Perquirere, 11(2): 130-140, dez. 2014
© Centro Universitário de Patos de Minas
http://perquirere.unipam.edu.br

Análise do paradigma híbrido na indústria de *software*

Hybrid paradigm of analysis in the software industry

Kéllyson Gonçalves da Silva
Especialista em Engenharia de Software pelo Centro Universitário de Patos de Minas (UNIPAM).
E-mail: kellyson.si@hotmail.com

William Chaves de Souza Carvalho
Mestre e Doutorando em Ciência da Computação pela Universidade Federal de Uberlândia (UFU).
E-mail: william@facom.ufu.br

Resumo: Este artigo analisa as metodologias híbridas de desenvolvimento de software baseadas em publicações de autores que já contribuíram com a comunidade científica e acadêmica através de pesquisas e estudos de casos abordando o tema. Primeiramente, aborda-se uma série de acontecimentos, em ordem cronológica, que vão desde as primeiras soluções pensadas para fazer frente à crise de software até o surgimento dos métodos híbridos. Em seguida, são detalhadas algumas metodologias mais conhecidas, tanto tradicionais, quanto ágeis. Feito isso, o uso dos paradigmas híbridos é descrito, e, por fim, o artigo é concluído com a análise dos trabalhos analisados.
Palavras-chave: Engenharia de Software. Metodologia Híbrida. Desenvolvimento de Software.

Abstract: This paper analyzes the hybrid methodologies of software development based on publications of authors who have contributed to the scientific and academic community through research and case studies that approach the topic. First, it addresses a series of events in chronological order, from the first solutions designed to confront the crisis of software until the emergence of the hybrid methods. Then, some methodologies already known are detailed, both traditional and agile. After that, the use of hybrid paradigms is described, and finally, the paper is concluded with the analysis of the works analyzed.
Keywords: Software Engineering. Hybrid Methodology. Software Development.

---

1 INTRODUÇÃO

A necessidade por melhoria contínua dos processos de desenvolvimento de software nas organizações é uma realidade desde a década de 1960. A Crise do Software, em 1970, evidenciou essa necessidade, em que o aumento da demanda por aplicações de qualidade começou a surgir, porém os profissionais de tecnologia tiveram problemas relacionados à gestão, além de fatores como a má interpretação ou a não compreensão dos requisitos envolvidos nos projetos, o que gerou, como consequências, códigos de difícil manutenção, extrapolação de prazos e orçamentos, desgastes entre clientes e fornecedores.

De acordo com Mota *et al.* (2011), a Engenharia de Software surgiu para resolver os problemas da crise. Baseando-se em modelos industriais, foram criadas as metodologias de desenvolvimento que organizavam de uma maneira profissional o modo de desenvolver softwares, dividindo-o em etapas, acompanhadas de uma série de documentos para especificar ao cliente o que seria desenvolvido, visando sempre à qualidade final do produto. Destacam-se como exemplos de métodos tradicionais: Cascata, Prototipação, RAD e Espiral.

Mainart *et al.* (2010) afirmam que o processo de desenvolvimento de software é bastante mutável, pois sempre há o surgimento de novos requisitos, sejam funcionais ou não funcionais por parte do cliente, e, utilizando o processo tradicional, é necessário alterar documentação e o produto em si, frequentemente ocorrendo casos de fracassos ao final do período de tempo estipulado nas fases iniciais do projeto.

Percebendo os problemas decorrentes do uso das metodologias tradicionais, 17 líderes experientes adotaram modos de trabalhos contrários aos principais conceitos dos métodos tradicionalistas em vários projetos e constataram que os novos meios eram eficientes. Em 2001, esses líderes se reuniram e criaram o Manifesto Ágil, e as premissas resultantes dessa reunião (Indivíduos e iterações, Software funcionando, Colaboração com o cliente e Adaptação a mudanças) deram origem a novas metodologias para desenvolvimento de software (BASSI FILHO, 2008).

As metodologias ágeis propõem a obtenção de resultados práticos em tempo inferior às utilizadas pelos processos tradicionais utilizados pelo mercado de desenvolvimento de software; elas pretendem fazer isso tirando o foco do processo e colocando-o no produto. Dessa forma, os métodos ágeis se propõem a dispensar ou modificar as etapas e a forma como os envolvidos realizam suas tarefas (BASSI FILHO, 2008). Destacam-se como exemplos de paradigmas ágeis os métodos SCRUM e XP.

