UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Filipe Sousa Costa

Fusão de um processo tradicional de desenvolvimento de software com uma metodologia ágil: um estudo de caso.

Uberlândia, Brasil

2017

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Filipe Sousa Costa

Fusão de um processo tradicional de desenvolvimento de software com uma metodologia ágil: um estudo de caso.

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. William Chaves de Souza Carvalho

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2017

Filipe Sousa Costa

Fusão de um processo tradicional de desenvolvimento de software com uma metodologia ágil: um estudo de caso.

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 20 de dezembro de 2017:

Prof. Dr. William Chaves de Souza Carvalho
Orientador

Ana Beatriz dos Santos Carvalho

Pedro Moisés de Sousa

Uberlândia, Brasil
2017

Dedico em primeiro lugar a Deus, pela sua importância em minha fé e vida durante esta caminhada. Espero que o conhecimento adquirido ao longo dessas anos possa trazer diversos benefícios para a sociedade cumprindo meu papel dentro do propósito d'Ele para minha vida. Aos meus pais Eder Costa e Odália Costa pelo apoio nas mais diversas situações no decorrer desses anos, aos meus amigos da faculdade, aos professores e por último, não menos importante, ao meu orientador Prof. Dr. William Chaves de Souza Carvalho por toda sua receptividade e amparo.

Agradecimentos

Agradeço em primeiro lugar ao Autor da Existência, Aquele que permite que todas as coisas se concretizem, nosso único e verdadeiro Deus. Em segundo lugar agradeço a todas as pessoas que diretamente ou indiretamente, contribuíram para a construção dos meus valores: meus pais e todos os que compartilharam um pouco do que sabem comigo e com os meus amigos nesta vida acadêmica.

"Seja você quem for, seja qual for a posição social que você tenha na vida, a mais alta ou a mais baixa, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá."

Ayrton Senna

Resumo

Este trabalho visa analisar as metodologias híbridas de desenvolvimento de software fundamentada em publicações de autores que já colaboraram com a comunidade científica e acadêmica por meio de pesquisa e estudos de caso do tema abordado. Primeiramente, discorre-se sobre a engenharia de software e o processo de software, posteriormente, é exposto um encadeamento de acontecimentos, em ordem cronológica, iniciando-se nas soluções primárias aspiradas para combater à crise de software até o advento das metodologias híbridas. Em seguida, é apresentada minuciosamente algumas metodologias de desenvolvimento de software mais notáveis, tanto as tradicionais, como as ágeis. Concluído isso o uso dos métodos híbridos é retratado, e para finalizar, o trabalho é terminado com um estudo de caso da metodologia híbrida.

Palavras-chave: Engenharia de Software. Metodologia Híbrida. Desenvolvimento de Software.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

RUP	Rational Unified Process
POO	Programação Orientada a Objetos
RAD	Rapid Application Development
UML	Unified Model Language
XP	eXtreme Programming
TDD	Test Driven Development
CMMI	Capability Maturity Model Integration
SEI	Software Engineering Institute
MHDCU	Metodologia Híbrida de Desenvolvimento Centrado no Utilizador

Sumário

1 Introdução

Um método de desenvolvimento de software é um conjunto de processos que contribuem na fabricação de um software e o efeito desses conjuntos é um produto que expõe a maneira como todo o desenvolvimento foi conduzido.

O software tornou-se o elemento chave na evolução de sistemas e produtos baseados em computador, e uma das tecnologias mais importantes de todo mundo (PRESSMAN, 2006).

A medida que a importância do software cresceu, a comunidade de software tem continuamente tentando desenvolver tecnologias quer tornem mais fácil, mais rápido e menos dispendioso construir e manter programas de computadores de alta qualidade. (PRESSMAN, 2006)

O desenvolvimento de software, na década de 70, ficou conhecido pela *Crise do Software* (PRESSMAN, 2006), nessa época não havia nenhuma ordem, projeto, por parte das empresas fabricantes de software quanto a sua implementação, isto resultava em muitas adversidades para as empresas, como por exemplo, alto custo, prazo extrapolado para entrega, insatisfação por parte cliente pelo fato de não ter suas expectativas atendidas, dificuldades futuras em manutenções e etc. Em meio a esta conjuntura emergiu a concepção de processos planejados e estruturados para o desenvolvimento de software, buscando atender as deficiências identificadas.

De acordo com (MOTA; LIMA; ROMANO, 2011), a Engenharia de Software surgiu para solucionar os problemas da Crise de Software. Tendo como base os modelos industriais uma nova metodologia de desenvolvimento foi elaborada de modo que se organizava de uma forma profissional o modo de desenvolvimento de softwares, separando-os em etapas, juntamente com uma série de documentos para especificar ao cliente o software que seria desenvolvido, buscando sempre um produto final de alta qualidade.

Atualmente é possível separar as metodologias de desenvolvimento de software em duas partes, a *metodologia tradicional* (pesada) e a *metodologia ágil*.

A metodologia tradicional surgiu nas circunstâncias de desenvolvimento de software bem discrepante do atual, naquela época era baseado apenas em mainframes e terminais burros (ROYCE, 1987). Todo o processo de um software era esboçado e documentado em um momento anterior a implementação, isto se fazia necessário devido ao alto custos para realizar modificações no código devido as limitações tecnológicas da época. Podemos citar como exemplo de métodos tradicionais os modelos cascata, prototipação, RAD e espiral.

(MAINART; SANTOS, 2010) afirma que o processo de desenvolvimento de software é bastante mutável, pois muitas vezes há o surgimento de novos requisitos, funcionais ou não funcionais por parte do cliente e quando a metodologia tradicional é adotada é preciso modificar todo o documento e o produto, constantemente, intercorrendo o não cumprimento nos prazos estipulados no inicio do projeto.

Constatando as dificuldades encontradas na aplicação de metodologias tradicionais, um grupo de coordenadores de projeto experientes empregaram procedimentos de desenvolvimento de software contrários aos principais conceitos da metodologia tradicional. Em 2001, esse grupo criou o Manisfesto Ágil e o resultado final dessa reunião deu-se os primórdios a novas metodologias de desenvolvimento de software (BASSI FILHO, 2008).

As metodologias ágeis surgiu em meio as dificuldades enfrentadas pelas pequenas e médias empresas fabricantes de software, pois o tempo empregado para determinar como o software seria desenvolvido (documentação) era maior do que o gasto no desenvolvimento do programa (SOMMERVILLE, 2007). Esta metodologia é baseada no código e aceita constantes mudanças nos requisitos sem preocupar-se demasiadamente com a documentação, nela foi dada uma autonomia maior aos programadores para se concentrarem no desenvolvimento do software. Elas buscam tirar o foco do processo e se concentrar mais no produto. Desta maneira, as metodologias ágeis tencionam a renunciar ou alterar as etapas e a maneira como os envolvidos executam suas atividades (BASSI FILHO, 2008). Podemos citar como exemplo de metodologias ágeis o SCRUM e XP.

As duas metodologias possuem seus pontos fortes e fracos. A metodologia tradicional é mais indicada para grandes projetos e de alto risco, ela se fundamenta na análise e projeto, com muita documentação, em contra partida, é prolongada qualquer alteração. Já os métodos ágeis é mais indicado para pequenos projetos, de baixo risco e para equipes de tamanho pequeno. Ela é alicerçada em código, maleável a alterações de requisitos, apesar disso, ela é frágil no que tange a contrato e de documentos (CARVALHO et al., 2012).

No meio de diversos debates sobre qual era mais recomendado, qual era melhor, surgiu os métodos híbridos, sustentando que abordagens seguidas a risca não agrada da maneira correta, aos costumes predominantes das empresas desenvolvedoras de software. O entendimento da metodologia híbrida, que é a fusão de um processo tradicional de desenvolvimento de software com uma metodologia ágil, vem com o intento de resolver esses questionamentos (MOTA; LIMA; ROMANO, 2011).

1.1 Objetivos

Visando contribuir com a expansão do conhecimento sobre o tema, esta pesquisa objetivará investigar algumas opções de fusão entre o mundo de metodologias tradicionais e o mundo de metodologias ágeis.

1.2 Método

Para tal será feito um estudo de caso em que serão comparados os resultados, segundo um conjunto de parâmetros definidos, de múltiplos projetos de uma fábrica de software avaliando ao término os impactos gerados na sua produtividade após a adoção da proposta combinada.

2 Fundamentos

2.1 Introdução

Este capítulo tem o objetivo de mostrar uma visão geral dos conceitos relacionados às áreas de conhecimento envolvidas neste trabalho, incluindo os processos Scrum e RUP, assim como o conceitos substanciais para o compreendimento da proposta de combinação Scrum-RUP.

2.2 Engenharia de Software

A engenharia de software é uma matéria da engenharia que abrange todos os conceitos da produção de um software, partindo dos estágios iniciais de especificação do sistema e caminhando até a sua manutenção, quando este já está em execução (SOMMERVILLE et al., 2003).

(PRESSMAN, 1995) alude a engenharia de software através de camadas (*tecnologia em camadas*). Qualquer atividade de engenharia de software tem de ser alicerçada com a camada de qualidade, em cima desta camada aparece os processo e posteriormente, os métodos e, acima destes, as ferramentas. No decorrer da história da engenharia de software foi concebido instrumentos computadorizados para ajudar o desenvolvimento. Houve um significativo progresso com essas iniciativas, conquanto existe ainda uma indispensabilidade de mão-de-obra humana.

Ao longo das últimas décadas foram elaboradas diversas metodologias para o desenvolvimento de software, conquanto existem atividades comuns a todas elas:

- *Especificação de software*: A funcionalidade e as restrições sobre suas operações devem ser estabelecidas.
- *Projeto e implementação*: Um software que siga à especificação deve ser fabricado.
- *Validação*: O software deve ser validado para assegurar que ele satisfaça o cliente.
- *Evolução*: O software deve evoluir para satisfazer às necessidades de mudanças do cliente.

(SOMMERVILLE, 2007)

2.3 Processos de software

Processos de software tem o objeto de supervisionar e coordenar projetos de desenvolvimento de softwares verdadeiros. Segundo (FILHO, 2003) um processo é um agrupamento de passos mais ou menos ordenados, composto por atividades, métodos, práticas e transformações, que almejam alcançar uma meta. Na engenharia de software, processos pode ser definido para procedimento como manutenção, desenvolvimento, compra e contratação de software.

Um processo de software pode ser conceituado também como um aglomerado de atividades e resultados correlacionados que coordenam à produção de um produto de software (SOMMERVILLE et al., 2003). Processos de software são grandes e dependem do entendimento do ser humano assim como em quaisquer processo intelectual. Dentro deste raciocínio, é identificado diversos processos de software, conquanto nenhum é ideal, pois foram elaborados seguindo às necessidades de cada empresa e de maneiras substancialmente diferentes.

De uma forma geral o desenvolvimento de um software envolve as seguintes etapas:

1. *Planejamento*: aqui o gerente de projetos procura uma base para fornecer estimativas coerentes de recursos técnicos, financeiros e de prazo para o desenvolver do projeto. Quando disponibilizado escopo do projeto, uma concepção de desenvolvimento deve ser criada, ou seja, um plano de projeto deve ser montado dispondo o processo a ser usado no desenvolvimento do software. Com o avanço do projeto, este planejamento tem de ser constantemente atualizado e detalhado, assim ao final de cada etapa do desenvolvimento, o planejamento por geral deve ser revisado e o planejamento da etapa posteriormente detalhado. Este planejamento e esta supervisão integram o processo de gerência de projeto.

2. *Análise e Especificação de Requisitos*: aqui há um crescimento no processo de levantamento de requisitos. O escopo elaborado anteriormente tem de ser aprimorado e então identificados os requisitos funcionais e não-funcionais. O engenheiro de software tem de assimilar o domínio do problema, as suas funcionalidades e seus possíveis comportamentos para entender a natureza do software a ser desenvolvido. Quando identificados os seus requisitos funcionais e os não-funcionais, estes tem de ser modelados, avaliados e documentados.

3. *Projeto*: aqui ocorre a junção dos requisitos tecnológicos aos requisitos do sistema, concebidos na etapa anterior. Podemos resumir em duas grandes etapas: projeto de arquitetura do sistema e projeto detalhado. A primeira etapa busca-se indicar a arquitetura geral do software, dispondo dos modelos concebidos na fase anterior, nesta arquitetura é necessário detalhar a estrutura de nível superior do desenvolvi-

mento e detectar seus principais componentes. Sucessivamente, na segunda etapa, deve-se detalhar cada componente identificado, esses mesmo devem ser posteriormente aperfeiçoados em alto nível de detalhamento, para que possam ser codificado e realizados os devidos testes.

4. *Implementação*: aqui entre-se na etapa da codificação, onde ocorre a implementação do projeto em linguagem de máquina.

5. *Testes*: aqui podemos dividir em duas sub-etapas: teste de unidade e teste de integração. Começa-se pelos testes de unidade, após sua verificação os seus resultados devem ser documentados, depois os componentes devem ser reunidos entre si até a concepção de um projeto homogêneo e então verificar quando estes estão integrados entre si.

6. *Entrega e Implantação*: após o teste, o produto deve ser colocado no mercado. O objetivo desta etapa e fazer com que o software atenda aos requisitos dos usuários, este processo é feito através da instalação do software e validação do mesmo.

7. *Operação*: aqui o software é operado pelos usuários no ambiente de trabalho.

8. *Manutenção ou evolução*: nesta etapa é quando o software é modificado visando reparar, melhorar ou acrescentar algum novo requisito. Aqui ocorre um ciclo iterativo com todas as fases anteriores do projeto.

2.4 Um pouco sobre metodologias

Mirando atingir o conceito de metodologia de desenvolvimento híbrido é vital termos o entendimento de como funciona seus progenitores, as metodologias tradicionais e ágeis.

