andamento do projeto	Fácil, pois os processos são padronizados.
DOCUMENTAÇÃO	Não há.	Documentação do que é considerado importante pela equipe do projeto
UTILIZAÇÃO EM GRANDES PROJETOS	Complexo por lidar informalmente com os processos	Possível, pois a padronização traz a segurança necessária.
DEFINIÇÃO DO *BACKLOG* DO PRODUTO	Informal, sem ferramenta de apoio	Faz uso das práticas do *PMBOK*, a fim de mitigar falhas por ausência de requisito.
GESTÃO DE RISCOS	Informal, nas mãos de toda equipe do projeto	Documentada, garantida pelo Mestre *Scrum*, analisada e mantida por todas equipe.

Os conhecimentos adquiridos com as boas práticas do guia *PMBOK*, permitem a melhoria das etapas do *Scrum*, de modo a atender toda a necessidade da equipe de projeto.

Foi possível melhorar as etapas do *Scrum* sem afetar o princípio básico de iteratividade, adaptabilidade e agilidade que o método permite.

Com isso, foi estabelecido um padrão em todas as etapas, o que trouxe mais segurança para os gestores de projeto, pois não há apenas um método baseado em conhecimento empírico, mas sim um método que garante a informação como parte da equipe do projeto.

7. CONSIDERAÇÕES FINAIS

O presente estudo evidenciou a carência existente entre os métodos de engenharia de *software*, com relação às áreas de interesse dos projetos, independentemente de optar por uma abordagem clássica, iterativa, ou ágil.

Porém a utilização do método *Scrum* só foi possível, pois sua filosofia iterativa, adaptativa, incremental e ágil, permite com facilidade adaptar novos processos, conhecimentos e rotinas, sem influenciar no resultado final do projeto.

É possível perceber, que toda mudança proposta com as práticas do *PMBOK*, somente seria possível se houvesse total aceite e contribuição de toda a equipe de projeto.

Atividades que antes eram realizadas baseadas no conhecimento da equipe, hoje são padronizadas mundialmente, o que facilita a gestão de pessoas, gestão do conhecimento e a gestão do projeto de *software*.

A figura 5 mostra o fluxo de processo do *Scrum*, adaptado aos conhecimentos adquiridos com as boas práticas do GUIA *PMBOK (PMI*, 2008).

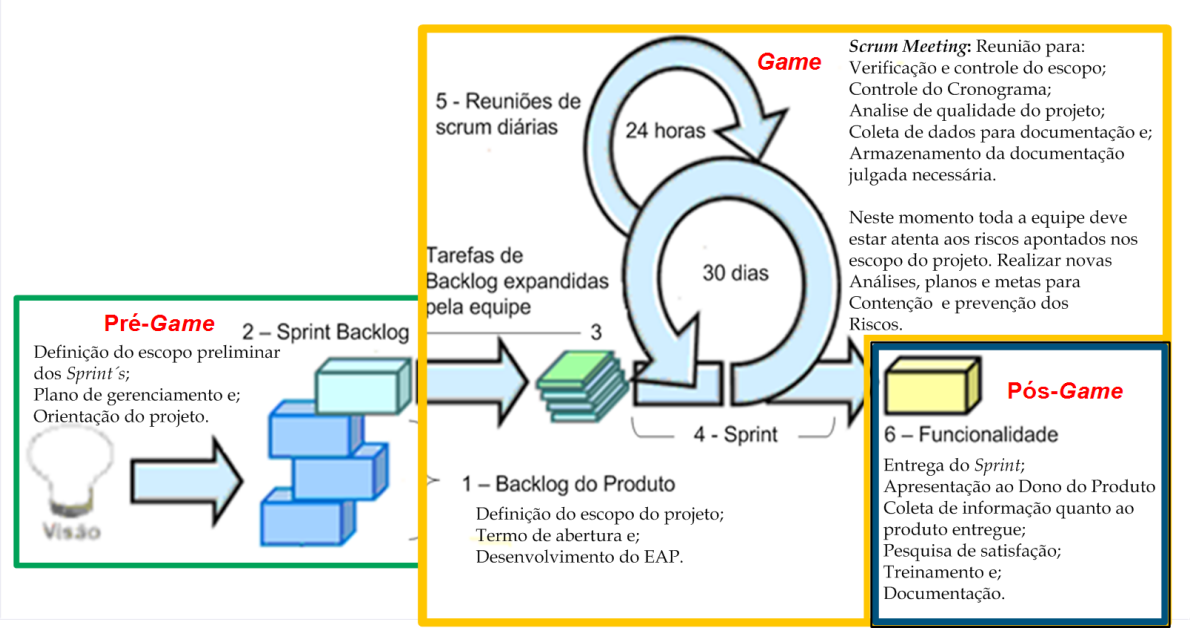

Figura 5. Fluxo do processo *Scrum* adaptado (BUENO et atl, 2009)