Os paradigmas tradicionais e ágeis possuem pontos fortes e fracos. O paradigma tradicional é recomendado para projetos de larga escala e alto risco e embasa-se na Análise e Projeto, com documentação abrangente, porém é lento para mudanças. Já os métodos baseados no paradigma ágil, são recomendados para projetos de baixo risco e de equipe e tamanhos pequenos. Baseia-se em código, adaptável a mudanças de requisitos, porém fraco na parte contratual e de documentos (CARVALHO *et al.*, 2011). O que não se discute é que, por alterarem características importantes dos métodos tradicionais, os paradigmas ágeis tornaram-se polêmicos e não inspiraram confiança nos profissionais mais conservadores.

Finalmente, os métodos híbridos surgiram quando os dois paradigmas foram questionados pela indústria de software, afirmando que abordagens seguidas especificamente não satisfazem de forma adequada a cultura das empresas. O conceito da metodologia híbrida, fusão dos processos ágeis e tradicionais, vem com o propósito de solucionar esses questionamentos (MOTA *et al.*, 2011).

Esse artigo consiste na análise, através de trabalhos já publicados, de como o desenvolvimento híbrido contribuiu com as instituições, empresas ou equipes de desenvolvimento que o aderiram.

2 UM POUCO SOBRE METODOLOGIAS

Para compreender o conceito de metodologia híbrida, é necessário conhecer sobre os métodos antecessores, tradicionais e ágeis.

2.1 O MODELO CASCATA

O modelo cascata requer uma abordagem sistemática, sequencial ao desenvolvimento do software, que se inicia no nível do sistema e avança ao longo da análise, projeto (ou design), codificação, teste e manutenção (PRESMAN, 1995). O processo de software não é um modelo linear simples, mas envolve uma sequência de iterações das atividades de desenvolvimento (SOMMERVILLE, 2003). O resultado de cada fase envolve um ou mais documentos que são aprovados. A fase seguinte não deve se iniciar até que a fase antecessora tenha sido concluída. Na prática, esses estágios se sobrepõem e trocam informações entre si. Durante o projeto, são identificados problemas com os requisitos; na codificação, são verificados problemas de projeto, e assim por diante.

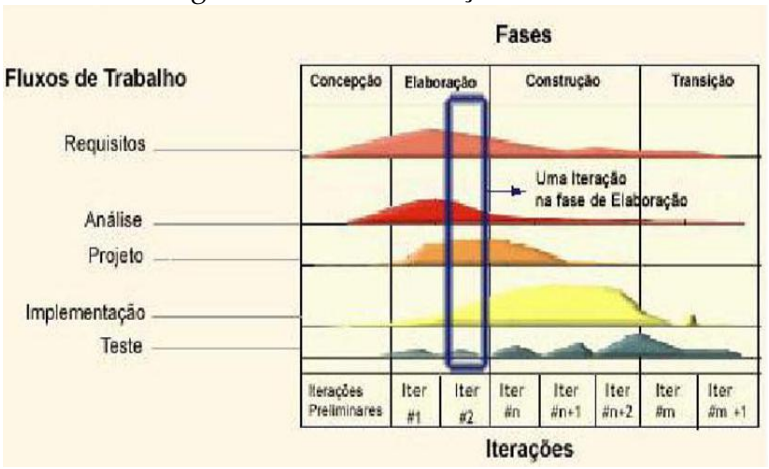
As vantagens do modelo em cascata consistem em identificar todos os requisitos e analisá-los antes da programação começar e minimizar as alterações feitas nos mesmos, à medida que o projeto avança. A principal desvantagem deste modelo consiste no fato da especificação do projeto inteiro precisar estar pronta antes da etapa do início da programação, além de transcorrer longo tempo entre o proposto do sistema e a entrega do mesmo.

2.2 PROCESSO UNIFICADO

O Processo Unificado é um exemplo de processo iterativo para projetos que utilizam a Programação Orientada a Objetos (POO). O desenvolvimento iterativo é organizado em uma série de miniprojetos, de duração fixa, chamados iterações; o produto de cada consiste em um sistema testado, integrado e executável. Cada iteração inclui suas próprias atividades de análise de requisitos, projeto, implementação e teste (LARMAN, 2004).

As melhores práticas do PU incluem: desenvolver softwares iterativamente, gerenciar requisitos, usar arquiteturas baseada em componentes, modelar o software visualmente através dos diagramas da UML (*Unified Model Language*) e verificar a qualidade do software.