2.4.1 Modelos de ciclo de vida

O início para a definição da arquitetura de um processo de desenvolvimento de software é a escolha de um modelo de ciclo de vida. É resumido a seguir uma discussão dos mais notáveis modelos de ciclo de vida, conforme apresentado em (PÁDUA FILHO, 2009), em que é usado o Diagrama de Atividades da UML (RUMBAUGH; JACOBSON; BOOCH, 2004) para relatar o *workflow* de subprocessos sugerido por cada modelo de ciclo de vida.

O modelo cascata é o mais conhecido destes modelos de ciclo de vida, nele os principais subprocessos são processados em sequência, conforme exposto na Figura 1. A simplicidade de gestão com pontos de controle bem definidos é uma vantagem deste modelo, entretanto é o seu ciclo de vida é bastante enrijecido que requer as atividades de requisitos, análise e projetos sejam executadas perfeitamente, caso contrário o risco de identificar obstáculos na correção de erros nas fases derradeiras são bem elevadas. Uma nuance deste modelo é o modelo *cascada com realimentação*, no qual é liberada que um subprocesso retorne ao anterior, conforme exposto na Figura 1.

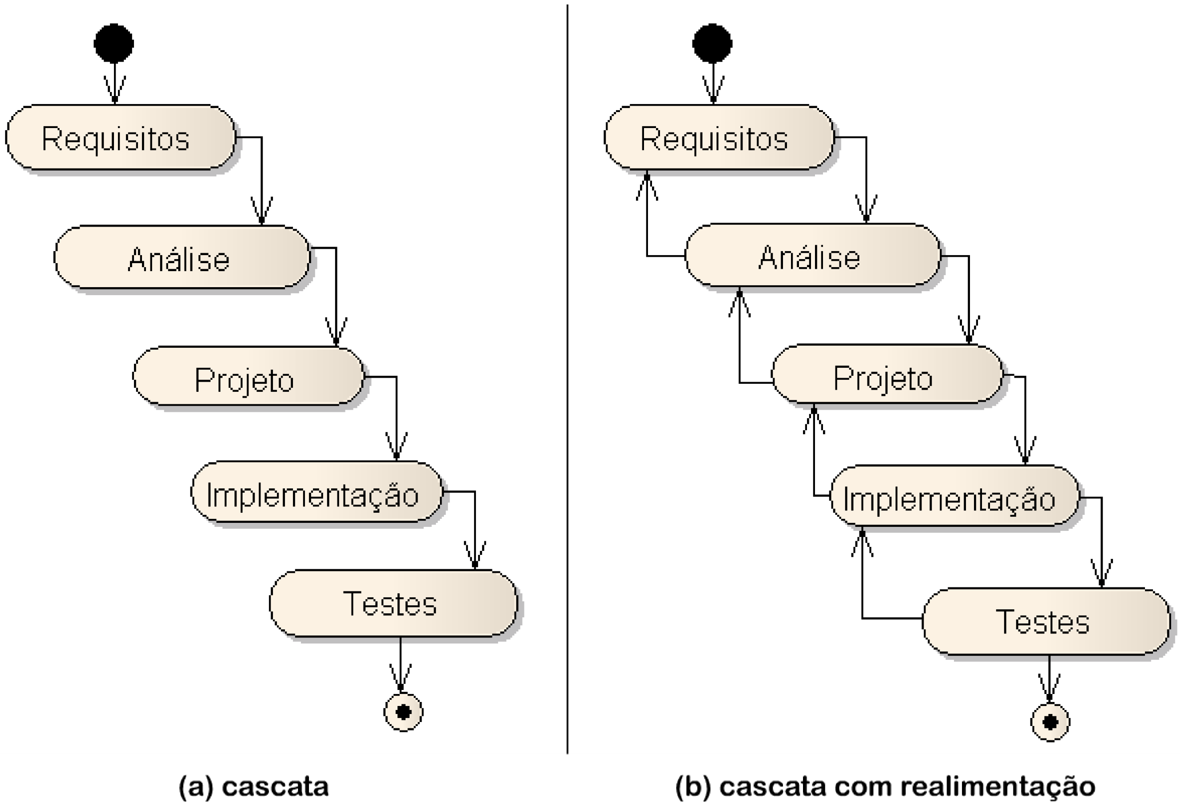

Figura 1 – Ciclo de vida em cascata e cascata com realimentação

Fonte: (PÁDUA FILHO, 2009)

Este ciclo de vida em espiral sugere que o software é desenvolvidos em uma séria de iterações (Figura 2). Todos os subprocessos passam por cada iteração e no final, uma *release* parcial do produto é disponibilizada para apreciação.

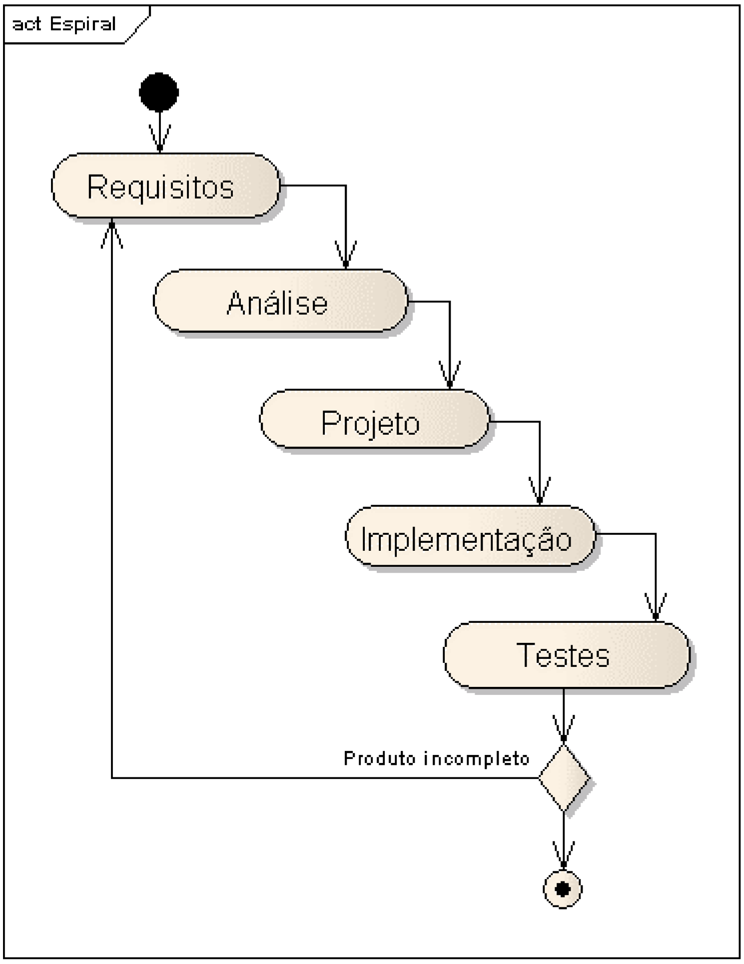

Figura 2 – Ciclo de vida em espiral

Fonte: (PÁDUA FILHO, 2009)

Na Figura 3 vemos o modelo de entrega evolutiva e então podemos afirmar que é admissível outros modelos de ciclo de vida mistos (PÁDUA FILHO, 2009).

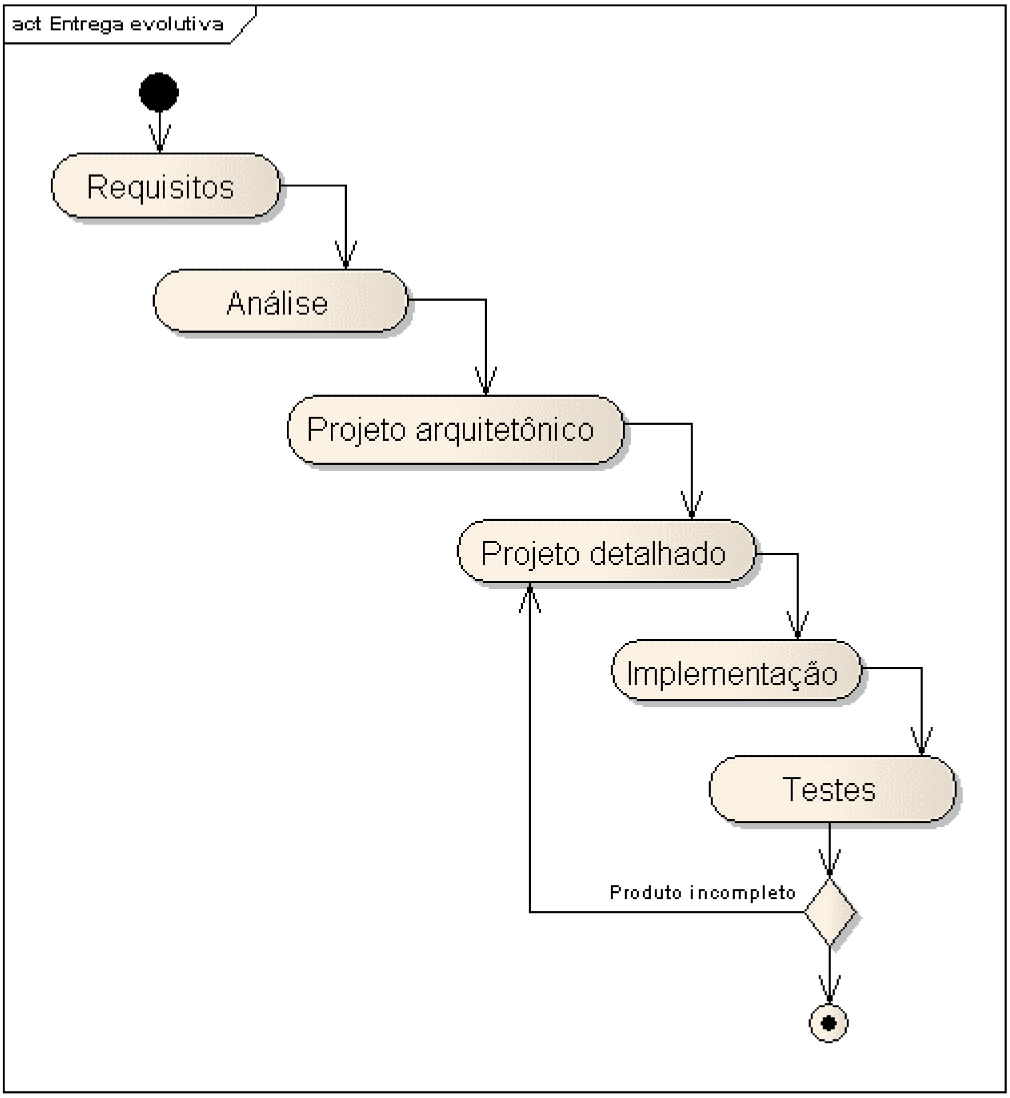

Figura 3 – Ciclo de vida entrega evolutiva

Fonte: (PÁDUA FILHO, 2009)

Nesta versáteis, os subprocessos de requisitos, análise e projeto arquitetônico são executados em cascata, buscando gerar uma arquitetura de sistema robusta, com a intenção de consolidar as principais decisões de projeto nas etapas iniciais e propiciar que o desenvolvimento daqui para frente não vivencie esforços drásticos de retrabalho.

Conquanto, de modo efetivo, estes modelos não são usados pelos processos de desenvolvimento de software da maneira como são mostrados. Verdadeiramente, em todo o tempo há uma fase de iniciação, este em que é realizada uma definição mínima dos requisitos do produto, para delinear seu escopo, e uma etapa de transição, que é quando o produto completo é implantada em seu ambiente final. Este modelo é nomeado de quase-espiral (Figura 4) (PÁDUA FILHO, 2009).

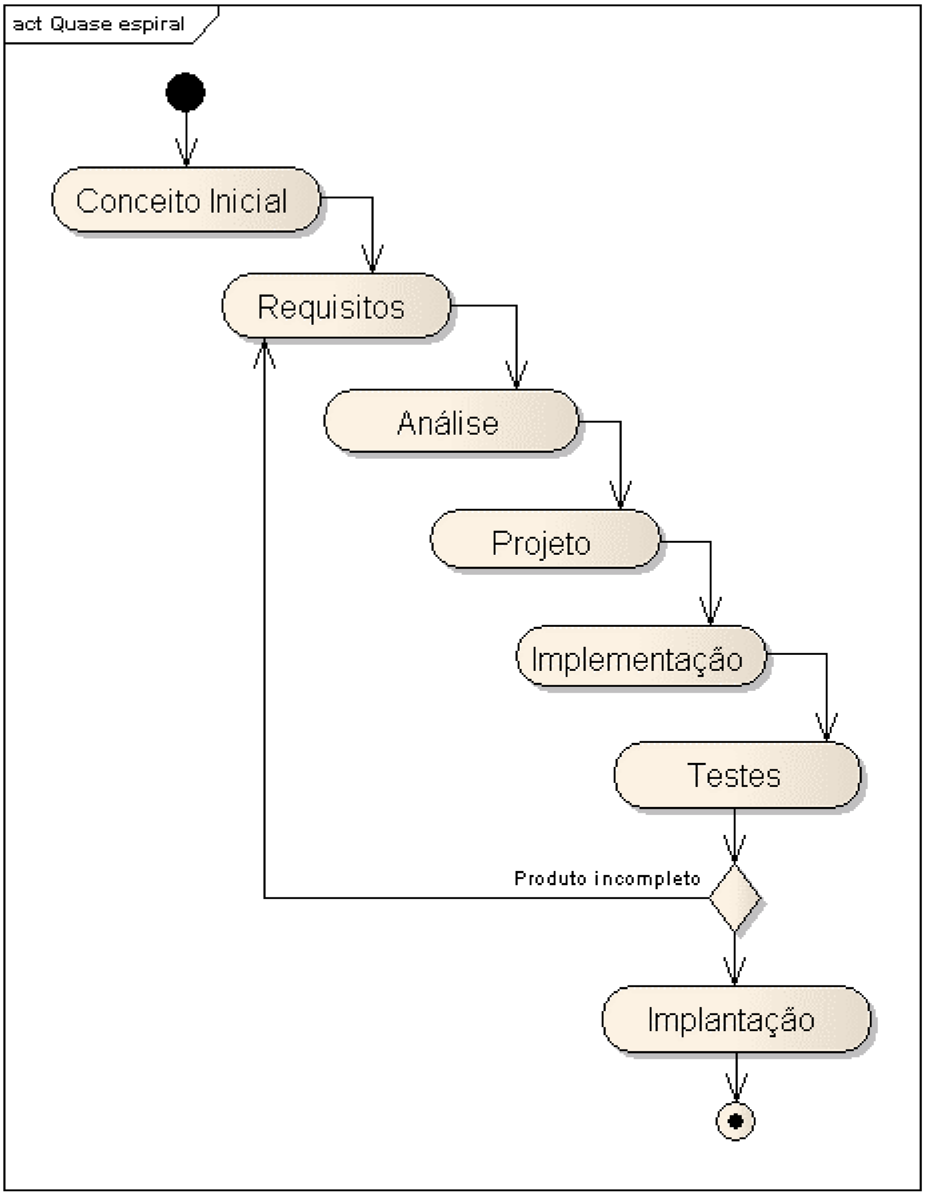

Figura 4 – Ciclo de vida quase-espiral

Fonte: (PÁDUA FILHO, 2009)

Existe outros modelos também, como por exemplo o *tome-boxed* (tempo delimitado, no qual um release é exatamente o que consegue-se realizar dentro de um prazo decidido.

