AVALIAÇÃO DOS MÉTODOS DE DESENVOLVIMENTO TRADICIONAIS E ÁGEIS PARA APLICAÇÃO EM UMA EMPRESA DE *SOFTWARE*.

Lorival Warmeling Matias[1]

Resumo:

Este artigo apresenta métodos que podem ser aplicados no desenvolvimento de *software*. Inicialmente é feito um detalhamento de alguns dos métodos de desenvolvimento tradicionais e ágeis, e por último, como esse artigo é bibliográfico e comparativo, é apresentado um posicionamento sobre esses métodos e como aplicá-los em uma empresa de *software*.

Os resultados obtidos foram conclusivos, foi possível verificar a relação entre as metodologias ágeis: FDD, XP e SCRUM, os modelos tradicionais: sequencial linear, prototipagem, clássico, incremental, espiral, RUP e RAD, e a relação entre ágil e tradicional.

Foi possível verificar que indiferente da metodologia escolhida, o que deve ficar definido é o processo, suas relações com os papéis (pessoas) e os valores/práticas/regras relacionadas, desta forma, a empresa deve possuir essa metodologia bem definida e para isso deve escolher qual método melhor se adapta, para garantir a conclusão dos projetos de desenvolvimento de *software*.

Palavras-chave: Desenvolvimento de *Software*. Desenvolvimento Clássico. Desenvolvimento Ágil.

Introdução

O termo engenharia de *software* surgiu em 1968 em uma conferência organizada para discutir problemas no desenvolvimento de software, a experiência inicial do desenvolvimento informal mostrou-se ineficaz, com muitos atrasos e alto custo. Novas técnicas e métodos eram necessários para controlar as etapas de desenvolvimento, principalmente em sistemas grandes e complexos (SOMMERVILLE, 2007).

A abordagem inicial para elaboração desses métodos era produzir software de qualidade, dentro do prazo e com custos previsíveis. Métodos como a análise estruturada, desenvolvida na década de 70, tentaram identificar componentes funcionais básicos de um sistema, esse método foi complementado por métodos orientados a

[1] Dados do autor ao final do artigo

objetos nas décadas de 80 e 90. Todas essas abordagens foram integradas em uma única abordagem chamada *Unified Modeling Language* (UML) (SOMMERVILLE, 2007).

Esses métodos são utilizados até os dias atuais devido a sua vasta abordagem na condução dos levantamentos de requisitos, controles de projetos, controle sobre o desenvolvimento, controle sobre a manutenção e finalização dos projetos (SBROCCO; MACEDO, 2012).

Em contrapartida, como o desenvolvimento de *software* não se assemelha ao desenvolvimento de um produto físico, por exemplo, *hardware*, onde a matéria-prima é palpável e a dificuldade é em reproduzi-lo, com o *software*, o problema principal é no desenvolvimento, já que sua replicação tem praticamente custo zero. Pensando nisso, na década de 90, iniciaram-se estudos para definição de metodologias que focaram nos fatores humanos e no atendimento das expectativas dos clientes e não meramente em atender a burocracia do processo, essa técnica ficou conhecida como metodologia *leve*. Como a aplicação dessa técnica teve retorno positivo, em 2001, seus idealizadores promoveram um manifesto reunindo todas essas práticas que são conhecidas atualmente como métodos ágeis (BASSI FILHO, 2008).

Como o objetivo deste artigo é avaliar os métodos de desenvolvimento para aplicação em uma empresa de desenvolvimento de *software*, serão apresentados os métodos tradicionais e ágeis e por fim será feito uma comparação entre eles e como aplicá-los.

Métodos de desenvolvimento tradicionais

É fato que um processo de qualidade gere produtos de qualidade, seguindo essa premissa, os métodos tradicionais buscam uma organização dos processos para garantir a qualidade, prazo e custo aceitáveis.

A idéia de aprimoramento de processos veio originalmente do engenheiro William Edwards Deming, que trabalhou na indústria japonesa depois da segunda guerra mundial com o objetivo de aumentar a qualidade dos produtos, Deming

juntamente com Juran e Ishikawa introduziram o conceito de controle estatístico de processo naquele país.

Processos existem em empresas de uma só pessoa ou em multinacionais, esses processos diferem dependendo do gral de maturidade das empresas, dos tipos de produtos, do tamanho da empresa, etc. (SOMMERVILLE, 2007).

Existem quatro classes de processos de *software*:

1. **Processos informais:** não existe processo definido, a equipe de desenvolvimento escolhe que processo será utilizado. Os procedimentos e suas relações são definidos quando forem exigidos pela equipe de desenvolvimento.
2. **Processos gerenciados:** existe um modelo de processo definido, a programação e os relacionamentos dos procedimentos são seguidos.
3. **Processos metódicos:** é usado um ou mais métodos com beneficio do apoio da ferramenta CASE[2], esse método normalmente é utilizado em sistemas que passaram por reengenharia.
4. **Processos de aprimoramento:** são processos com objetivos de aprimoramento, possuem orçamentos para esse fim. Pode possuir uma avaliação quantitativa.

A seguir serão apresentados os métodos tradicionais de desenvolvimento de *software*.

- **Modelo sequencial linear**

  Normalmente chamado de ciclo de vida clássico ou modelo em cascata, é o método mais antigo e o mais amplamente utilizado das metodologias tradicionais (SBROCCO; MACEDO, 2012), ele sugere um sequenciamento no desenvolvimento passando por análise, projeto, codificação, teste e manutenção.

[2] Ferramenta CASE (do inglês *Computer-Aided Software Engineering*) é uma classificação que abrange todas ferramentas baseadas em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes.

A modelagem desse método inicia fazendo o levantamento de todos os requisitos para todas as partes do sistema e é necessária uma análise de alto nível e também um conhecimento estratégico sobre regras de negócio. Nesse caso o analista precisa ter tanto domínio sobre as informações do software quanto das funções necessárias, nesse método os requisitos do sistema são documentados e revistos pelo cliente.

O projeto é um processo de múltiplos passos que engloba os seguintes temas: estrutura de dados; arquitetura do software; representação da interface e detalhes procedimentais.

Inevitavelmente ao colocar o *software* em uso, são detectadas omissões ou erros nos requisitos levantados inicialmente, erros de programação ou ainda a necessidade de adicionar novas funcionalidades, desta forma, deve haver uma evolução do *software* para que o mesmo se torne útil. Outro aspecto desse método é que o programa executável somente é liberado quando todas as funcionalidades estiverem desenvolvidas, o que pode ocasionar divergências que serão vistas somente quando o sistema for liberado (SOMMERVILLE, 2007).

- **Modelo de prototipagem**

  É quando o cliente define os objetivos do sistema, mas não faz o detalhamento das entradas, processamento e saídas. Partes do software são desenvolvidas e já liberadas, permitindo ao cliente interagir com as entradas e saídas tornando assim a descoberta e/ou melhoria de requisitos mais fáceis.

  Esse modelo estabelece o ciclo de processos: ouvir o cliente, construir/revisar o protótipo e o cliente testar o protótipo, esse ciclo pode ser reiniciado tantas vezes quanto for necessário.