É importante ressaltar, que a utilização de boas práticas de gestão de projetos, não alterou o fluxo do processo do método, mas sim, possibilitou uma estrutura padronizada, gerando assim, reeducação de todos os envolvidos, com a inserção de novos conhecimentos.

Com isso foi possível utilizar as facilidades agregadas pelo uso do método *Scrum*, acrescidas pelos benefícios de uma gestão formal, proposta pelos conhecimentos adquiridos com o estudo do *PMBOK*:

- Gestão iterativa de pessoas.
- Processos curtos e padronizados;
- *Software* funcional, com documentação inteligente;
- Projetos com a gestão mais próxima da previsibilidade;
- Colaboração do cliente, resultados documentos e;
- Segurança nas respostas a modificações.

O presente estudo viabilizou a possibilidade dos gestores de projetos, utilizarem uma forma ágil de gestão de projetos de *software* e ainda sim, manter o controle formal sobre todas as áreas de envolvimento do projeto.

REFERÊNCIAS

ARAUJO L.B.P. Estudo comparativo da compatibilidade entre as melhores práticas do PMI e SCRUM. FIAP: São Paulo, 2009.

BUENO R; CLAUDIANO E.C.; PROLUNGATTI J.L; SERRA R. Protótipo de ferramenta de gerenciamento de backup. FAPI: São Paulo, 2009.

BUNETT, R.C; MACHADO, C. A.F. Gerência de projetos na engenharia de software em relação

as práticas do PMBOK. PUC: Paraná, 2002. Disponível em: <http://celepar7cta.pr.gov.br/portfolio.nsf/b239398b4e7d02ec03256d9c003fdcb8/617e42000235b79703256c08006afbc1/$FILE/_h8tin523ecdkm2834ckg70sjfd9in8rrj8pkmsobc_.doc>. Acesso em: 30 dez. 2010.

MARTINS, J.C.C. Técnicas para o gerenciamento de projeto de software. Rio de Janeiro: Brasport, 2007.

PFLEEGER, S.L. Engenharia de Software, teoria e prática. 2° Ed, São Paulo: Prentice Hall, 2004.

PRESSMAN, R.S. Software Engineerin: A Pratitioner´s Approach. 6° Ed, São Paulo: McGRAW-Hill, 2006.

PROJECT MANAGER INSTITUTE. Um guia do conhecimento em gerenciamento de projetos. 4° Ed, PMI, 2008.

REZENDE, D.A.Engenharia de Software e Sistema de Informação. 3° Ed, Rio de Janeiro: Brasport, 2005.

RIBEIRO, M.P; SOUZA, T.P. Rational Unified Process: uma abordagem gerencial. IME – Instituto Militar de Engenharia: Rio de Janeiro, 2005.

ROUILLER, A.C. Melhoria de processo de software livre. UFLA/FAEPE, 2008. Disponível em: <http://gerpro2008.googlecode.com/files/Apostila_GERENCIA_DE_PROJETOS_DE_SOFTWARE.pdf > Acesso em 05/Jun/2011.

SCHWABER, K. Guia do Scrum. ScrumAlliance, 2009.

SOARES, M.S. Métodos Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. Universidade Presidente Antônio Carlos : Minas Gerais, 2004. Disponível em: < http://revistas.facecla.com.br/index.php/reinfo/article/view/146/38>. Acesso em: 30 dez. 2010.

SOMMERVILLE, I. Engenharia de Software. 6° Ed, São Paulo: Person Addison, 2007.

STANDISH GRORUP. CHAOS Summary 2011. Disponível em:

<http://standishgroup.com/newsroom/chaos_manifesto_2011.php>Acesso em 10/08/2011.

TELES, V.M. Um estudo de caso da adoção das práticas e valores do Extreme Programming. UFRJ – Universidade Federal do Rio de Janeiro , 2005.