2.4.2 Metodologia tradicional de desenvolvimento de software

Não existe um nome unificado para designar a metodologia tradicional de desenvolvimento de software, os mais conhecidos na literatura são os termos "tradicional", "dirigido por plano", "disciplinado"e "pesado". Dentre estas expressões o mais utilizado pelos autores é o "dirigido por plano", possivelmente pela expressão "tradicional"remeter a ideia de "defasado, ultrapassado"; já a expressão "pesada"remete a ideia de "complexidade, dificuldade"; e a expressão "disciplinado"remete a ideia de que poderá não haver disciplina nas metodologias ágeis (LEFFINGWELL, 2006).

Esta metodologia é alicerçada na aplicação sistemática de princípios da engenharia, compreendendo as áreas de qualidade e engenharia de sistemas. Planejamento e criação de arquiteturas bem solidificadas são o foco dessa metodologia, segundo (BOEHM; TURNER, 2004). Podemos dar destaques aos seus objetivos que são previsibilidade, estabilidade e alta garantia. Esta metodologia preza por artefatos bem definidos, verificação, validação, e interfaces detalhadas. Existem conceitos intrinsecamente ligados ao mundo do método dirigido por plano. Podemos resumir na tabela 1 (BOEHM; TURNER, 2004).

Tabela 1 – Conceitos do paradigma dirigido por plano (BOEHM; TURNER, 2004)

Conceito	**Descrição**
Melhoria de processo	Um programa de atividades projetado para buscar um melhor performance e amadurecimento dos processos organizacionais. Avanço dos processos aumentou a começar do trabalho de gestão de qualidade de Deming, Crosby e Jura e tem como finalidade subir a capacidade dos processos de trabalho.
Capacidade de processo	A habilidade ligada de um processo produzir resultados planejados. À medida que a capacidade de cada processo aumenta, ela se torna prevista e mensurável, e as maiores causas de baixa qualidade e produtividade são controladas ou suprimidas.
Maturidade organizacional	Pela comum melhoria de sua capacidade de processo, uma organização é dita maturada. Maturidade abrange não só a capacidade individual de processo, mas também a aplicação geral de processos padrão na organização. Processos comuns são preservados e pessoas são treinadas na sua aplicação. Projetos adequam os artefatos comuns para atender suas necessidades e, uma vez que produzem artefatos comuns, a organização pode iniciar a medir sua eficiência e eficácia, e melhorá-las baseando-se em métricas.
Grupo de processo	Uma acervo de especialistas que simplificam a definição, manutenção e melhoria dos processos usados pela organização.

Gestão de risco	Um processo organizado e analítico para identificar inseguranças que podem causar danos ou perdas (identificar riscos), mensurar os riscos identificados, requintar e aplicar planos de gestão de riscos para prevenir ou lidar com eles.
Verificação	Verificação assegura que os produtos de trabalho (e.g. especificações, projetos, modelos) refletem devidamente os requisitos apropriados para eles (elaborar o produto corretamente).
Validação	Validação confirma a adequação ou valor de um produto de trabalho para o propósito ao qual se destina (construir o produto correto).
Arquitetura de sistema de software	Uma arquitetura de sistema de software define uma coleção de componentes de software e de sistema, conectores e restrições; uma coleção de declarações de necessidades dos envolvidos no sistema; e uma lógica que demonstra que os componentes, conectores e restrições definem o sistema que, se implementado, satisfaria a coleção de declarações de necessidades dos envolvidos.

A tabela 2 mostra exemplos de ponderações de Boehm como representantes dos métodos tradicionais ou dirigido por plano (BOEHM; TURNER, 2004).

Tabela 2 – Exemplos de abordagens dirigidas por plano (BOEHM; TURNER, 2004)

Abordagem	**Proponentes**	**Descrição**	**Referências**
Cleanroom	Harlan Mills, IBM	Usa controle estatístico de processo a verificação matemática para o desenvolvimento de software com confiabilidade certificada. Qualquer a abordagem é inclinada em código livre de erros.	(BECKER; WHITTAKER, 1997), (PROWELL et al., 1999)

CMMI	SEI, DoD, NDIA, Roger Bate, Jack Ferguson, Mike Phillips	É um modelo de referência de capacidade e maturidade de processos, e também um conjunto de métodos de avaliação que tratam uma variedade de disciplinas usando um vocabulário e arquitetura comuns, e um conjunto de áreas de processo.	(SEI, 2002), (SEI, 2006)
PSP / TSP	Watts Humphrey, SEI	PSP é um framework estruturado de formulários, orientações e procedimentos para se desenvolver software. Ele é direcionado para o uso de auto-avaliação para melhorar as habilidades individuais de programação. TSP baseia-se no PSP e apóia o desenvolvimento de software de porte industrial por meio do controle e planejamento de equipes.	(HUMPHREY, 1996), (HUMPHREY, 2000)

2.4.3 Metodologia ágil de desenvolvimento de software

A metodologia ágil de desenvolvimento de software remete ao inicio da programação como uma arte e não por um simples processo industrial (BOEHM; TURNER, 2004). A união do feedback e alterações está estritamente ligado aos métodos ágeis e eles são concebidos para aceitar uma grande quantidade de alterações na sua concepção (WILLIAMS; COCKBURN, 2003). Ademais, esta metodologia salienta a importância do planejamento adaptativo, simplicidade e liberação contínua de software com valor operacional com pequenas iterações com tempo fixado. Em contrapartida com a metodologia tradicional (dirigida por plano), a metodologia ágil aposta na criatividade dos integrantes da equipe e não nos processos (NERUR; MAHAPATRA; MANGALARAJ, 2005) (DYBA, 2000).

Em um encontro em 2001 surgiu o "Manifesto para Desenvolvimento Ágil de Software"por um conjunto de profissionais da área (BECK et al., 2001), que instaurou quatro principais valores desta metodologia:

- Indivíduos e interações mais que processos e ferramentas
- Software em funcionamento mais que documentação abrangente

- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano

(BOEHM; TURNER, 2004) afirma que uma metodologia verdadeiramente ágil deve englobar os seguintes atributos:

- Iterativo: contém muitos ciclos;
- Incremental: não ocorre a entrega do software de uma única vez;
- Auto-organizável: os times escolhem a maneira de trabalhar;
- Emergentes: processos, princípios e estruturas de trabalho são descobertas durante o projeto ao invés de serem predeterminadas.

2.4.3.1 Práticas ágeis

Alicerçado a estes valores e atributos profissionais de desenvolvimento de software foi proposta diversas atividades práticas que possuem caráter, técnico-metodológico (design simples, refatoração, desenvolvimento por teste (TDD), integração contínua) e gerencial (iterações curtas, reuniões diárias, retrospectivas, time-boxing).

2.4.3.2 Processo e modelo de ciclo de vida

Segundo (RAMSIN; PAIGE, 2008) não há especificações consistentes e claras nas literaturas sobre os métodos ágeis. (AMBLER, 2008) mostra uma figura geral do ciclo de vida de desenvolvimento ágil de software, separando em 3 fases este ciclo.

- Iniciação.
- Elaboração e Construção.
- Transição.

Além de mais uma fase de Produção, conforme ilustrado na figura 5.

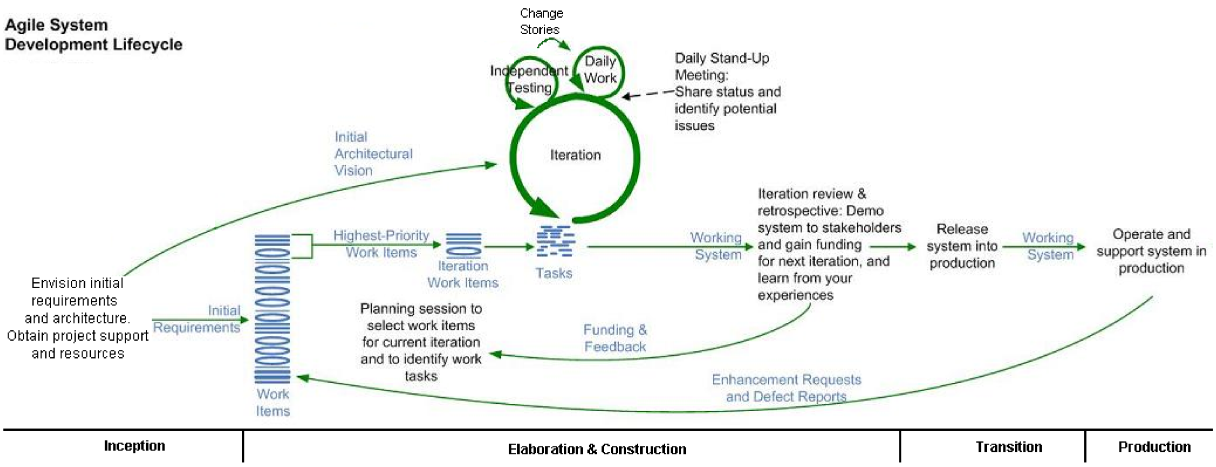

Figura 5 – O ciclo de vida do desenvolvimento ágil de sistemas

Fonte: (AMBLER, 2008)

Outro exemplo de método ágil que é citado por (PÁDUA FILHO, 2009) é o modelo de ciclo de vida *time-boxed*, que indica tempos fixos de intervalos e ocorre a entrega do que se consegue fazer. Aqui encontra-se a a descontinuação mais drástica segundo a ideia das metodologias ágeis, como dito por Leffingwell (LEFFINGWELL, 2006): é uma inversão da pirâmide sobre "qual aspecto determina qual". Historicamente é estimado recursos e prazos, e fixa-se somente o escopo. Já o modelo *time-boxed* propõe o contrário, a fixação dos recursos e prazos e então ocorre a estimação do escopo, conforme figura 6

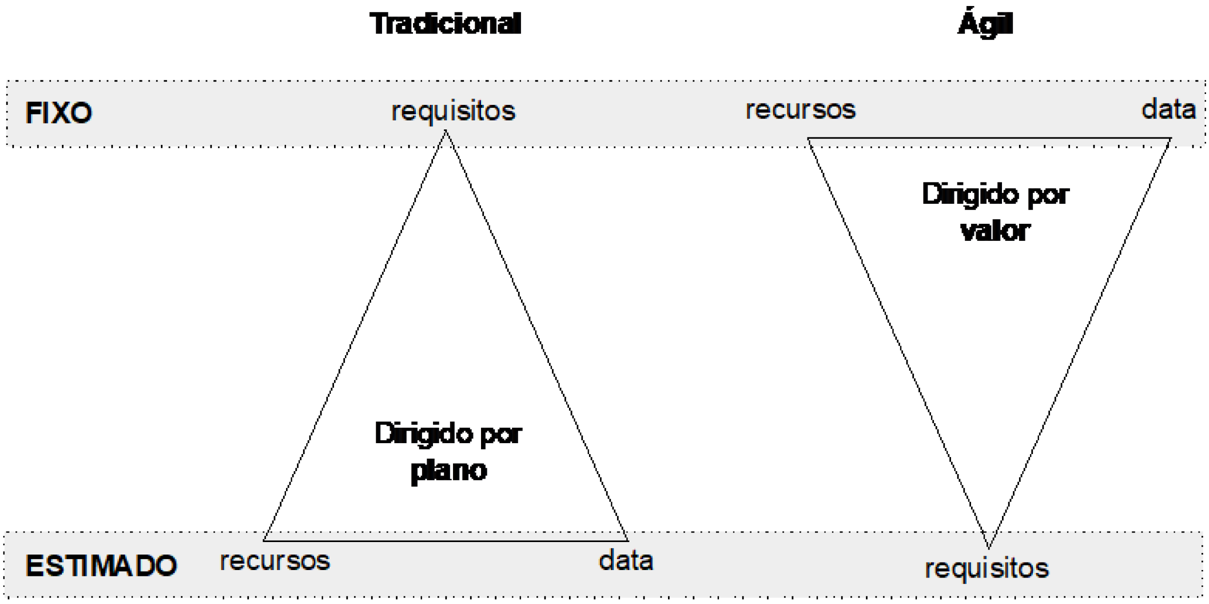

Figura 6 – Recurso e prazo estimados versus escopo estimado

Fonte: (LEFFINGWELL, 2006)

2.4.3.3 Requisitos

Os requisitos na metodologia ágil comumente é exposto em forma de *estórias* (*user stories*). Isto nada mais é que textos narrativos que expõe a maneira que o software irá funcionar para o usuário final (cliente).