  Por mais que o cliente e o desenvolvedor gostem desse tipo de método, ele pode apresentar sérios problemas (SBROCCO; MACEDO, 2012):

  - Quando uma parte do software é liberada, o cliente acredita ser a versão completa, e normalmente reclama se for exposta a

necessidade de ajustes para garantir a qualidade ou a evolução das funcionalidades.

- A fim de liberar mais rapidamente, o desenvolvedor desconsidera pontos importantes, como por exemplo: linguagem de programação ou sistema operacional, ficando assim, limitado ao que conhece ou está facilmente disponível. E que pode, ao passar do tempo, tornar-se parte integral do sistema por desconhecimento dos pontos frágeis do desenvolvimento.

- **Modelo clássico**

Esse modelo surgiu em uma época diferente da atual onde imperava os *mainframes* e os chamados "terminais burros", onde os desenvolvimentos e alterações tinham alto custo, devido às ferramentas de desenvolvimento não possuírem, por exemplo, depuradores de código. Devido a isso o planejamento e a documentação eram amplamente utilizados.

Composto de etapas de fácil entendimento foi o primeiro processo de desenvolvimento de software publicado (SBROCCO; MACEDO, 2012), segue abaixo imagem demonstrando as etapas:

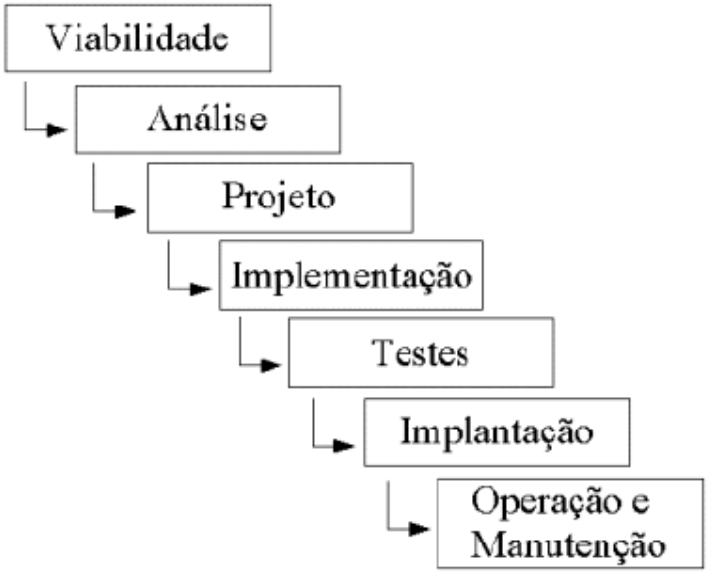

Figura 1 - Modelo clássico.

- **Desenvolvimento incremental**

Essa modelo mescla as técnicas de desenvolvimento sequencial linear com as de prototipagem, a idéia é separar o desenvolvimento em grupos de interações e fazer entregas ao cliente a cada integração, desta forma, ao final de cada interação é efeito uma avaliação das funcionalidades liberadas e da necessidade de alteração, permitindo a formação de um novo plano de desenvolvimento e assim deixando o *software* mais completo a cada integração. Segue abaixo imagem ilustrando esse modelo.

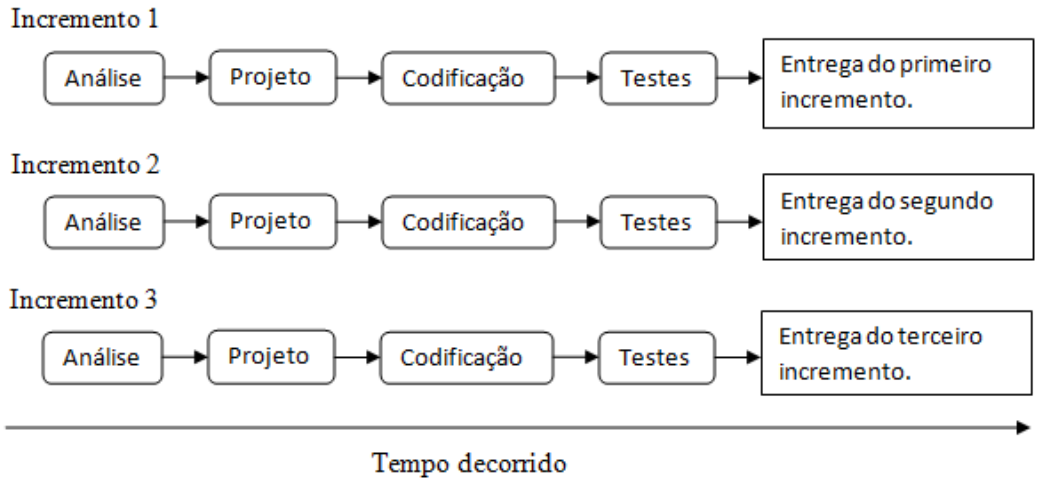

Figura 2 - Etapas e processos do modelo incremental.

Esse modelo tem como objetivo a entrega de um produto operacional em cada interação, são versões simplificadas do produto final que dispõem funcionalidades para os usuários. Esse modelo pode ser útil quando não se dispõe de mão de obra suficiente para todo o projeto, permitindo adicionar mais recursos a cada incremento entregue (SBROCCO; MACEDO, 2012).

No entanto, os incrementos devem ser pequenos (até 20 mil linhas de código) e a cada incremento deve ser entregue uma funcionalidade do sistema (SOMMERVILLE, 2007).

- **Desenvolvimento em espiral**

Esse modelo combina técnicas de prototipagem com aspectos controlados e sistemáticos do modelo sequencial linear. Esse modelo oferece suporte ao desenvolvimento de verões incrementais, como por exemplo, em versões iniciais pode ser prototipagem, em versões finais pode ser entregue funcionalidades ainda mais completas.

A estrutura do modelo espiral é distribuída em atividades normalmente conhecidas como regiões de tarefa, segundo Pressman (2001 *apud* Sbrocco e Macedo, 2012, p.66), existem de três a seis destas regiões de tarefas, conforme descrito a seguir:

1. **Comunicação com o cliente**: tarefas de comunicação entre desenvolvedor e cliente para a obtenção dos objetivos.
2. **Planejamento**: tarefas de definição de recursos, prazos, entre outras.
3. **Análise de risco**: tarefas de avaliação de riscos técnicos e gerenciais, por exemplo, se houver dúvida sobre os requisitos há a necessidade da construção de um protótipo do sistema.
4. **Desenvolvimento e validação**: tarefas para construir, testar, instalar e apoiar o usuário (documentação e treinamento).
5. **Avaliação pelo cliente**: tarefas para obter o posicionamento do cliente sobre as funcionalidades oferecidas e o modelo de implantação, oportunizando uma realimentação advinda do cliente e a liberação para o próximo incremento.

Esse modelo pode ser aplicado tanto para pequenos, quanto grandes projetos, cada tarefa pode definir um conjunto de atividades e essas variam dependendo do tamanho do projeto. Diferentemente do modelo clássico, o modelo espiral pode ser utilizando durante todo o clico de desenvolvimento do software, visto que sempre requer um novo planejamento após a realimentação da avaliação do cliente. Da mesma forma que ocorre com outros modelos clássicos, é fundamental a avaliação apurada dos ricos para garantir aos clientes que o modelo é sustentável, a imagem a seguir ilustra o desenvolvimento em espiral.