A tabela 3 a seguir mostra um breve descrição dos processos ágeis mais citados na literatura (DYBA; DINGSOYR, 2009).

Tabela 3 – Métodos ágeis mais referenciados na literatura

Processo	**Proponentes**	**Descrição**	**Referências**
DSDM – Dynamic Software Develop-ment Method	Dane Faulkner e outros	Divide projetos em três fases: pré-projeto, ciclo de vida do projeto e pós-projeto. Nove princípios norteiam DSDM: envolvimento com usuário, autonomia da equipe do projeto, entrega frequente, tratar necessidades presentes de negócio, desenvolvimento iterativo e incremental, permite alterações, escopo geral definido antes de o projeto iniciar, teste durante o ciclo de vida, e comunicação eficiente e efetiva.	(STAPLETON, 2003)
Scrum	Schwaber, Sutherland e Beedle	Um framework para processo de desenvolvimento ágil com ênfase em práticas de gerenciamento de projeto.	(SCHWABER, 1997), (SCHWABER; BEEDLE, 2001)

XP, XP2	Kent Beck, Eric Gamma e outros	Foca em melhores práticas de desenvolvimento. Consiste em doze práticas: planning game, pequenas releases, metáforas, design simplificado, teste, refatoração, programação em pares, propriedade coletiva, integração contínua, semana de 40 horas, clientes on site e padrões de codificação. A versão revisada "XP2" consiste das seguintes "práticas primárias": sentar-se juntos, toda equipe, workspace informativo, trabalho energizado, programação em pares, estórias, ciclo semanal, ciclo trimestral, descanso, build de 10 minutos, integração contínua, programação test-first e design incremental. Há também 11 "práticas corolárias".	(BECK, 2000a), (BECK, 2000b)
Crystal methodologies	Alistair Cockburn	Uma família de métodos para equipes co-localizadas de tamanhos e criticalidades diferentes: Clear, Yellow, Orange, Red, Blue. O método mais ágil, Crystal Clear, foca na comunicação de pequenas equipes desenvolvendo software que não seja crítico com relação a vidas. Características: entrega frequente, melhoria reflexiva, comunicação osmótica, segurança pessoal, foco, fácil acesso a usuários experts, e requisitos para o ambiente técnico.	(COCKBURN, 2004)

FDD – Feature-Driven Development	Peter Coad e Jeff DeLuca	Combine desenvolvimento ágil e dirigido por modelos com ênfase em modelo de objetos inicial, divisão do trabalho em features, e design iterativo para cada feature. Afirma ser adequado para desenvolvimento de sistemas críticos. Uma iteração de uma feature consiste em duas fases: design e desenvolvimento.	(PALMER; FELSING, 2001)
Lean Software Develop-ment	Mary e Tom Poppen-dieck	Uma adaptação dos princípios de lean production (produção magra), o sistema de produção da Toyota para desenvolvimento de software. Consiste de sete princípios: eliminação de desperdício, ampliação do aprendizado, decidir o mais tarde possível, entregar o mais rápido possível, dar autonomia à equipe, criar integridade, e ver o todo.	(POPPENDIECK; POPPENDIECK, 2003)

2.4.4 RUP - Processo Unificado

O Processo Unificado, também conhecido como RUP, é uma amostra de processo iterativo para projetos que usam a Programação Orientada a Objetos (POO). O desenvolvimento iterativo é estruturado em um encadeamento de pequenos projetos, com um intervalo de tempo fixo, conhecido como iterações; o produto de cada compõe-se de um sistema verificado, integrado e executável. Qualquer iteração engloba suas próprias atividades de análise de requisitos, projetos, implementação e teste (LARMAN, 2002).

As mais boas técnicas de Processo Unificado incluem: desenvolver softwares iterativamente, gerir requisitos, utilizar arquiteturas fundamentadas em componentes, realizar a modelagem de softwares usando os diagramas UML e analisar a qualidade do software.

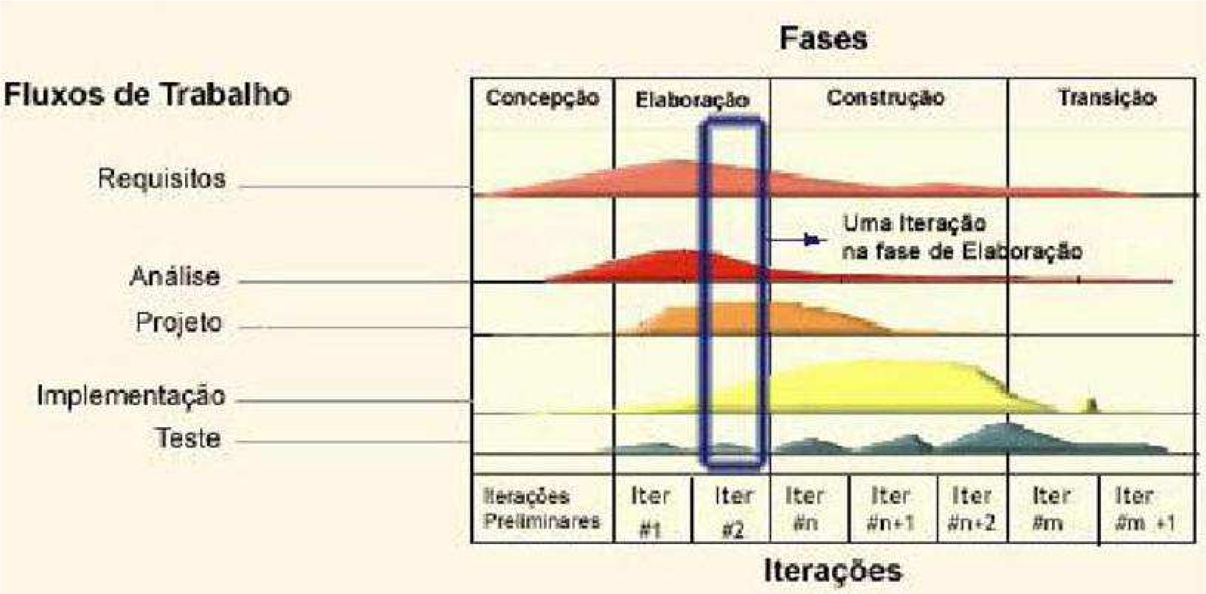

Figura 7 – Fases e iterações do PU

Fonte: (CANEZ, 2011)

Na Figura 7 é ilustrado as fases do Processo Unificado e, ao mesmo tempo, podemos destacar o conceito de iteração (ciclos). No decorrer de cada iteração, observamos que os fluxos de trabalho (requisitos, análise, projeto, implementação e teste) encontram-se em andamento.

Todo fluxo apresenta uma intensidade grande no decorrer das seguintes fases:

- *Requisitos:* etapas de fecundação e preparação;
- *Análise:* sobretudo na etapa de preparação;
- *Projeto:* etapas de preparação e fabricação;
- *Implementação:* sobretudo na etapa de fabricação;
- *Teste:* sobretudo nas etapas de fabricação e transição.

(LARMAN, 2004) reitera que o ciclo de vida iterativo é fundamentado no aperfeiçoamento e incrementação de um sistema através de diversas iterações, com feedback e adequações cíclicas como essenciais propulsores para dirigir-se para um sistema propício. O sistema evolui incrementalmente no decorrer do tempo, iteração por iteração, explicando por que esta abordagem é conceituada como desenvolvimento iterativo e incremental.

2.4.5 eXtreme Programming (XP)

EXtreme Programming, Programação Extrema, mais conhecida como XP, é um metodologia leve que usa o processo ágil, desta forma, é necessário realizar uma análise profunda antes de se dar início. Antagonicamente ao Processo Unificado, a XP esquiva-se da documentação formal e deposita sua confiança mais na verbal, todavia, as duas metodologias são de desenvolvimento incremental e iterativo.

Este metodologia foi foi elaborada visando atender equipes pequenas ou médias no processo de desenvolvimento de software, pois estas buscam tratar incessantemente com requisitos vagos e em contínua mudança (BECK, 2000a).

O XP é indicado para pequenos e médios projetos, que podem variar de um a seis meses, próprio também em situações em que se tem pouco prazo para entrega. Em média, o grupo de 2 a 10 programadores é que aderem ao XP.

Comunicação, feedback, simplicidade e coragem são preceitos essenciais para a formação básica desta metodologia. Segundo FOWLER estes preceitos podem ser contemplados por doze práticas.

Para KNIBERG muitas destas práticas, podem ser adotadas com outras metodologias, trazendo um aperfeiçoamento em processos e motivação. Será esclarecido as causas de sua seleção seguidamente:

- Programação em pares: assente um código de qualidade superior, em razão do recebimento de "refatoração"praticamente em tempo de execução e além disso propaga o conhecimento no meio dos integrantes da equipe, criando um ganho na produtividade.

- Desenvolvimento orientado a testes (TDD): funciona como uma forma de proteção, dado que por meio do desenvolvimento de testes que podem ser realizados automaticamente, garante-se ao desenvolvedor funcionalidades novas que são inseridas no sistema, assim poderá ser identificado qualquer inconsistência, gerando assim uma maior segurança aos desenvolvedores.

- Integração contínua: garante a resolução de prováveis problemas futuramente e garante a economia de tempo, pois caso seja identificado alguma dificuldade na integração este será tranquilamente identificado, por se tratar de um pequeno lote integrado.

- Design incremental: para permitir uma continuidade (manutenção) mais ágil e facilitada, incia-se de um design simples, pois assim permite-se executar melhoras contínuas no sistema.

- Propriedade coletiva do código: gera-se uma crescente responsabilidade de todos os integrantes do projeto, como uma equipe, pois a pessoa não fica "responsável"por apenas um aglomerado de código. Também gera uma enorme facilidade em manutenção, transmissão de conhecimento e qualidade.

- Padrão de codificação: é implantado um padrão definido de código (linguagem) para ser usado em todo o projeto, assim haverá qualidade e facilidade de manutenção.

- Ritmo sustentável: não gerar sobrecarga de trabalho para equipe, pois fatalmente gerará um queda na qualidade no serviço realizado e dificilmente gerará um aumento na produção.

Estas são as seis etapas que esta metodologia compreende. A técnica de fatoração contínua é empregada com certa periodicidade.

1. Levantamento de requisitos

2. Análise

3. Desenho da arquitetura

4. Implementação

5. Teste

6. Manutenção

2.4.6 Scrum

O Scrum é uma metodologia de desenvolvimento de software ágil, que se manisfestou no início da década de 90. Ela opera de maneira prática e busca uma quantidade menor de documentação. Neste método não desconsidera-se ferramentas, processos, documentação, contratos ou ideias, mas tem como foco os indivíduos e iterações, a competência do software, a cooperação e os feedbacks. O Scrum foge dos padrões recomendados pelos métodos tradicionais, conquanto não entre em objeção com o mesmo. Ele da ênfase na comunicação, trabalho em grupo, flexibilidade e trabalho incremental.

Assegurar uma maior flexibilidade, aptidão para tratamento de sistemas complexos e simples, fabricar sistemas sujeitos a requisitos iniciais e adicionais na execução do projeto. Essas coisas somente são alcançáveis pelo fato de que no final de um clico de desenvolvimento, têm-se um produto executável e testado.

É possível dividir esta metodologia em três níveis:

1. **Planejamento *(backlog)*:** aqui são definidos os processos, designer da arquitetura do sistema, o pessoal e a liderança, os pacotes a serem desenvolvidos. Nesta situação, há a colaboração dos clientes e dos demais departamentos, ocorrendo o levantamento de requisitos e a incumbência de prioridade dos mesmos. O cliente nesta metodologia é parecido ao do método XP, pois ele é peça fundamental na equipe de desenvolvimento, pois assim há uma grande convivência com os desenvolvedores. No nível de etapas (sprints), para cada equipe é atribuída uma cota do backlog para ser trabalhada, e este, fica isento de modificações durante o desenvolvimento. Cada sprint pode durar de uma a quatro semana, e ao término, é mostrado um executável da trabalho realizado.

2. **Sprint *(Ciclos, Etapas)*:** seguindo o pensamento de (PRESSMAN, 2006), são feitas reuniões diariamente, organizadas pelos líderes das equipes, com duração rápida (15 minutos) de forma que todos os desenvolvedores respondam a três questionamentos simples:

    a) O que você realizou desde a última reunião?
    b) Quais problemas você enfrentou?
    c) Em que você trabalhará até a próxima reunião?

    Por meio destas reuniões, algumas vantagens devem ser destacadas:

    - Maior aproximação entre as membros das equipes;
    - Solução rápida para os problemas;
    - Partilha de conhecimento;
    - Medição da evolução do desenvolvimento diariamente;
    - Atenuação significativa de riscos.

    Podemos dar destaque a fase de sprint na Figura 8, que é onde ocorre o planejamento, implementação do produto, revisão, exposição do mesmo e progresso do *product backlog*. É importante frisar a representação das reuniões diárias, que são essenciais nessa metodologia.