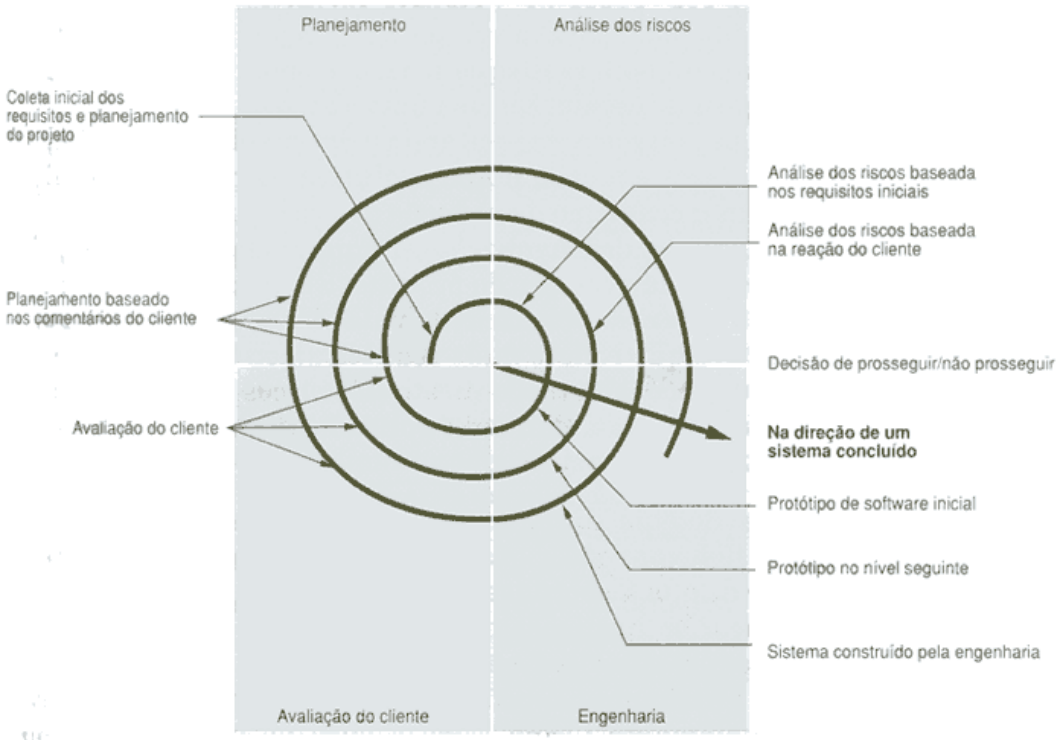

Figura 3 - Modelo espiral, segundo Sbrocco e Macedo (2012).

- **IBM *Rational Unified Process*® (RUP)**

Esse modelo foi inicialmente desenvolvido pela Rational Corporation e em seguida adquirido pela IBM. É uma plataforma de processos de desenvolvimento de *software* onde tarefas são distribuídas para garantir que a qualidade esteja de acordo com as necessidades dos clientes.

Esse modelo possui três perspectivas que demonstram como são relacionadas às atividades, a distribuição do tempo e as práticas durante a sua execução (SOMMERVILLE, 2007):

- **Estática - eixo vertical**: representa as atividades do fluxo de trabalho (*workflow*), são elas:
  1. **Modelagem de negócio:** processos de negócio são modelados com a ajuda dos casos de uso do UML.
  2. **Requisitos:** os requisitos do sistema são desenvolvidos com base nos casos de uso.

3. **Análise e projeto:** é criado o projeto utilizando os modelos de arquitetura, componente, objetos e sequência.
4. **Implementação:** são implementados os componentes e estruturados em subsistemas. É possível fazer a geração de código automaticamente com base nos modelos.
5. **Teste**: o teste é feito de modo interativo e estende-se até o término do desenvolvimento.
6. **Implantação**: cria-se a versão e faz-se a distribuição para os usuários finais.
7. **Gerenciamento de Configuração e Mudança:** essa atividade é referente as configurações e mudanças no sistema.
8. **Gerenciamento de Projeto:** atividade de apoio para o gerenciamento do desenvolvimento.
9. **Ambiente**: atividade que disponibiliza as ferramentas necessárias para o desenvolvimento.

- **Dinâmica - eixo horizontal**: representa o tempo e outros aspectos do projeto, é dividido em quatro fases sequenciais: inicial, elaboração, construção e transição. Cada fase representa um marco, ou seja, é a entrega e finalização de determinada parte do projeto e em seguida é feito uma avaliação para avançar pára a próxima fase.

- **Prática**: são papéis, atividades e artefatos a serem utilizados durante o processo.
  1. **Papel**: é uma definição abstrata de um conjunto de atividades, é normalmente realizado por um individuo ou um conjunto de indivíduos.
  2. **Atividade**: é uma unidade de trabalho de um indivíduo que responde por determinado papel.
  3. **Artefato**: é o produto do processo, e sempre, por mais insignificantes que seja, existe uma pessoa responsável. São pelos artefatos de entrada e saída que as atividades se comunicam.

Segue imagem ilustrando as fases do RUP®:

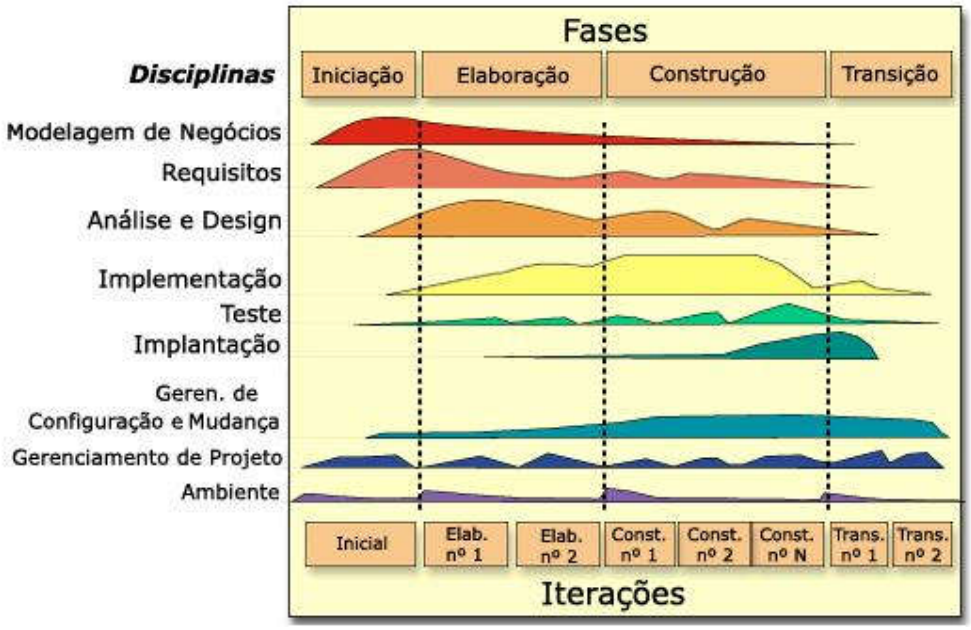

Figura 4 - Fases do RUP®

Fase de iniciação

Nessa fase e feito o levantamento do esforço, riscos, as exigências que devem ser analisadas antes de dar sequência no projeto. E importante também nesse momento fazer a formulação do escopo do projeto. Essa fase deve proporcionar um caminho no gerenciamento dos requisitos e no gerenciamento do projeto. Se o projeto não tiver aderência ao negócio, o mesmo pode ser cancelado nessa fase.