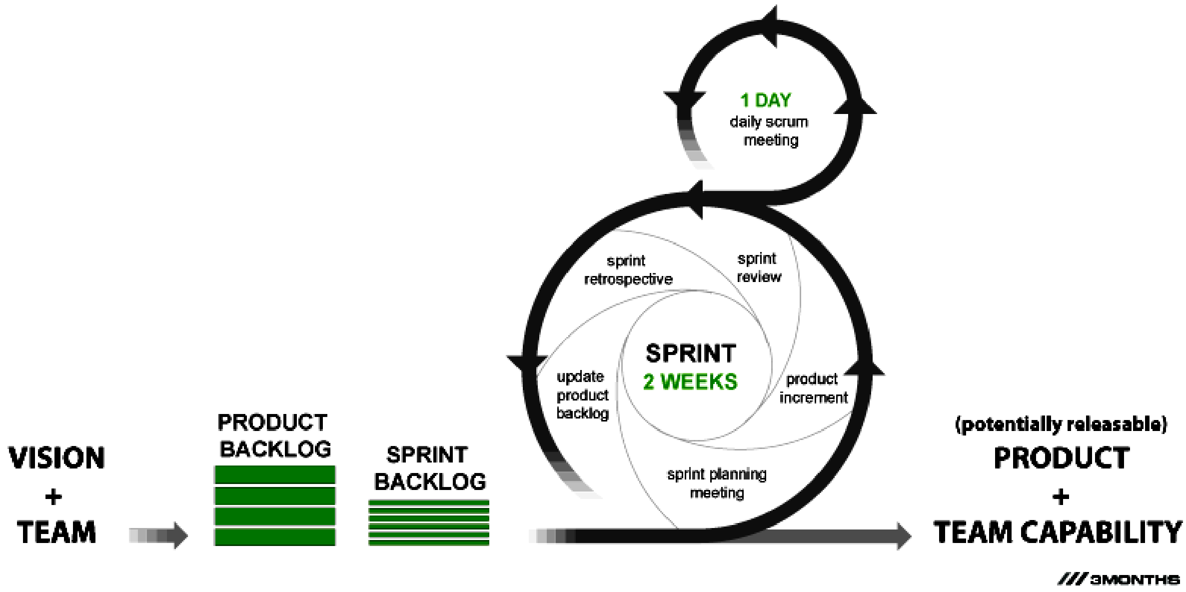

Figura 8 – Sprint (ciclos de desenvolvimento)

Fonte: (ILLUSTRATING. . . , 2010)

Durante o tempo de revisão da sprint, os prazos de entregas devem ser cumpridos, pois desta forma será possível realizar a apresentação para o cliente e demais *stakeholders*, desta maneira, qualquer opinião de mudanças serão inclusas ao *backlog*. A revisão apresenta benefícios, como por exemplo:

- Demonstração de resultados concretos ao cliente;
- Integração e teste do software;
- Motivação extra para equipe.

Com o intuito de ajudar no processo de criação, o Scrum adere um quadro, intitulado KanBan, que por meio dele é possível conhecer quem está desenvolvendo e o quê, quais dificuldades estão sendo enfrentadas, o que é precisa fazer ainda, o que já está pronto e o que já foi testado, ou ainda se encontra em fase de testes. Outrossim, é possível acompanhar minuciosamente o processo de criação, policiando os prazos e impedindo que mais de uma pessoa realize a mesma tarefa. Estas tarefas e os *bugs* (erros) são discrepantes por cores com o objetivo de facilitar a leitura e o entendimento do quadro. A Figura 9 mostra um quadro Kanban, onde observamos que cada coluna simboliza uma fase de desenvolvimento.

No decorrer da fase de desenvolvimento, um integrante da equipe escolhe uma tarefa para ser desenvolvida, então a exclui da coluna "para fazer (TO DO)"e a coloca "em desenvolvimento (DEV)". Após sua finalização, os testes são realizados, e se tudo ocorrer sem problemas, a atividade é concluída e movida para a coluna "feito (DONE)", caso contrário deve-se realizar o registro do problema, na sessão específica para isto (bugs) e a tarefa retorna para a coluna "em desenvolvimento (DEV)". Esse processo todo é feito até que a aplicação fique completa no final.

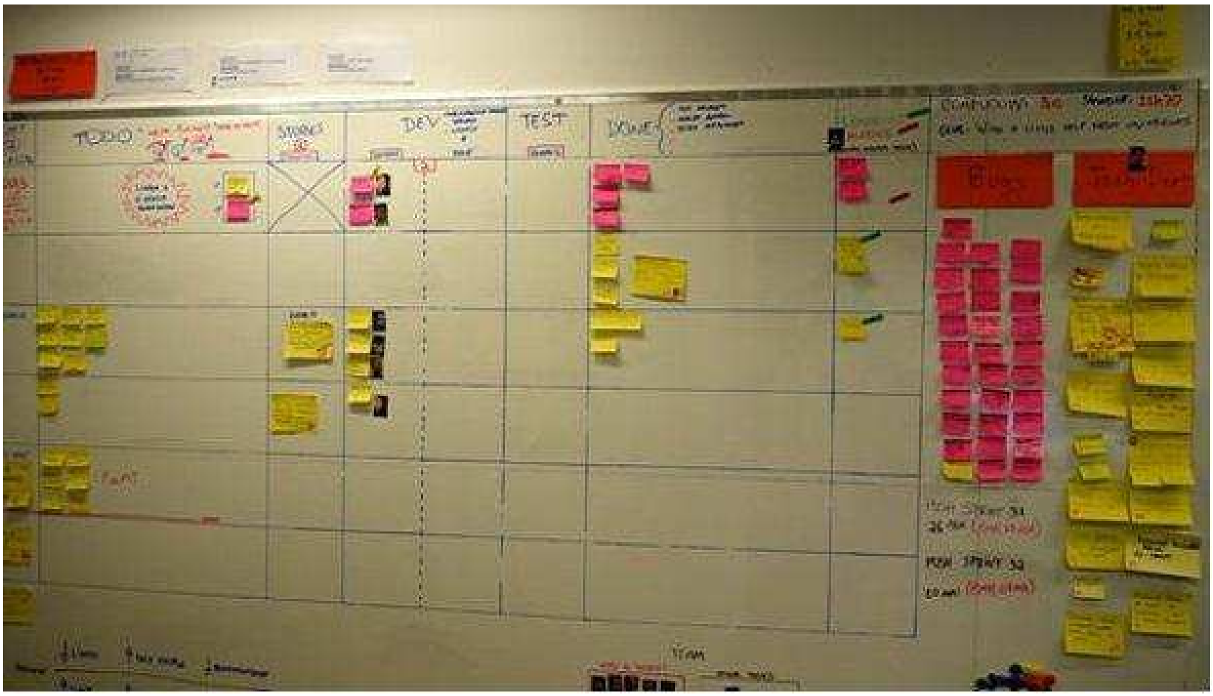

Figura 9 – O quadro Kanban

Fonte: (VALENTE, 2010)

3. **Conclusão:** Aqui encontra-se a última fase, o início dela dar-se-á no momento em que todos os tópicos (tempo, competitividade, requisitos, qualidade e custo) são aceitáveis. Neste momento transcorrem os testes de integração, testes de sistema, documentação do usuário, elaboração do manual do usuário e conteúdo de marketing.

2.4.7 CMMI - Capability Maturity Model Integration

O CMMI - Capability Maturity Model Integration (SEI, 2002) não é uma metodologia de desenvolvimento de software, ela é um modelo de referência de maturidade de capacidade que descreve um caminha de melhoria evolutiva que parte de processos e *ad hoc*, até chegar a processos maduros e disciplinados, que visam a melhoria expressiva da qualidade dos produtos e a eficiência do trabalho (SEI, 2006). Explanado pelo SEI da Universidade Carnegie Mellon, o CMMI busca arquitetar um único modelo para o processo de aperfeiçoamento corporativo, unificando diferentes modelos e disciplinas.

As técnicas indicadas pelo CMMI são juntadas em áreas de processos, que é um grupo de ações relacionadas a uma determinada área que, quando realizadas em conjunto, atendem a um conjunto de objetivos considerado importante para a realização de upgrades na área. O CMMI foi guiado por modelos que utilizavam formas diferentes para dimensionar o domínio das organizações com relação às áreas de processo:

- **Nível de maturidade:** Cada nível é uma fase de upgrade dos processos, em um grupo predefinido de áreas de processo, em que todos os objetivos foram alcançados dentro de um conjunto.
- **Nível de capacitação:** que é um indicativo de upgrade de processos dentro uma única área.

Aceita-se os dois níveis no CMMI, mas o nível de maturidade ocupa um espaço maior da atenção pois está diretamente concatenado ao contexto industrial do modelo. A Tabela 4 mostra a lista de níveis de maturidade estabelecidos pelo CMMI.

Tabela 4 – Níveis de maturidade do CMMI (PÁDUA FILHO, 2009)

Nível	**Nome**	**Descrição**
1	Executado (performed)	Processos informais e *ad hoc*, às vezes caóticos.
2	Gerido (managed)	Processos planejados e executados conforme políticas.
3	Definido (defined)	Processos bem caracterizados, entendidos e padronizados.
4	Gerido quantitativamente (quantitatively managed)	Processos geridos em função de objetivos quantitativos de qualidade e desempenho.
5	Otimizante (optimizing)	Processos em melhoria contínua.

Para conhecermos um nível de maturidade é preciso alcançar os objetivos de um conjunto de áreas de processo, vejamos na tabela 5 o agrupamento das áreas de processo do CMMI por nível de maturidade.

Tabela 5 – Áreas de processo do CMMI por nível de maturidade

Nível de Maturidade	**Sigla**	**Nome**	**Categoria**
2	REQM	Gestão de Requisitos	Engenharia
2	PP	Planejamento de Projetos	Gestão de projetos

2	PMC	Monitoração e Controle de Projetos	Gestão de projetos
2	SAM	Gestão de Acordos com Fornecedores	Gestão de projetos
2	MA	Medição e Análise	Suporte
2	PPQA	Garantia da Qualidade de Processos e Produtos	Suporte
2	CM	Gestão de Configurações	Suporte
3	RD	Desenvolvimento de Requisitos	Engenharia
3	TS	Solução Técnica	Engenharia
3	PI	Integração de Produtos	Engenharia
3	VER	Verificação	Engenharia
3	VAL	Validação	Engenharia
3	OPF	Focalização dos Processos da Organização	Gestão de processos
3	OPD	Definição dos Processos da Organização	Gestão de processos
3	OT	Treinamento da Organização	Gestão de processos
3	IPM	Gestão Integrada de Projetos	Gestão de projetos
3	RSKM	Gestão de Riscos	Gestão de projetos
3	DAR	Análise e Resolução de Decisões	Suporte
4	OPP	Desempenho dos Processos da Organização	Gestão de processos
4	QPM	Gestão Quantitativa de Projetos	Gestão de projetos
5	OID	Inovação e Implantação na Organização	Gestão de processos
5	CAR	Análise e Resolução de Causas	Suporte

3 Trabalhos relacionados

3.1 Introdução

Mesmo com a evolução da indústria no uso das metodologias ágeis, há muito chão para percorrer, pois é necessário levantar as reais implicações do uso dessa metodologia e sua combinação com abordagens tradicionais. As pesquisas ainda estão no inicio e este setor encontra-se em um estágio imaturo e pouquíssimo explorado (DYBÅ; DINGSØYR, 2008) (DYBA; DINGSOYR, 2009) (ROMBACH; SEELISCH, 2008).

Uma explosão de propostas de desenvolvimento de software aconteceram nas últimas décadas (RAMSIN; PAIGE, 2008) (ROMBACH; SEELISCH, 2008). Não é do conhecimento de todos a medição do real valor que um processo possui sobre o outro, e a literatura na área é quase sempre contagiada com favoritismo e subjetividade, o que atiça um trabalho científico (RAMSIN; PAIGE, 2008).

O fato de até o momento ter sido realizado poucos estudos empíricos com a exigência aceitável de confiabilidade e validade é mínimo o conhecimento dos reais benefícios da metodologia ágil até o momento (DYBÅ; DINGSØYR, 2008) (DYBA; DINGSOYR, 2009). O modelo XP -eXtreme Programming é uma das poucos metodologias que foram realmente experimentadas realmente, o que nos dá uma base empírica (BECK, 2000a). O Scrum é um *framework* que tem sido muito bem aceito pela indústria, mesmo sendo o método menos pesquisado, quando equiparado a sua popularidade nas empresas (DYBÅ; DINGSØYR, 2008). A unificação dos métodos ágeis com métodos tradicionais é uma área de pesquisa ainda não desbravada (GALAL-EDEEN; RIAD; SEYAM, 2007).

Abaixo é apresentada uma revisão da literatura relacionada a este pesquisa, subdividida em seções com objetivos específicos:

Combinação ágil e tradicional: é exposto e tratado as propostas de união de métodos ágeis e tradicionais com o objetivo de analisar seus resultados quantitativos e qualitativos para ajudar no levantamento de características que a proposta de integração deve-se levar em conta.

Produtividade em desenvolvimento ágil: é exposto e tratado os estudos empíricos existentes sobre o impacto de produtividade pelo uso de processos ágeis.

O procedimento adotado neste revisão foi o de realizar busca nas bases de dados mais abrangentes na área (IEE XPlore, ACM Digital Library, ScienceDirect e Google Scholar), fazendo buscas exploratórias principalmente com os termos relacionados aos objetivos da pesquisa:

- Agile (+ software OU development)
- Agility (+ software OU development)
- Scrum
- Traditional (+ software OU development) (+ agile)
- Plan-driven (+ software OU development) (+ agile)
- Productivity (+ software OU development) (+ agile)

A cada consulta, os resultados passaram por uma pré-filtragem por título e depois, em caso de dúvida, por resumo. A seleção dos trabalhos também foi baseada na recente revisão sistemática sobre estudos empíricos em desenvolvimento ágil (DYBÅ; DINGSØYR, 2008). Não foi dada especial consideração neste capítulo aos trabalhos sobre propostas, reflexões, sugestões e relatos de experiências não baseados em evidências ou baseados apenas em evidência anedótica. Os demais trabalhos que possuem algum amparo científico, ou que possuem alguma relevância direta para a discussão da pesquisa desta tese, são discutidos a seguir.

3.2 Combinação ágil e tradicional

É discutido como a maneira de preparação da arquitetura no desenvolvimento com o método XP pode ser enriquecido com a serventia das orientações de diversos métodos específicos para arquitetura desenvolvidos pelo SEI da Universidade Carnegie Mellon, como pode ser visto a seguir (NORD; TOMAYKO; WOJCIK, 2004) (NORD; TOMAYKO, 2006):

- **Quality Attribute Workshop (QAW)**: Auxilia a equipe de desenvolvimento a compreender o problema levantando requisites de atributos de qualidade na forma de cenários. Basear-se em cenários e objetivos de negócio assegura que os desenvolvedores tratem os problemas certos. (BARBACCI et al., 2003)
- **Attribute-Driven Design (ADD)**: Define a arquitetura do software baseando o processo de projeto nos cenários de atributos de qualidade priorizados que o software precisa contemplar. Este método ajuda a identificar as coisas mais importantes a se fazer para assegurar que o projeto esteja no caminho certo de atender os principais atributos de qualidade e entregar valor ao cliente. (BASS L.; CLEMENTS,)
- **Trade-off Analysis Method (ATAM) e Cost-Benefit Analysis Method (CBAM)**: Oferecem orientações detalhadas para análise do projeto e obtenção de feedback rápido sobre riscos. O ATAM oferece aos arquitetos de software um

framework para a compreensão de trade-offs e riscos que se apresentam à medida que tomam decisões arquiteturais. O CBAM auxilia os arquitetos a considerar o retorno do investimento (ROI) de cada decisão arquitetural e oferece orientações a respeito de trade-offs econômicos envolvidos. (BARBACCI et al., 2003) (BASS L.; CLEMENTS,).

Esta proposta de Nord et. al. compõe-se, em essência, de um grupo de indicações de como os métodos desenvolvidos da Carnegie Mellon podem ser processados de forma ligada a alguns princípios ou práticas da filosofia ágil que o método XP incorpora. Este projeto de Nord et. al. não é uma pesquisa, ela é apenas uma proposta e não demonstra resultados decorrentes de algum estudo empírico que comprove suas considerações. A proposta tem a vantagem de oferecer práticas e orientações aprofundadas para criação da arquitetura de software sob várias perspectivas, o que enriquece o gerenciamento de risco, comunicação coerente entre envolvidos no projeto, apoio a decisões técnicas e financeiras com maior responsabilidade, e apoio a sistemas críticos. A desvantagem desta proposta é a falta de evidências empíricas quantitativas e qualitativas para identificar e basear seus benefícios reais. Mesmo assim esta pesquisa foi incluído na discussão devido ao seu foco em arquitetura de software e pela sua importância do SEI e de seus trabalhos na melhoria de processos.

3.2.1 Técnica para balanceamento ágil e tradicional

Em 2004 foi publicado uma obra literária sobre balanceamento de processo ágil orientado por plano (BOEHM; TURNER, 2004) em conjunto com outras obras literárias correlacionadas (BOEHM; TURNER, 2004) (BOEHM; TURNER, 2003a) (BOEHM; TURNER, 2003b).

Tamanho do projeto, criticalidade do projeto, dinamismo do ambiente de desenvolvimento em relação a alterações, pessoal e fatores culturais formam um aglomerado de fatores críticos em um projeto que visava determinar o quanto o desenvolvimento do mesmo deve ser ágil ou tradicional de acordo com a Boehm e Turner. O risco é a chave para se encontrar o equilíbrio ideal entre agilidade e "disciplina", analisando este risco à luz dos cinco fatores mencionados, como mostrado na figura 10.

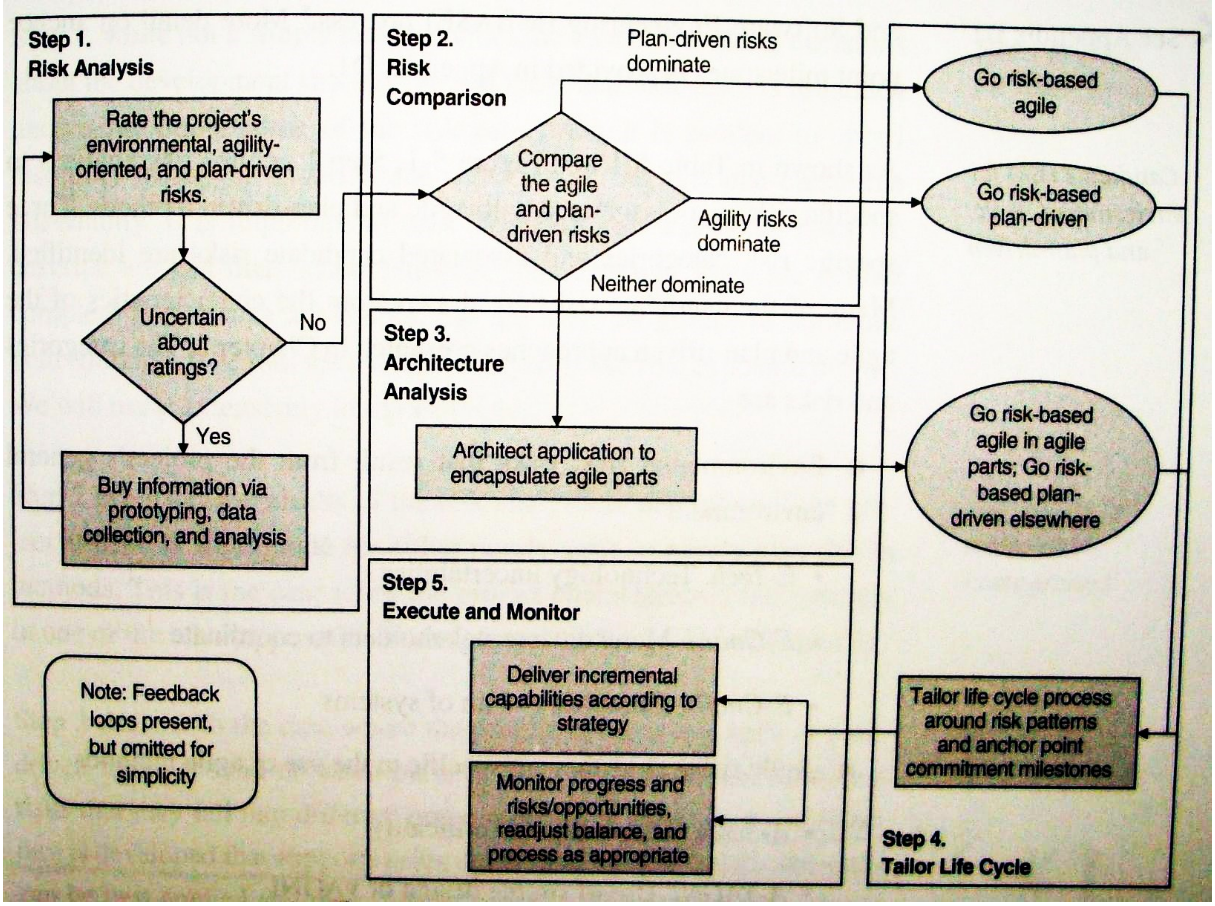

Figura 10 – Visão geral do método de balanceamento ágil e tradicional

Fonte: (BOEHM; TURNER, 2004)

Boehm e Turner testaram sua metodologia em duas ocasiões e a mesma funcionou satisfatoriamente, conquanto a sua abordagem ainda requer desenvolvimento, mas podemos destacar o seu funcionamento aceitável nas situações em que foi aplicada.

A escolha dos cinco fatores e a identificação do risco como ponto chave para o balanceamento é uma vantagem a ser destacada, pois a mesmo é alicerçada na bagagem de empresas das mais diversas organizações. Este projeto aguça a interesse pelo fato de conter uma discussão bem abrangente especificamente sobre métodos ágeis *versus* métodos tradicionais. Em contrapartida não é disponibilizado um *framework*, apenas uma ferramenta para auxiliar a decidir o quanto de cada um das duas metodologias deve ser usada em determinado projeto.

3.2.2 Integração de agilidade no modelo stage-gate

Na figura 11 podemos visualizar um estudo sobre a integração da metodologia XP no modelo *stage gate* (KARLSTRÖM; RUNESON, 2006) (KARLSTROM; RUNESON, 2005). (COOPER, 2001) - um tipo de modelo de gestão de projetos usualmente utilizado na indústria, levando em consideração que o desenvolvimento de software não é uma atividade isolado, e sim correspondente a subprojetos em um local mesclado de desenvolvimento de sistemas, marketing, planejamento de produção, etc. O modelo *stage gate* preconiza passos sequenciais bem definidos por quais um projeto deve passar, de forma a dar apoio não somente à comunicação dentro do projeto, mas também para tomada de decisão dos envolvidos. *Gate* como é chamada a passagem de um estágio para outro.

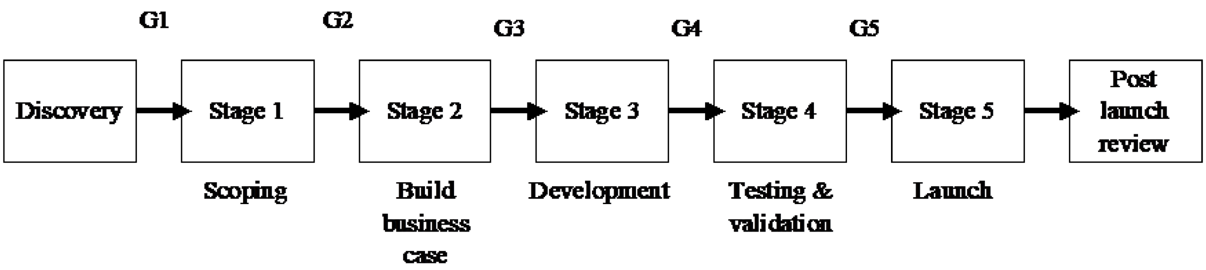

Figura 11 – Modelo Stage Gate

Fonte: (COOPER, 2001)

Após um estudo de caso fundamentado em entrevistas com profissionais de três grandes empresas, foi identificado que métodos ágeis disponibilizam ferramentas robustas para microplanejamento, controle diário do trabalho e relato de progresso. Também foi identificado neste estudo de caso que as equipes podem comunicar-se de maneira bem mais eficaz quando se valem de software funcionando e reuniões pessoalmente, do que quando comparado com documentos formais. Conquanto, este modelo disponibiliza aos métodos ágeis formas de coordenar trabalho com outras equipes e de interligar com as áreas de marketing e alta gestão. Estes aspectos é parte essencial da contribuição central e o ponto forte da pesquisa. Em contrapartida Karlström e Runeson focaram o estudo mais no *gate* 3 (Figura 11, ou seja, nas suposições e questões relacionadas na passagem do estágio 2 para o 3, que é usado a metodologia XP. Outrossim, a entrevista com profissionais é que está alicerçada a pesquisa, o que deixa o estudo muito exposto a erros por causa da natureza deste método de colhimento de dados (YIN, 2015).

3.2.3 Vantagens e problemas dos métodos ágeis

Foi desenvolvido por Petersen e Wohlin (PETERSEN; WOHLIN, 2009) estudo focada em levantar as vantagens e problemas dos métodos ágeis baseada na revisão de literatura de Dybå e Dingsøyr (DYBÅ; DINGSØYR, 2008), e também é incluso os resultados de outra pesquisa (PIKKARAINEN et al., 2008) além também de sua pesquisa própria realizada na empresa Ericsson AB. A pesquisa de Petersen e Wholin é que ela foi realizada em larga escala quando comparada com as outras pesquisas. Eles identifificam as seguintes vantagens na pesquisa:

- **Vantagem 1:** Melhor passagem de conhecimento para melhor comunicação e feedback frequente de cada iteração. (BAHLI; ZEID, 2005), (KARLSTRÖM; RUNESON, 2006), (SVENSSON; HOST, 2005), (PIKKARAINEN et al., 2008), (PETERSEN; WOHLIN, 2009).
- **Vantagem 2:** Clientes são identificados pelos desenvolvedores como muito preciosos, permitindo aos desenvolvedores ter discussões e ter rápido feedback. (KARLSTRÖM; RUNESON, 2006), (MANN; MAURER, 2005), (SVENSSON; HOST, 2005), (TESSEM, 2003)
- **Vantagem 3:** Programação em pares auxilia no aprendizado se os parceiros são alterados regularmente. (TESSEM, 2003)
- **Vantagem 4:** Controle do processo, transparência e qualidade são aumentados através de integração contínua e tarefas pequenas e gerenciáveis. (KARLSTRÖM; RUNESON, 2006), (PETERSEN; WOHLIN, 2009)
- **Vantagem 5:** XP é muito encaminhado pelo aspecto técnico, avigorando os engenheiros e assim aumentando sua motivação. (KARLSTRÖM; RUNESON, 2006)
- **Vantagem 6:** Equipes pequenas e reuniões cara a cara diariamente agregam a cooperação e ajudam a obter melhores insights no processo de desenvolvimento. (SVENSSON; HOST, 2005), (PETERSEN; WOHLIN, 2009)
- **Vantagem 7:** O ambiente de trabalho é percebido como pacífico, confiável, responsável, e preservador da qualidade de vida profissional. (ROBINSON; SHARP, 2004)
- **Vantagem 8:** Clientes gostam da participação ativa em projetos pois isso os concede controlar o projeto e o processo de desenvolvimento, além de se manterem atualizados. (ILIEVA; IVANOV; STEFANOVA, 2004)
- **Vantagem 9:** Os desenvolvedores identificam o local de trabalho confortável e se sentem mais produtivos usando programação em pares. (MANNARO; MELIS; MARCHESI, 2004)
- **Vantagem 10:** Programadores estudantes identificam a qualidade do código utilizando programação em pares. (MANNARO; MELIS; MARCHESI, 2004)
- **Vantagem 11:** Equipes pequenas com pessoas tendo diferentes papéis demandam somente pequenas porções de documentação pois esta é trocada pela comunicação

direta facilitando o aprendizado e entendimento de cada um. (PETERSEN; WOHLIN, 2009)

- **Vantagem 12:** Pequenos pacotes de requisitos permitem que se realize e libere pacotes de requisitos rapidamente, o que leva à redução da volatilidade de requisitos nos projetos. (PETERSEN; WOHLIN, 2009)

- **Vantagem 13:** O desperdício de trabalho não utilizado (documento de requisitos, componentes implementados, etc.) é reduzido pois os pequenos pacotes iniciados são sempre implementados. (PETERSEN; WOHLIN, 2009)

(PETERSEN; WOHLIN, 2009)

As vantagens números 12 e 13 são as únicas consideradas únicas. As vantagens 1, 4 e 6 foram consideradas equivalentes às outras já conhecidas. A vantagem número 11, mesmo sendo considerada parecida com a vantagem 6 foi dado um espaço exclusivo pela importantancia de se substituir documentação por comunicação.