Fase de elaboração

Nesse momento é desenvolvido o entendimento sobre o projeto, é definido e validado a arquitetura a ser utilizada (execução, desenho e implementação da fase de construção). Ao final dessa fase tem-se uma descrição dos requisitos (casos do uso da UML), da arquitetura e um plano de desenvolvimento.

Fase de construção

Nessa fase é feito a elaboração do projeto, programação e os testes, desta forma, não é interessante adicionar processos ou ferramentas nesse momento,

para assim, garantir um processo estável. Nessa fase ainda existe um esforço, principalmente nas primeiras interações, de design e um contato muito próximo com o cliente e os demais *stakeholders*. Ao final dessa fase tem-se um sistema do *software* desenvolvido juntamente a documentação para ser liberara para os usuários.

Fase de transição

Nessa fase são finalizados todos os materiais que serão disponibilizados aos usuários finais, como também é feito os testes das funcionalidades diretamente na base do cliente. Nesse momento é criado um *release* do produto. Também são recebidos os *feedbacks* dos usuários para fazer os ajustes necessários. Idem a fase de construção, não se deve adicionar processos ou ferramentas, isso pode ser feito nos próximos projetos. Ao final dessa fase, tem-se o *software* documentado e funcionando na base operacional.

- **A abordagem RAD (*Rapid Application Development*)**

Desenvolvimento rápido de aplicação, ou simplesmente RAD, utiliza o método incremental e destaca o ciclo extremamente curto de desenvolvimento, esse método é parecido com o desenvolvimento linear, só que utilizando componentes. Se o entendimento dos requisitos e do objetivo do projeto e claro e simplificado, é possível utilizar esse método para construção de aplicações em curto espaço de tempo.

A abordagem RAD é dividida em cinco fases, conforme a seguir:

1. **Modelagem de negócio:** é definido que tipo de informação abrange o negócio, quem as gera, quem utiliza e quem processa essas informações.
2. **Modelagem de dados:** são definidos os objetos, suas características (atributos) e como se relacionam.
3. **Modelagem de processos:** é desenhado o processo com objetivo de atender a função do negócio e são definidos os procedimentos necessários para a manipulação dos objetos de dados.
4. **Geração da aplicação:** enfatiza a idéia de reutilização ou criação de componentes para futura utilização em linguagens não convencionais.

5. **Testes e entrega:** a idéia desse método é reutilização de componentes, que anteriormente já foram testados, desta forma, presumisse que não se perca muito tempo com os testes. E para que esse método não falhe é importante que desenvolvedor e cliente estejam engajados na premissa de desenvolvimento rápido.

Esse método é também chamado de CBSE (*Component-Based Software Engineering*), ele destaca a integração desses componentes em vez de construí-los do zero. Essa técnica é muito comum em integrações emergentes como Web Services (SOMMERVILLE, 2007).

Métodos ágeis de desenvolvimento de *software*

A idéia central do desenvolvimento ágil é redefinir continuamente as prioridades, mesmo demonstrando certo grau de informalidade, essas metodologias não deixaram de utilizar processos, ferramentas, documentos, negociação de contratos, etc. Só que as utilizam de forma reduzida e objetiva.

Entre as motivações para a utilização das metodologias ágeis podemos citar: priorização do código-fonte em vez de documentação, adaptabilidade em vez de predeterminação e são orientadas a pessoas e não a processos.

O conceito de adaptabilidade é muito interessante nos dias de hoje, visto que mesmo que fosse possível o levantamento consistente e completo de requisitos, existem fatores inerentes ao processo que podem simplesmente mudar a forma como determinada situação é tratada, por exemplo, alteração no modelo de negócio do cliente ou tendências de mercado, logo, é necessária uma característica mutável nesses requisitos.

Outro ponto interessante na relação entre desenvolvimento tradicional e ágil é quando um projeto de *software* é considerado bem-sucedido, no caso do tradicional é quando está dentro do prazo e custo, já no desenvolvimento ágil é quando o *software* agrega valor ao negócio do cliente, mais do que simplesmente o custo do desenvolvimento.

O conceito de gestão orientada a pessoas requer um time muito eficaz, que interajam entre si de forma eficiente. Já no processo tradicional isso não se faz necessário, visto que o conhecimento sobre o que deve ser feito encontra-se no processo e não nas pessoas, desta forma, pessoas são partes substituíveis do projeto (SBROCCO; MACEDO, 2012).

Com base na crescente necessidade de mais dinamismo das empresas e respostas mais rápidas as mudanças do mercado, a aplicação de metodologias ágeis acaba sendo mais eficientes, pois vem de encontro com essas necessidades.

A seguir serão apresentadas algumas das técnicas de desenvolvimento ágil.

- **Feature Driven Development (FDD)**

  A metodologia FDD foi criada em Singapura nos anos 90, mais especificamente entre 97 e 98, foi utilizada pela primeira vez para o desenvolvimento de um sistema bancário internacional, que já havia sido considerado inviável dentro do prazo determinado (SBROCCO; MACEDO, 2012).

  A modelagem FDD divide os papeis em três grandes grupos (BASSI FILHO, 2008):

  1. **Desenvolvimento da aplicação (papéis chave):** gerente de projetos, especialista em negócios, arquiteto chefe, gerente de desenvolvimento, programador chefe e dono da classe (programador).
  2. **Auxiliares ao desenvolvimento (papéis de apoio):** gerente de release, guru de linguagem, engenheiro de integração, toolsmith[3].
  3. **Opcionais ao desenvolvimento (papéis opcionais):** testadores, implantadores, escritores técnicos.

[3] Toolsmith é o responsável por criar ferramentas de apoio à equipe de desenvolvimento e testes. Pode atuar também como centralizador de conhecimento gerenciando sites intranet (Wiki). Normalmente é atribuição de programadores júnior ou recém graduados.

Os processos do FDD são compostos por cinco etapas que abrangem as fases de modelagem e implementação (BASSI FILHO, 2008), são elas:

1. **Desenvolvimento do modelo geral:** os especialistas em negócio chegam a um consenso sobre os objetivos, escopo e funcionalidades do projeto, criando assim uma lista de requisitos do sistema.
2. **Construir a lista de funcionalidades:** é criada a lista de funcionalidades com bases nos requisitos, contendo informações detalhadas do que deve ser desenvolvido.
3. **Planejar por funcionalidade:** é feito a ordenação das funcionalidades pelo programador chefe em pacotes para facilitar a liberação ao cliente, e a distribuição para os programadores (donos das classes).
4. **Modelar por funcionalidade:** os programadores chefes selecionam conjuntos de funcionalidades e montam pequenas equipes que trabalham em paralelo sobre seu comando.
5. **Implementar por funcionalidade:** é feito o desenvolvimento, testes e inspeções no código. Assim que as funcionalidades desenvolvidas forem validadas pelo programador chefe são integradas ao projeto principal e é dado inicio a outro conjunto de funcionalidades.