As mais importantes vantagens identificas na metodologia ágil tem haver com benefícios de comunicação ampliando o aprendizado e passagem de conhecimento como pode ser observado nas vantagens 1, 2, 6 e 11 e que ratifica os resultados de (KARLSTRÖM; RUNESON, 2006). Além do mais, é ressaltado que as pessoas acham praticáveis a utilizam dos métodos agéis, como pode ser visto nas vantagens 7 e 9, e os programadores se entusiasmam mais, visto na vantagem 5, coom exceção de que uma pesquisa mais nova identificou uma maior cansaço nos programadores depois da desão ao Scrum (LI; MOE; DYBÅ, 2010). Os outros estudos referem-se a *feedback* entre clientes e programadores (vantagens 2 e 8) e benefícios da programação em pares, visto nas vantagens 3, 9 e 10.

É exposto abaixo também os problemas detectados por Petersen e Wohlin. A lista de problemas é bem maior que a de vantagens, pois muitos destes foram descobertas inéditas. O fato da pesquisa de Petersen e Wohlin ter sido realizada em larga escala, explorou as fraquezas das metodologias ágeis e assim gerou estes problemas inéditos quando comparada aos estudos anteriores realizados em projetos menores (COHEN; LINDVALL; COSTA, 2004) (RAMSIN; PAIGE, 2008).

- **Problema 1:** A realização de testes contínuos requer muito esforço (exemplo: a criação de um ambiente de teste integrado é difícil para diferentes plataformas e dependências entre sistemas). (SVENSSON; HOST, 2005) (PETERSEN; WOHLIN, 2009)

- **Problema 2:** Não há foco suficiente em arquitetura no desenvolvimento ágil, acarretando decisões de projeto ruins. (MCBREEN; BY-BECK, 2002), (ROSENBERG; STEPHENS, 2003), (PETERSEN; WOHLIN, 2009)

- **Problema 3:** Desenvolvimento ágil é fraco em projetos de grande escala. (COHEN; LINDVALL; COSTA, 2004), (PETERSEN; WOHLIN, 2009)
- **Problema 4:** Programação em pares é percebida como exaustiva e ineficiente. (ILIEVA; IVANOV; STEFANOVA, 2004), (MACKENZIE; MONK, 2004), (TESSEM, 2003)
- **Problema 5:** Membros da equipe precisam ser altamente qualificados para obter sucesso usando agilidade. (HILKKA; TUURE; ROSSI, 2005)
- **Problema 6:** As equipes são altamente coesas, o que significa que a comunicação entre as equipes funciona bem. Porém, a comunicação entre equipes é prejudicada. (KARLSTRÖM; RUNESON, 2006), (PIKKARAINEN et al., 2008)
- **Problema 7:** O fortalecimento dos engenheiros torna os gestores temeroses inicialmente, e assim requer treinamento suficiente para os gestores. (KARLSTRÖM; RUNESON, 2006)
- **Problema 8:** Implementação começa muito cedo, assim questões técnicas aparecem muito cedo do ponto de vista gerencial. (KARLSTRÖM; RUNESON, 2006)
- **Problema 9:** Clientes onsite tem que se comprometer com todo desenvolvimento, o que os coloca sob stress. (MARTIN; BIDDLE; NOBLE, 2004)
- **Problema 10:** Da perspectiva dos estudantes, programação em pares não é aplicável se um parceiro é muito mais experiente que o outro. (MELNIK; MAURER, 2002)
- **Problema 11:** A entrega dos requisitos para serem projetados demora devido ao processo de decisão ser complexo. (PETERSEN; WOHLIN, 2009)
- **Problema 12:** A lista de prioridades é essencial no modelo da empresa para se trabalhar e é difícil de criar e manter. (PETERSEN; WOHLIN, 2009)
- **Problema 13:** Atividades de design possuem livre capacidade devido a longos períodos de espera pois em engenharia de requisitos a tomada de decisões complexas tomam muito tempo. (PETERSEN; WOHLIN, 2009)
- **Problema 14:** Redução de cobertura de testes nos projetos devido à falta de testes independentes e escassez de projetos, demandando que a última versão do sistema compense a cobertura. (PETERSEN; WOHLIN, 2009)
- **Problema 15:** O processo da empresa requer que se produza muita documentação de testes. (PETERSEN; WOHLIN, 2009)
- **Problema 16:** A gestão de configuração requer alto esforço para coordenar o alto número de releases internas. (PETERSEN; WOHLIN, 2009)

- **Problema 17:** O desenvolvimento do ambiente de configuração para selecionar partes para a customização de soluções demanda um longo tempo devido ao início tardio do trabalho de empacotamento do produto e uso de bibliotecas de programação sequencial. (PETERSEN; WOHLIN, 2009)

- **Problema 18:** O trabalho de empacotamento do produto é aumentado pois ele ainda é visto de um ponto de vista técnico, mas não de um ponto de vista comercial. (PETERSEN; WOHLIN, 2009)

(PETERSEN; WOHLIN, 2009)

Podemos separar em três categorias os problemas encontrados na pesquisa:

- **Problemas gerais:** são aqueles que estão ligados aos aspectos centrais da filosofia ágil, como por exemplo os prejuízos no *design* da solução por causa do baixo enfoque em arquitetura (Problema 2), dificuldade na adpatação para testes contínuos (Problema 1), necessidade de membros qualificados (Problema 5), desvantagem da pouca documentação (Problema 6) e deficiência em contextos de larga escala (Problemas 3, 11, 12, 13, 14, 15, 17 e 18).

- **Problemas específicos:** é relacionado a alguma atividade mais específica, assim como as desvantagens de se ter o cliente *onsite* (Problema 9) e as desvantagens da progrmação em pares (Problema 4 e 10).

- **Problemas gerenciais:** são os que lesam o trabalho dos gerentes de projeto, como a perda de controle (Problema 7), crescimento da dinâmica na gestão de questões técnicas (Problema 8) e de *releases* internas (Problema 16).

A pesquisa desenvolvida por Dybå e Dingsøyr (DYBÅ; DINGSØYR, 2008) e o vindouro aperfeiçoamento e consolidação feitos por Peterson e Wohlin (PETERSEN; WOHLIN, 2009) foram de suma importância na elaboração desta monografia. As vantagens identificadas são importantes na busca de melhrar a qualidade da construção da pesquisa, pois auxiliam no levantamento de hipóteses e a formualação da explicação dos resultados aguardados. Em contrapartida, os problemas encontrados são importantes para ajudar na elaboração do modelo de processo proposto, de modo a tentar abrandar alguns dos problemas existentes.

4 A Metodologia híbrida na literatura

(BOEHM; TURNER, 2004) visualizaram cinco fatores críticos em um projeto para deseignar o quanto o desenvolvimento do projeto tem de ser ágil ou tradicional. Os fatores descobertos abrangem o tamanho do projeto, a criticidade, o dinamismo do ambiente de desenvolvimento com relação a alterações, os recursos humanos relacionados e, por último, os fatores culturais. A obra mostra dois casos nos quais a abordagem híbrida trabalhou adequadamente e evidencia como vantagem do trabalho a escolha dos cinco fatores e a identificação do risco, fator principal para o balanceamento entre ágil e o tradicional, baseada em diversos projetos das mais diversificadas empresas.

(KARLSTROM; RUNESON, 2005) estudaram sobre a incorporação do método XP no modelo *stage gate* - modelo de gestão de projetos tomado na indústria, partido em subprojetos, em um local de desenvolvimento de sistemas, marketing, planejamento de produção etc. Este modelo prescreve estágio estabelecidos por quais um projeto tem de passar. É parecido ao modelo Cascata e ao conceito de fases no RUP, prestando um apoio à comunicação no decorrer do projeto e às tomadas de decisões por parte dos envolvidos. A expressão *gate* concerne à passagem de um estágio para outro.

O estudo de caso qualitativo criado foi alicerçado em entrevistas com profissionais de três grandes corporações, com isso os autores identificaram que os métodos ágeis concedem ferramentas para planejamento, controle da rotina de trabalho e relatórios de progresso. Ademais, as equipes conversam entre si de maneira mais efetiva quando se valem de software funcionando e reuniões pessoais, do que em documentações. Já o modelo *stage-gate* oferece aos métodos ágeis formas de coordenação de trabalho com outras equipes e de comunicação com as demais áreas.

(ILIEVA; IVANOV; STEFANOVA, 2004) notaram o fator produtividade de dois projetos parecidos, um usando os métodos tradicionais e o outro um método híbrido baseada em PSD e XP. Os projetos possuíam cerca de 900 pessoas/horas e foram desenvolvidos por duas equipes compostas de 4 pessoas que usavam tecnologia J2EE. Após três iterações de medições, os resultados apresentaram um crescimento de 42 por cento para a equipe ágil.

(ALVES et al., 2011) divulgou uma proposta de um processo híbrido denominado SCRUM-RUP, que baseou-se na integração de algumas práticas de Scrum em um processo de desenvolvimento alicerçado em RUP. O estudo de caso foi realizado em uma empresa experiente no uso de uma método customizado do RUP, e tinha como objetivo avaliar como o processo SCRUM-RUP influenciou a produtividade de desenvolvimento em comparação ao método já usado. Foi pesquisado, para efeito de comparação, seis projetos desenvolvidos usando RUP e oito utilizando SCRUM-RUP.

Para contribuir na análise de resultados, (ALVES et al., 2011) aplicou aos participantes do estudo de caso umas questões para avaliar pontos de vistas de cada profissional em relação a metodologia em análise. Um aumento na produtividade nos projetos foram identificados, só não foi possível identificar até qual ponto específico a proposta (metodologia) trouxe resultado. O autor do estudo explicou que não foi possível tirar conclusões sobre o ganho teórico do processo SCRUM-RUP quando comparado ao RUP, por causa da ausência de números maiores em relação à quantidade de projetos analisados para representarem uma amostra significativa para generalização estatística.

(COSTA; LOUREIRO; REIS, 2010) em seu trabalho desenvolvido buscou descrever uma metodologia híbrida focalizada para o desenvolvimento de software com fins didáticos (educacionais). Foi nominado de MHDCU (Metodologia Híbrida de Desenvolvimento Centrado no Utilizador). Esta metodologia foi aplicada no desenvolvimento de um jogo lúdico focado, em um primeiro momento, a alunos do primeiro e segundo ciclos do Ensino Fundamental, com o objetivo de contribuir o ensino sobre preservação da natureza e recursos naturais. No decorrer do desenvolvimento do software, ocorreu uma grande cooperação dos usuários finais (alunos e professores) nas fases de testes e validações. A equipe de desenvolvimento era compostas por profissionais das áreas de educação e tecnologia. A documentação elaborada, manuais do usuário e outros documentos, passou por diversas revisões e atualizações, a medida que era necessário.

A MHDCU teve como alicerce os seguintes elementos da metodologia ágeis: simplicidade, correção e melhoria contínua do código de software e entrega incremental. (COSTA; LOUREIRO; REIS, 2010), por fim, afirmaram que os processos iterativos e incrementais coligados aos procedimentos de prototipagens utilizados, incluindo as ferramentas de avaliação e monitorização, foram uma maneira eficaz da MHDCU se justapor às mudanças constantes de requisitos.

5 Conclusão

Depois da análise dos trabalhos referenciados anteriormente, infere-se que a adesão de uma metodologia híbrida deve levar em conta diversos pontos além dos procedimentais para obter resultados satisfatórios. (BOEHM; TURNER, 2004) atribuíram como fundamentais: o tamanho do projeto, a criticidade, a energia do ambiente de desenvolvimento com relação a modificações, os recursos humanos envolvidos e os fatores culturais. O fator de risco foi classificado como noção para o balanceamento entre práticas tradicionais e ágeis.

No meio do fatores citados alguns foram trivialmente destacados nos trabalhos dos autores que pesquisaram sobre a metodologia híbrida. Os recursos humanos tem grande importância, visto que estratégias de coordenação de trabalho ofertadas pelas metodologias tradicionais amplificadas pela maior cooperação dos participantes sejam em reuniões pessoais, partilhando conhecimento, ou interagindo diretamente com a equipe, como visto no MDHCU por (COSTA; LOUREIRO; REIS, 2010), contribuem significativamente para o sucesso com métodos híbridos. Tanto (KARLSTROM; RUNESON, 2005) quanto (ALVES et al., 2011) têm essa mesma opinião. Além do mais, acrescentaram a diminuição da documentação e o micro gerenciamento como pontos primordiais para o êxito, ratificando, assim, os elementos criticidade, adaptação às modificações e tamanho do projeto.

Por último, afere-se que usar o método híbrido estudado não trás qualquer garantia de sucesso em todos os projetos, uma vez que, devemos levar em observação o fator cultural da empresa, pois qualquer mudança costuma-se gerar uma certa resistência. Ultrapassada esta fase, o bom é planejar cuidadosamente o que adaptar de melhor das duas metodologias (tradicional e ágil) para satisfazer as necessidades reais do processo de desenvolvimento de software da empresa.

Referências

ALVES, N. M. M. et al. Integração de princípios de desenvolvimento ágil de software ao rup-um estudo empírico. Universidade Federal de Uberlândia, 2011. Citado 3 vezes nas páginas 46, 47 e 48.

AMBLER, S. Agile software development at scale. *Balancing agility and formalism in software engineering*, Springer, p. 1–12, 2008. Citado 2 vezes nas páginas 24 e 25.

BAHLI, B.; ZEID, E. A. The role of knowledge creation in adopting extreme programming model: an empirical study. In: IEEE. *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on*. [S.l.], 2005. p. 75–87. Citado na página 42.

BARBACCI, M. R. et al. *Quality Attribute Workshops (QAWs), 3rd ed., tech. report CMU/SEI-2003-TR-016*. [S.l.]: Software Engineering Institute, Carnegie Mellon University. Pittsburgh., 2003. Citado 2 vezes nas páginas 38 e 39.

BASS L.; CLEMENTS, P. K. R. *Software Architecture in Practice*. [S.l.: s.n.]. Citado 2 vezes nas páginas 38 e 39.

BASSI FILHO, D. L. Experiências com desenvolvimento ágil. *São Paulo: IME-USP, Dissertação (Mestrado em Ciências da Computação), Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo*, 2008. Citado na página 12.

BECK, K. *Extreme programming explained: embrace change*. [S.l.]: addison-wesley professional, 2000. Citado 3 vezes nas páginas 27, 30 e 37.

BECK, K. *Extreme programming explained: embrace change*. [S.l.]: 2. ed. addison-wesley professional, 2000. Citado na página 27.

BECK, K. et al. Manifesto for agile software development. 2001. Citado na página 23.

BECKER, S. A.; WHITTAKER, J. A. *Cleanroom Software Engineering Practices*. [S.l.]: IGI Global, 1997. Citado na página 22.

BOEHM, B.; TURNER, R. Rebalancing your organization's agility and discipline. *Parallel Computing Technologies*, Springer, p. 1–8, 2003. Citado na página 39.

BOEHM, B.; TURNER, R. Using risk to balance agile and plan-driven methods. *Computer*, IEEE, v. 36, n. 6, p. 57–66, 2003. Citado na página 39.

BOEHM, B.; TURNER, R. Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods. In: IEEE. *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. [S.l.], 2004. p. 718–719. Citado 9 vezes nas páginas 8, 21, 22, 23, 24, 39, 40, 46 e 48.

CANEZ, A. S. *Processo Unificado (PU) - Unified Process Fases do Processo Unificado*. 2011. Disponível em: <http://www.adonai.eti.br/wordpress/2011/04/processo-unificado-pu-unified-process/fases-pu/>. Acesso em: 21 jul. 2017. Citado na página 29.

CARVALHO, W. C. d. S. et al. Análise dos efeitos do turnover na produtividade de processos de software tradicionais e híbridos. Universidade Federal de Uberlândia, 2012. Citado na página 12.

COCKBURN, A. *Crystal clear: a human-powered methodology for small teams*. [S.l.]: Pearson Education, 2004. Citado na página 27.

COHEN, D.; LINDVALL, M.; COSTA, P. An introduction to agile methods. *Advances in computers*, Elsevier, v. 62, p. 1–66, 2004. Citado 2 vezes nas páginas 43 e 44.

COOPER, R. G. *Winning at new products*. [S.l.]: Perseus Publishing, Cambridge, MA, 2001. Citado na página 41.

COSTA, A. P.; LOUREIRO, M. J.; REIS, L. P. Metodologia híbrida de desenvolvimento centrado no utilizador aplicada ao software educativo. *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informação (AISTI), n. 6, p. 1–16, 2010. Citado 2 vezes nas páginas 47 e 48.

DYBA, T. Improvisation in small software organizations. *IEEE Software*, IEEE, v. 17, n. 5, p. 82–87, 2000. Citado na página 23.

DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Information and software technology*, Elsevier, v. 50, n. 9, p. 833–859, 2008. Citado 4 vezes nas páginas 37, 38, 41 e 45.

DYBA, T.; DINGSOYR, T. What do we know about agile software development? *IEEE software*, IEEE, v. 26, n. 5, p. 6–9, 2009. Citado 2 vezes nas páginas 26 e 37.

FILHO, W. de P. P. *Engenharia de software*. [S.l.]: LTC, 2003. v. 2. Citado na página 15.

FOWLER, M. The new methodology. *Wuhan University Journal of Natural Sciences*, Springer, v. 6, n. 1, p. 12–24, 2001. Citado na página 30.

GALAL-EDEEN, G.; RIAD, A.; SEYAM, M. Agility versus discipline: Is reconciliation possible? In: IEEE. *Computer Engineering & Systems, 2007. ICCES'07. International Conference on*. [S.l.], 2007. p. 331–337. Citado na página 37.

HILKKA, M.-R.; TUURE, T.; ROSSI, M. Is extreme programming just old wine in new bottles: A comparison of two cases. *Journal of Database Management*, IGI Global, v. 16, n. 4, p. 41, 2005. Citado na página 44.

HUMPHREY, W. S. *Introduction to the personal software process (sm)*. [S.l.]: Addison-Wesley Professional, 1996. Citado na página 23.

HUMPHREY, W. S. *Introduction to the team software process*. [S.l.]: Addison-Wesley Professional, 2000. Citado na página 23.

ILIEVA, S.; IVANOV, P.; STEFANOVA, E. Analyses of an agile methodology implementation. In: IEEE. *Euromicro Conference, 2004. Proceedings. 30th*. [S.l.], 2004. p. 326–333. Citado 3 vezes nas páginas 42, 44 e 46.

ILLUSTRATING Scrum - A new and improved Scrum Diagram 3moths Blog. 2010. Disponível em: <http://blog.3months.com/2010/01/10/illustrating-scrum-a-new-and-improved-scrum-diagram/>. Acesso em: 29 jul. 2017. Citado na página 33.

KARLSTROM, D.; RUNESON, P. Combining agile methods with stage-gate project management. *IEEE software*, IEEE, v. 22, n. 3, p. 43–49, 2005. Citado 3 vezes nas páginas 41, 46 e 48.

KARLSTRÖM, D.; RUNESON, P. Integrating agile software development into stage-gate managed product development. *Empirical Software Engineering*, Springer, v. 11, n. 2, p. 203–225, 2006. Citado 4 vezes nas páginas 40, 42, 43 e 44.

KNIBERG, H. Scrum e xp direto das trincheiras. *Estocolmo: C4Media Inc*, 2007. Citado na página 30.

LARMAN, C. *Utilizando UML e padrões*. [S.l.]: Bookman Editora, 2002. Citado na página 28.

LARMAN, C. *Agile and iterative development: a manager's guide*. [S.l.]: Addison-Wesley Professional, 2004. Citado na página 29.

LEFFINGWELL, D. *Scaling Software Agility*. [S.l.]: Addison Wesley, 2006. Citado 2 vezes nas páginas 21 e 25.

LI, J.; MOE, N. B.; DYBÅ, T. Transition from a plan-driven process to scrum: a longitudinal case study on software quality. In: ACM. *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*. [S.l.], 2010. p. 13. Citado na página 43.

MACKENZIE, A.; MONK, S. From cards to code: How extreme programming re-embodies programming as a collective practice. *Computer Supported Cooperative Work (CSCW)*, Springer, v. 13, n. 1, p. 91–117, 2004. Citado na página 44.

MAINART, D. d. A.; SANTOS, C. M. *Desenvolvimento de Software: processos ágeis ou tradicionais? uma visão crítica*. Dissertação (Mestrado) — Faculdade Presidente Antônio Carlos de Teófilo Otoni; Universidade Federal dos Vales do Jequitinhonha e Mucuri - UFVJM, Teófilo Otoni – MG, 2010. Disponível em: <http://blog.3months.com/2010/01/10/illustrating-scrum-a-new-and-improved-scrum-diagram/>. Acesso em: 29 jun. 2017. Citado na página 12.

MANN, C.; MAURER, F. A case study on the impact of scrum on overtime and customer satisfaction. In: IEEE. *Agile Conference, 2005. Proceedings*. [S.l.], 2005. p. 70–79. Citado na página 42.

MANNARO, K.; MELIS, M.; MARCHESI, M. Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies. *Extreme programming and agile processes in software engineering*, Springer, p. 166–174, 2004. Citado na página 42.

MARTIN, A.; BIDDLE, R.; NOBLE, J. When xp met outsourcing. *Extreme Programming and Agile Processes in Software Engineering*, Springer, p. 51–59, 2004. Citado na página 44.

MCBREEN, P.; BY-BECK, K. F. *Questioning extreme programming*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. Citado na página 43.

MELNIK, G.; MAURER, F. Perceptions of agile practices: A student survey. *Extreme Programming and Agile Methods—XP/Agile Universe 2002*, Springer, p. 103–113, 2002. Citado na página 44.

MOTA, R. L.; LIMA, P. B.; ROMANO, B. L. Um modelo para definição de metodologia de desenvolvimento de software baseado em pessoas. 2011. Disponível em: <http://www.cafw.ufsm.br/eati/2011/anais/artigos/91321.pdf>. Acesso em: 29 abr. 2017. Citado 2 vezes nas páginas 11 e 12.

NERUR, S.; MAHAPATRA, R.; MANGALARAJ, G. Challenges of migrating to agile methodologies. *Communications of the ACM*, ACM, v. 48, n. 5, p. 72–78, 2005. Citado na página 23.

NORD, R.; TOMAYKO, J. E.; WOJCIK, R. Integrating software-architecture-centric methods into extreme programming (xp). 2004. Citado na página 38.

NORD, R. L.; TOMAYKO, J. E. Software architecture-centric methods and agile development. *IEEE software*, IEEE, v. 23, n. 2, p. 47–53, 2006. Citado na página 38.

PALMER, S. R.; FELSING, M. *A practical guide to feature-driven development*. [S.l.]: Pearson Education, 2001. Citado na página 28.

PETERSEN, K.; WOHLIN, C. Context in industrial software engineering research. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement*. [S.l.], 2009. p. 401–404. Citado 5 vezes nas páginas 41, 42, 43, 44 e 45.

PIKKARAINEN, M. et al. The impact of agile practices on communication in software development. *Empirical Software Engineering*, Springer, v. 13, n. 3, p. 303–337, 2008. Citado 3 vezes nas páginas 41, 42 e 44.

POPPENDIECK, M.; POPPENDIECK, T. *Lean Software Development: An Agile Toolkit: An Agile Toolkit*. [S.l.]: Addison-Wesley, 2003. Citado na página 28.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: Makron books Sao Paulo, 1995. v. 6. Citado na página 14.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: Makron books Sao Paulo, 2006. v. 6. Citado 2 vezes nas páginas 11 e 32.

PROWELL, S. J. et al. *Cleanroom software engineering: technology and process*. [S.l.]: Pearson Education, 1999. Citado na página 22.

PÁDUA FILHO, W. *Engenharia de Software*. [S.l.]: 3. ed. Rio de Janeiro: LTC, 2009. Citado 8 vezes nas páginas 8, 16, 17, 18, 19, 20, 25 e 35.

RAMSIN, R.; PAIGE, R. F. Process-centered review of object oriented software development methodologies. *ACM Computing Surveys (CSUR)*, ACM, v. 40, n. 1, p. 3, 2008. Citado 3 vezes nas páginas 24, 37 e 43.

ROBINSON, H.; SHARP, H. The characteristics of xp teams. In: SPRINGER. *International Conference on Extreme Programming and Agile Processes in Software Engineering*. [S.l.], 2004. p. 139–147. Citado na página 42.

ROMBACH, D.; SEELISCH, F. Formalisms in software engineering: Myths versus empirical facts. *Lecture notes in computer science*, Springer, v. 5082, p. 13–25, 2008. Citado na página 37.

ROSENBERG, D.; STEPHENS, M. *Extreme programming refactored: the case against XP*. [S.l.]: Apress, 2003. Citado na página 43.

ROYCE, W. W. Managing the development of large software systems: concepts and techniques. In: IEEE COMPUTER SOCIETY PRESS. *Proceedings of the 9th international conference on Software Engineering*. [S.l.], 1987. p. 328–338. Citado na página 11.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. *Unified modeling language reference manual, the*. [S.l.]: Pearson Higher Education, 2004. Citado na página 16.

SCHWABER, K. Scrum development process. In: *Business object design and implementation*. [S.l.]: Springer, 1997. p. 117–134. Citado na página 26.

SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2001. v. 1. Citado na página 26.

SEI, S. *Capability Maturity Model Integration*. Dissertação (Mestrado), 2002. Disponível em: <http://www.sei.cmu.edu/cmmi>. Acesso em: 30 set. 2017. Citado 2 vezes nas páginas 23 e 34.

SEI, S. Cmmi® for development (cmmidev), v1. 2, cmu/sei-2006-tr-008. *Software Engineering Institute*, 2006. Citado 2 vezes nas páginas 23 e 34.

SOMMERVILLE, I. Engenharia de software, 8ª edição, tradução: Selma shin shimizu mel-nikoff, reginaldo arakaki, edilson de andrade barbosa. *São Paulo: Pearson Addison-Wesley*, v. 22, p. 103, 2007. Citado 2 vezes nas páginas 12 e 14.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2003. v. 6. Citado 2 vezes nas páginas 14 e 15.

STAPLETON, J. *DSDM: Business focused development*. [S.l.]: Pearson Education, 2003. Citado na página 26.

SVENSSON, H.; HOST, M. Introducing an agile process in a software maintenance and evolution organization. In: IEEE. *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*. [S.l.], 2005. p. 256–264. Citado 2 vezes nas páginas 42 e 43.

TESSEM, B. Experiences in learning xp practices: A qualitative study. *Extreme programming and agile processes in software engineering*, Springer, p. 1012–1012, 2003. Citado 2 vezes nas páginas 42 e 44.

VALENTE, P. *Product Ower na prática*. Dissertação (Mestrado), 2010. Disponível em: <<http://www.slideshare.net/pedrovalente/product-owner-na-prtica>. Acesso em: 15 mai. 2017. Citado na página 34.

WILLIAMS, L.; COCKBURN, A. Guest editors' introduction: Agile software development: It's about feedback and change. *Computer*, IEEE Computer Society Press, v. 36, n. 6, p. 39–43, 2003. Citado na página 23.

YIN, R. K. *Estudo de Caso-: Planejamento e Métodos*. [S.l.]: Bookman editora, 2015. Citado na página 41.