Essa metodologia também possui um conjunto de oito práticas recomendadas, que combinadas, aumentam a integração e o entendimento do projeto (BASSI FILHO, 2008), são elas: modelagem de domínio de objetos, desenvolvimento por funcionalidade, propriedades de classes, equipes por funcionalidade, inspeções, versões com regularidade, gerenciamento de configuração e relatórios de progresso.

- **Extreme Programming (XP)**

A metodologia XP foi criada por Kent Beck em 1996. Esse método foi resultado de um projeto crítico na empresa Chrysler, ela possuía um sistema de folha de pagamento com sérios problemas de custo e prazo, que após a aplicação das técnicas de Beck, o projeto que foi iniciado do zero, foi entregue com excelente qualidade e dentro do prazo (BASSI FILHO, 2008).

Considerada uma metodologia ágil popular, é utilizada quando os requisitos não são estáveis, ou seja, podem modificar a qualquer momento. Utiliza modelo incremental, onde o cliente já pode utilizar as funcionalidades conforme sãs liberadas e já oportuniza a implementação de novas funcionalidades tornando o processo de desenvolvimento mais rápido e eficaz, para projetos onde há necessidade de uma maior formalidade, esse método não é encorajado (SBROCCO; MACEDO, 2012).

Inicialmente é feito o levantamento de todos os requisitos transformando-os em histórias de usuários, em seguida os desenvolvedores estipulam o tempo de cada história, fazem o planejamento das interações, montam a próxima interação, fazem a identificação das tarefas, montam os casos de teste e fazem a implementação em duplas. Todos os dias é feito uma reunião informativa do que foi desenvolvido e do que será desenvolvido no dia seguinte (COSTA FILHO, 2006).

Esse método de desenvolvimento é sustentado por valores, práticas e papéis (SBROCCO; MACEDO, 2012), conforme descritos a seguir:

Valores

1. **Comunicação:** o XP enfatiza a comunicação entre os integrantes do projeto e principalmente um contato direto com o cliente, para assim, sanar todas as dúvidas ou implicações que possam surgir durante o andamento do projeto. É importante cuidar do clima e o bem-estar entre as pessoas, o clima deve ser extrovertido e profissional, sem desvios de atenção ou perda de foco. Pessoas rancorosas ou não comunicativas devem ser afastadas desse tipo de projeto.

2. **Simplicidade:** o importante é focar nos requisitos sem a necessidade de desenvolvimentos complexos que não agregam valor para o cliente, desta forma, essa metodologia reforça que todo desenvolvimento fora da necessidade devem ser desencorajado, para que não haja perda de recursos em funcionalidades não aplicadas. Se existir uma oportunidade

de um desenvolvimento mais complexo para melhorar o sistema, esse deve ser desenvolvido quando for necessidade do cliente. Por outro lado, a simplicidade não quer dizer desenvolver de qualquer jeito, sem levar em consideração recursos que poderão comprometer o sistema ou o processo do cliente no futuro.

3. **Coragem:** diferentemente do desenvolvimento tradicional, na metodologia XP, muitos requisitos podem aparecer durante o desenvolvimento, que nesse caso, os desenvolvedores devem ter a coragem de enfrentar processos complexos ou de alto risco, mesmo que inicialmente o cliente não deseje tais desenvolvimentos, visto que pode existir quebra de cultura ou paradigmas em que o cliente está inserido.

4. **Feedback:** esse valor enfatiza o feedback do cliente assim que cada parte do sistema é liberada, para que haja uma sinergia entre desenvolvimento e cliente.

5. **Respeito:** o respeito deve estar presente em todas as relações com o grupo e com os clientes, ações como: discriminar alguém que tenha posição inferior ou pela forma como trabalha, pode provocar reações negativas irreversíveis, desfocando os objetivos do projeto e tirando a sinergia do grupo.

Práticas

No primeiro livro de Beck foram propostas doze práticas, que transformam os valores em ações e segundo ele, devem ser seguidas (BASSI FILHO, 2008), são elas: versões pequenas, jogo do planejamento, *design* simples, programação em pares, testes, refatorações, integração continua, propriedade coletiva do código, ritmo sustentável, cliente presente, metáfora e padrões de código.

Uma consideração importante sobre a prática de teste, ou desenvolvimento guiado por teste, essa prática sugere que toda funcionalidade

tenha um teste automatizado (testes unitários), em algumas situações é necessário o teste por aceitação, que são aprovados pelo cliente.

Após algum tempo, grupos que utilizam essa metodologia entenderam que era necessário uma adaptação dessas práticas, pois não era possível sua aplicação integral, desta forma, essas práticas foram reformuladas e derem origem a dois grupos chamados: práticas primárias e práticas corolário.

As praticas primárias são descritas por treze práticas que ajudam na introdução da metodologia, são elas: integração contínua, programação em pares, trabalho energizado, desenvolvimento dirigido por testes, sentar junto, time completo, área de trabalho informativa, histórias, ciclo semanal, ciclo trimestral, folga, *build* ágil e *design* incremental.

As práticas corolário, divididas em onze, são de aplicação mais difícil, principalmente sem ter aplicado as práticas primárias, são elas: código compartilhado, envolvimento real com o cliente, implantação incremental, continuidade da equipe, redução da equipe, análise de causa inicial, código e testes, repositório de código unificado, implantação diária, contrato de escopo negociável e pague-pelo-uso.

Papéis

Os papéis são importantes dentro da equipe, porém não necessariamente todos precisam coexistir ou serem atribuídos as mesmas pessoas, Beck descreveu em seu livro os seguintes papéis (BASSI FILHO, 2008) e (SBROCCO; MACEDO, 2012):

1. **Gerente de projeto**: é quem administra o projeto, quem remove impedimentos, promove a comunicação necessária entre equipe e cliente.
2. **Analistas de negócio**: é quem faz o levantamento dos requisitos justamente com o cliente e ajuda-o a definir as histórias.
3. **Arquiteto**: analisa a estrutura do sistema e propõe alterações.
4. **Gerente de produto**: é quem escreve as histórias e estabelece os ciclos de desenvolvimento.

5. **Desenvolvedor**: essa função abrange tanto análise, quanto desenvolvimento, sua atribuição é estimar, desenvolver e testar as funcionalidades. Como a programação é em par, é importante que um dos dois tenha experiência, assim conforme os integrantes dos pares são trocados, as experiências serão distribuídas para toda a equipe. Caso os dois integrantes não tenham experiência, é necessário treinamentos antes do desenvolvimento.
6. **Analistas de teste**: é quem monta os casos de teste com a ajuda do cliente, essa função não pode ser atribuição do desenvolvedor.
7. **Redator técnico**: é quem monta a documentação do *software*.
8. **Projetista de interação**: avaliam a utilização do sistema pelo cliente.
9. **Coach**: normalmente é o programador mais experiente, é quem ajuda a equipe com as técnicas e metodologias do XP.
10. **Tracker**: é quem define os gráficos e informações de monitoramento do projeto e os mantém atualizados no ambiente de trabalho, é desta forma que toda a equipe acompanha o andamento do projeto.

- **SCRUM**

O termo "SCRUM" surgiu em 1986 após a publicação do artigo "*The new new product development game*" por Hirotaka Takeuchi e Ikujiro Nonaka na revista *Harvard Business Review* (PHAM; PHAN, 2011), e formalizada por Ken Schwaber (BASSI FILHO, 2008), que após analisar a tática do jogo *rugby* quando acontece algo inesperado ou para reiniciar o jogo, todos os participantes agem em conjunto, integrados e cada um em uma tarefa específica, ajudando assim, todo o grupo em um objetivo comum (COSTA FILHO, 2006).

O SCRUM é interativo e incremental, sugere equipes pequenas de até dez integrantes onde há necessidade constante de mudança de requisitos. Essa metodologia está dividida em três partes: papéis, artefatos e cerimônias, a estrutura de funcionamento é por ciclos, popularmente conhecida por *sprints*, a duração de cada *sprint* é normalmente de duas a quatro semanas (SBROCCO; MACEDO, 2012).

Papéis

São divididos em três:

1. ***Scrum Master*:** é o facilitador, promove a comunicação de todos os envolvidos, representa o cliente no projeto, remove impedimentos durante o desenvolvimento, ajuda o *Product Owner* a tomar as melhores decisões na condução do projeto e mantêm o foco da equipe. Esse papel tem obrigação de verificar se os processos da metodologia *Scrum* são seguidos.

2. ***Product Owner*:** ou simplesmente, dono do produto, é quem transforma necessidades em valores de negócio para o cliente, neste caso, representando-o. Tem a função de atualizar o *Product Backlog*, definir as prioridades com base no ROI (*Return of Investment*), definir a data de liberação do produto e aprovar ou reprovar os desenvolvimentos.

3. **Equipe *Scrum*:** é a equipe de desenvolvimento, normalmente possuí de cinco a nove pessoas, essas pessoas devem ter habilidades para analisar, desenvolver, testar, entre outras. Não deve haver uma distinção de níveis ou atribuições, a atenção deve estar voltada para o projeto, não pode ser trocado integrante durante a *sprint*. Suas tarefas principais são: estimar os tempos, dividir os itens da *Sprint Backlog* em tarefas, garantir qualidade, apresentar os resultados ao *Product Owner*.

Cerimônias

Ao longo da *sprint* existem pelo menos quatro cerimônias (reuniões):

1. ***Sprint Planning Meeting:*** reunião de planejamento da *sprint*, com duração de no máximo oito horas, é feito a seleção da lista do *Product Backlog* pelo *Product Owner* que define a meta, posteriormente a equipe define o *Sprint Backlog* (detalhamento em tarefas) para cumprir a meta estabelecida. Todos os envolvidos participam dessa reunião.

Como no Scrum não tem a definição de tempo das tarefas, e sim do tamanho delas, existem uma técnica que pode ser utilizada para medir o tamanho das tarefas, é o ***Planning Poker*** ou pôquer do planejamento.

Nessa técnica a equipe senta ao redor de uma mesa e o facilitador apresenta a funcionalidade que deve ser desenvolvida, após o entendimento de todos sobre a tarefa em questão, cada membro escolhe uma das cartas e coloca na mesa virada para baixo, assim que todos escolherem as cartas, elas são apresentadas e a equipe e essa deve chegar a um consenso entre os valores, em que cada integrante pode expor os motivos pela seleção e assim convencer o restante da equipe, assim que houver consenso o número selecionado é associado à funcionalidade. Essa mesma sequência é feita com todas as funcionalidades da *SprintBacklog*.

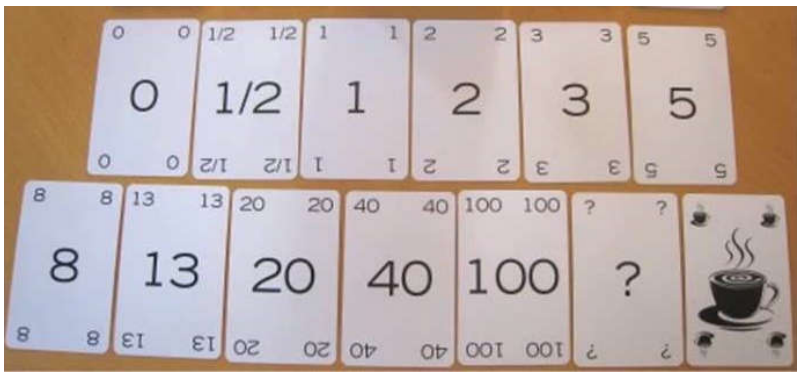

Figura 5 - Cartas do pôquer do planejamento.

Existem cartas especiais, como por exemplo, a zero que significa que a funcionalidade tem um tamanho insignificante e que levaria minutos para ser realizada, a carta com interrogação indica que o integrante não tem noção de quanto tempo levaria para a tarefa ser realizada e por último a xícara de café, que sugere ao grupo uma pausa, pois existe um cansaço que pode

comprometer o resultado da estimativa (SBROCCO; MACEDO, 2012).

2. ***Daily Meeting (*ou *Daily Scrum)*:** reuniões diárias com duração de até 15 minutos. Possibilita que cada membro da equipe exponha o trabalho realizado desde a última reunião, os problemas ocorridos e o que será feito até a próxima. Na reunião não devem ser discutidos temas técnicos, isso pode ser feito em outro momento. Participam dessa reunião a equipe *Scrum*, o *Scrum Master* como facilitador e outras pessoas, mas somente como ouvintes.

3. ***Sprint Review*:** reunião de revisão, com duração de até quatro horas é feita ao final da *sprint*, são apresentados os itens do *Product Backlog* desenvolvidos e suas funcionalidades ao *Product Owner* e esse pode aceitar ou não. Caso surjam novas funcionalidades essas deverão entrar no *Product Backlog* e novamente serem priorizadas. Participam dessa reunião a equipe *Scrum*, o *Scrum Master* e o *Product Owner*.

4. ***Sprint Retrospective*:** reunião retrospectiva, com duração de até três horas, deve acontecer após a reunião de revisão da *sprint*. Ela oportuniza a melhora continua apontando pontos positivos e negativos e possibilita que a equipe reflita sobre esses pontos a fim de melhorá-los nas próximas *sprints*. Participam dessa reunião a equipe *Scrum*, o *Scrum Master* e quando necessário o *Product Owner*.

Artefatos

Com o resultado das cerimônias, são criados três artefatos:

1. ***Product Backlog*:** é uma lista geral das funcionalidades necessárias para o desenvolvimento do projeto, deve ser mantido pelo *Product Owner*, ele deve ter acesso a fazer alterações nessas

funcionalidades, exceto se já estiverem na *sprint*. A ordenação deve ser feita com base no ROI (*Return of Investment*).

2. ***Sprint Backlog*:** lista de funcionalidades selecionadas na reunião de planejamento da *sprint* (*Sprint Planning Meeting*), essas funcionalidades são divididas em tarefas e cada tarefa deve ter a quantidade de horas para ser feita. A ordenação dessa lista também deve ser feita com base no ROI (*Return of Investment*). Para o acompanhamento das *Sprints* pode ser utilizado um quadro Kanban chamado de *Task Board*, ele é muito útil durante as reuniões, nele é possível criar uma divisão para o *product backlog* e outra para *sprint backlog* e controlar as funcionalidades de cada lista.

3. ***Burndown Chart*:** é um gráfico utilizado para demonstrar o andamento da *sprint*, na coluna vertical demonstra as funcionalidades ou horas de trabalho (depende da adaptação da equipe) e a coluna horizontal representa os dias, desta forma, é possível observar uma linha de funcionalidade ou horas durante os dias de duração da *sprint*. Segue exemplo abaixo do gráfico burndown:

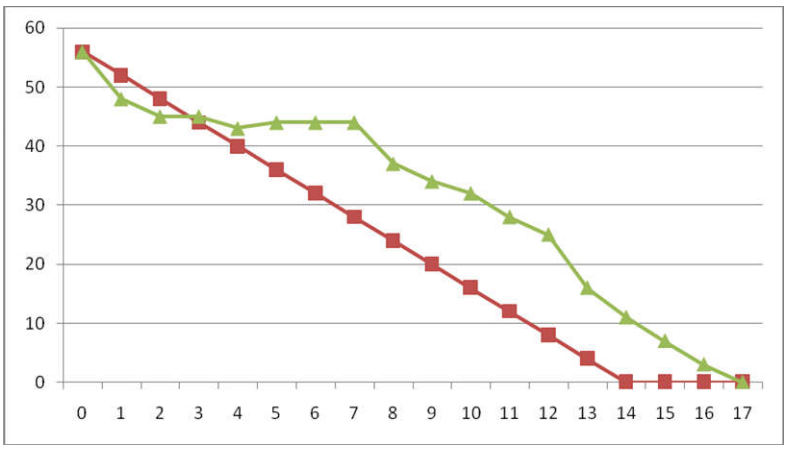

Figura 6 - Exemplo de Gráfico Burndown. Fonte: Autor.

Como pode ser verificado, quando a linha verde fica abaixo da linha vermelha significa que o desenvolvimento está adiantado,

caso fique acima, quer dizer que a entrega não está ocorrendo conforme o previsto. Com a utilização desse gráfico é possível analisar de forma rápida e objetiva como foi o andamento da *sprint*.

Ciclos de desenvolvimento (*Sprint*)

Como pode ser verificado na figura 7 a seguir, no início é elaborada a lista de requisitos (*Product Backlog*) pelos desenvolvedores e cliente. Após a elaboração dessa lista é definido o custo do projeto, os riscos, os integrantes da equipe, o *Scrum Master* (que é um dos integrantes da equipe), ferramentas de trabalho e também as datas para a entrega das funcionalidades priorizadas pelo cliente.

Após a elaboração do *Product Backlog,* a equipe deve definir o *Sprint Backlog*, que são as atividades que a equipe fará na próxima *sprint*. Nesse momento também é feito a divisão das responsabilidades com cada integrante da equipe.

Após a elaboração da *Sprint Backlog* a equipe deve iniciar as atividades de análise, desenvolvimento, testes e ao final deve apresentar ao cliente as funcionalidades desenvolvidas, para que ele possa dar *feedback*. Caso haja necessidade de correções, melhorias ou novas funcionalidades, essas devem ser adicionadas no *Product Backlog* e novamente ordenadas pelo *Product Owner*, dando início a próxima *sprint*.

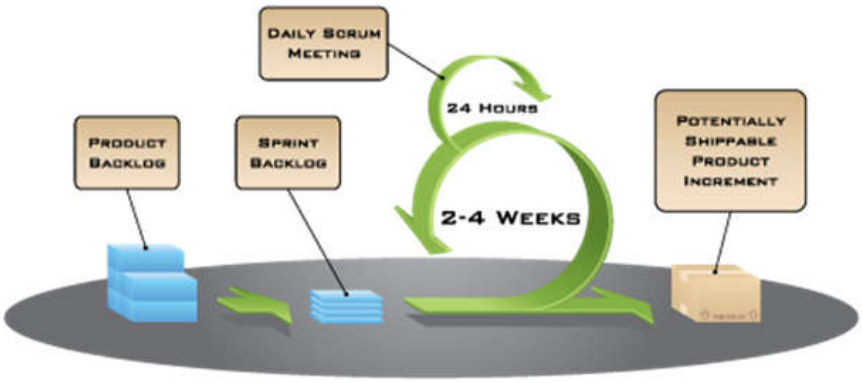

Figura 7 - Funcionamento da metodologia SCRUM.
Fonte: Sbrocco e Macedo (2012).

Dentro da *sprint* existem atividades de controle que gerenciam os itens não entregues, os problemas no processo e na codificação e montam-se estratégias para evitá-los (SBROCCO; MACEDO, 2012).

Considerações sobre os métodos e sua aplicação

Os métodos tradicionais de desenvolvimento de *software* concentram sua força em garantir que o processo seja totalmente controlado e documentado ao ponto de não depender exclusivamente de determinados indivíduos para o andamento do projeto. Só que ao mesmo tempo, tornam-se onerosos demais para empresas de pequeno e médio porte, que dispõem de equipes reduzidas e não conseguem alocar recursos em tarefas que não tenham retorno rápido e direto.

Um dos problemas dos métodos tradicionais é a necessidade de um levantamento completo dos requisitos, e que muitas vezes o cliente não tem domínio para deixar explícitas essas informações, o que gera consequentemente problemas no andamento do projeto sempre que algum destes requisitos é modificado ou adicionado.

A seguir são demonstrados os processos dos métodos tradicionais a fim de posicionar sobre o que cada metodologia abrange, segue:

Metodologia	**Processos (roteiro)**
Modelo sequencial linear (cascata) > Sequencial	Análise Projeto Codificação Teste

	Manutenção
Modelo de prototipagem > Sequencial por ciclo	Ouvir o cliente, Construir/Revisar o protótipo Cliente testar o protótipo
Modelo clássico > Sequencial	Viabilidade Análise Projeto Implementação Testes Implantação Operação e manutenção
Desenvolvimento incremental > Sequencial por incremento/ciclo	Análise Projeto Codificação Teste Entrega
Desenvolvimento em espiral > Sequencial por incremento/ciclo	Comunicação com o cliente Planejamento Análise de risco Desenvolvimento Validação Avaliação pelo cliente
IBM *Rational Unified Process*® (RUP) > Sequencial por fase/marco	Modelagem de negócio Requisitos Análise e projeto Implementação Teste Implantação Ger. Configuração e Mudança Gerenciamento de Projeto Ambiente
Rapid Application Development (RAD) > Sequencial por incremento (Orientando a componentes)	Modelagem de negócio Modelagem de dados Modelagem de processos Geração da aplicação Testes e entrega

Tabela 1 - Comparação das metodologias tradicionais. Fonte: Autor.

Com base nos processos de cada método tradicional, é feito a seguir um posicionamento de cada tipo de projeto relacionado a cada método, segue:

- Em projetos onde há um entendimento claro sobre o que deverá ser desenvolvido, não há riscos e não há necessidade de maior controle, o modelo sequencial linear pode ser útil.

- Caso o projeto tenha a mesma característica que o exposto no modelo sequencial linear e existe a característica de desenvolvimentos em componentes, ou seja, a reutilização de componentes prontos para aumentar a velocidade do desenvolvimento, o método indicado é o RAD.

- Se o projeto requer, além do exposto do método sequencial linear, um controle sobre a implantação no cliente, o modelo clássico pode ser utilizado.

- Se não existe um domínio claro sobre o que será desenvolvido, o cliente também não demonstra uma certeza quanto ao entendimento do processo, o modelo melhor aplicado nesse caso pode ser o de prototipagem, onde o cliente pode ter uma idéia de como será o desenvolvimento, antes mesmo de ser feito, ajudando-o no levantamento dos requisitos.

- Caso o projeto seja consideravelmente grande, ao ponto de ser necessário dividir o desenvolvimento em partes e que o cliente possa interagir com cada uma dessa partes liberadas, existem dois métodos tradicionais que podem ser utilizados, o incremental e o espiral.

- Caso o projeto seja grande, complexo, deva existir um controle extenso sobre o andamento do projeto, das atividades envolvidas, das fases e das pessoas o método mais indicado é o RUP.

Diferentemente dos tradicionais, os métodos ágeis são aplicadas em ambientes onde há maior necessidade de mudança de escopo, equipes menores e consequentemente menos documentação, a principal diferença entre o método tradicional e o ágil é a forma como o processo e conduzido.

A seguir apresento uma tabela com as três metodologias ágeis analisadas e dividindo-as em duas partes, segue:

Metodologia	**Processos (roteiro)**	**Papéis (pessoas)**
FDD	Desenvolvimento do modelo geral Construir a lista Planejamento	-principais: Gerente de Projetos Especialista em Negócio

	Modelagem Desenvolvimento	Arquiteto Chefe Gerente de Desenvolvimento Programador Chefe Desenvolvedor (incluí teste) -apoio: Gerente de Release Guru Linguagem Engenheiro de Integração Toolsmith
XP	Levantamento Requisitos Construir a lista Planejamento Modelagem Desenvolvimento Apresentar ao cliente	- principais: Gerente de Projetos Analista Negócio Arquiteto Gerente de Produto Desenvolvedor Analistas de Teste Redator Técnico Projetista de Interação -apoio: Coach Tracker
SCRUM	Levantamento Requisitos (PB) Construir a lista (SB) Planejamento (S) Modelagem (S) Desenvolvimento (S) Apresentar ao cliente (S) PB=*Product Backlog* SB=*Sprint Backlog* S=*Sprint*	Gerente de Projetos (PO) Analista Negócio (SM) Arquiteto (ES) Gerente de Produto (ES) Desenvolvedor (ES) Analistas de Teste (ES) Redator Técnico (ES) Projetista Interação (ES) PO=*Product Owner* SM=*Scrum Master* ES=Equipe *Scrum*

Tabela 2 - Comparação das metodologias ágeis. Fonte: Autor.

A diferença entre as metodologias ágeis é a forma de gerenciar o processo e a relação entre as pessoas, por exemplo, no método FDD, todo o processo é gerenciado por funcionalidade, desta forma, atende tanto a pequenas, médias quanto grandes equipes.

Já o método XP, atende bem a equipes pequenas ou médias, visto de sua característica onde os desenvolvedores têm contato direto com o cliente e a ferramenta de feedback é constantemente utilizada.

No método SCRUM, existe um controle maior sobre como os papéis e os processos se relacionam, esse método visa detalhar como deve ser o andamento do desenvolvimento em um ambiente onde podem surgir necessidades novas a todo o momento.

Fica evidente após essa comparação que em ambas as metodologias as etapas do processo de desenvolvimento são bem semelhantes, até porque são necessárias para que o *software* seja desenvolvido e entregue ao cliente, o que diferencia uma metodologia de outra é a forma como os processos são gerenciados, o envolvimento das pessoas no projeto e o tipo de negócio que o *software* está proposto a atender.

Considerações finais

Algumas metodologias tratam os processos com nomes diferentes, outras simplesmente não destacam, mas acabam comentando em alguma parte das práticas ou valores ou simplesmente não comentam, mas inevitavelmente é necessário passar por todos os processos, mesmo que de forma superficial, dependendo do tipo de projeto.

As diferenças entre a metodologia tradicional e ágil são apresentadas a seguir:

Tradicional	**Ágil**
Focado no processo.	Focado no negócio.
Baixa adaptabilidade a mudança.	Auxílio a mudanças.
As normas e regras são impostas.	Os padrões são definidos de forma adaptativa com base em projetos anteriores.
Contratos formais completos.	Contratos mínimos e definidos de modo a resguardar os interesses da equipe de desenvolvimento e do cliente.
A equipe é impositiva no relacionamento com o cliente.	A relação com o cliente é aberta, podendo ter auxilio constante no andamento do projeto.
Equipes maiores e muitas vezes geograficamente divididas.	Equipes pequenas, de até dez indivíduos.

Aumento expressivo de custos na finalização dos projetos devido há necessidade de alteração de contratos e documentação.	Como o desenvolvimento é incremental, não são atribuídos custos ao projeto.
Recomendado para projetos críticos. Por exemplo, onde envolve risco de morte.	Não recomendado para projetos críticos, justamente porque a necessidade de alteração é aceita com naturalidade por todos os envolvidos.

Tabela 3 - Diferenças entre a metodologia tradicional e ágil,
Fonte: Sbrocco e Macedo (2012).

A aplicação de métodos de desenvolvimento com certeza é necessária em qualquer empresa de desenvolvimento de *software*. Fica claro que o desenvolvimento com base em uma metodologia possibilita um controla maior sobre a forma como os processos e/ou pessoas são gerenciados. A não aplicação de uma metodologia traz diversos problemas ao *software*, que inevitavelmente pode vir a sucumbir.

Desta forma, a conclusão desse artigo é que em todos os casos, é necessário que o processo esteja definido com base na metodologia que melhor se adaptada a empresa.

Referências

BASSI FILHO, Dairton Luiz. **Experiências com desenvolvimento ágil.** 2008. 151 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/451 34/tde-06072008-20351 5/ptbr.php>. Acesso em: 06 abr. 2015.

COSTA FILHO, Edes Garcia da. **Integração de padrões organizacionais e de processo ao método ágil Scrum.** 2006. 119 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Computação, Universidade Federal de São Carlos, São Carlos, 2006. Disponível em:

<http://www.bdtd.ufscar.br/htdocs/tedeSimplificado/tde_busca/arquivo.php?codArquivo=1112>. Acesso em: 30 set. 2015.

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar de. **Metodologias ágeis**: Engenharia de software sob medida. São Paulo: Érica, 2012. 254 p. (ISBN 978-85-365-0398-1).

SOMMERVILLE, Ian. **Engenharia de software.** 8. ed. São Paulo: Pearson Addison-wesley, 2007. 552 p. (ISBN 978-85-88639-28-7). Tradução de: Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa; Revisão técnica: Kechi Kirama.

PHAM, Andrew; PHAN, Phuong-van. **Scrum em ação:** gerenciamento e desenvolvimento Ágil de projetos de software. São Paulo: Novatec, 2011. 287 p. (ISBN 978-85-7522-285-0). Tradução de: Edgard B. Damiani.

Dados do autor

Nome: Lorival Warmeling Matias
Instituição: Unibave - Centro Universitário Barriga Verde
Formação e/ou função: Graduado em Sistemas de Informação
Contato: lorival77@gmail.com