Figura 1 – Fases e Iterações do PU



Fonte: CANEZ, 2011

A Figura 1 apresenta as fases do Processo Unificado e, ao mesmo tempo, chama a atenção do leitor para o conceito de iteração (ciclos). Durante cada ciclo, todos os fluxos de trabalho (Requisitos, Análise, Projeto, Implementação e Teste) estão em execução. Cada fluxo possuiu maior intensidade durante as seguintes fases do PU:

- Requisitos: fases de Concepção e Elaboração;
- Análise: principalmente na fase de Elaboração;
- Projeto: fases de Elaboração e Construção;
- Implementação: principalmente na fase de Construção;
- Teste: principalmente nas fases de Construção e Transição.

Larman (2004) afirma que o ciclo de vida iterativo é baseado em refinamentos e incrementos de um sistema por meio de múltiplas iterações, com realimentação (*feedback*) e adaptação cíclicas como principais propulsores para convergir para um sistema adequado. O sistema cresce incrementalmente ao longo do tempo, iteração por iteração, razão pela qual esta abordagem é conhecida como desenvolvimento iterativo e incremental.

2.3 PROGRAMAÇÃO EXTREMA – XP

A XP é uma metodologia leve que utiliza processo ágil, com isso, deve-se fazer um estudo minucioso e detalhar a arquitetura antes de começar. Ao contrário do PU, a XP foge da documentação formal e confia mais na oral, mas ambas as metodologias são abordagens de desenvolvimento incremental e iterativo.

Essa metodologia possui seis fases: levantamento de requisitos, análise, desenho da arquitetura, implementação, teste e manutenção. O processo de fatoração contínua é aplicado com frequência. A XP é recomendada para pequenos e médios projetos, que podem variar de 1 a 36 meses, indicado, também, em casos de agenda curta e prazos críticos. Em média, a equipe de programadores que adotam XP é composta de 2 a 10 pessoas.

2.4 SCRUM

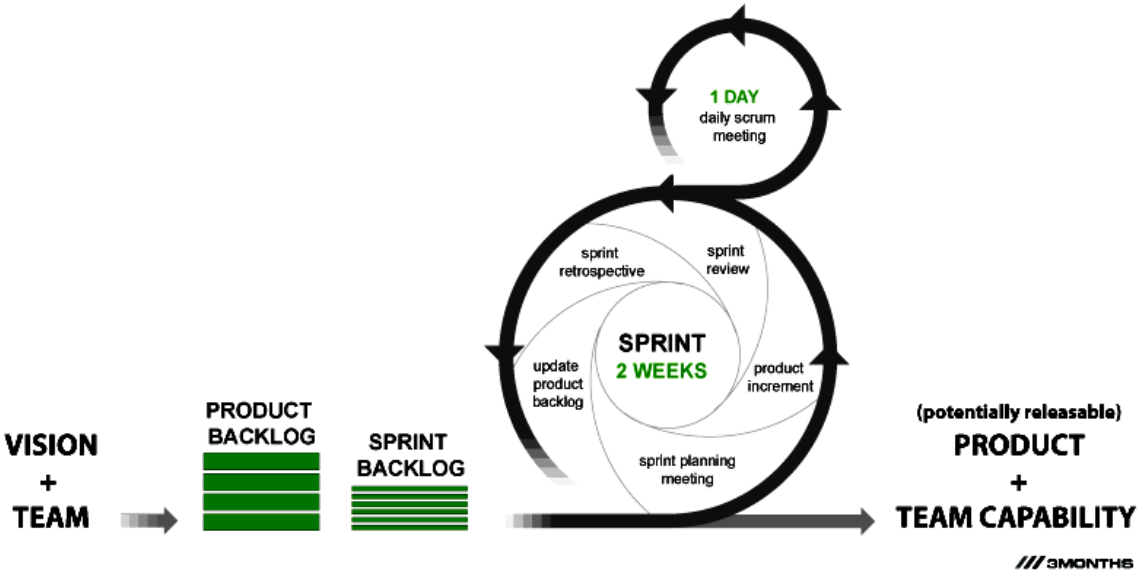
O Scrum é um método ágil de desenvolvimento, surgido no início dos anos 90, que funciona de forma empírica e visa a menor quantidade de documentação possível. Não rejeita ferramentas, processos, documentação, contratos ou planejamentos, mas suas prioridades são os indivíduos e iterações, a executabilidade do software, a colaboração e os *feedbacks*. O Scrum não segue padrões propostos pelas metodologias tradicionais, porém não é contra. Possui ênfase na comunicação, no trabalho em equipe, na flexibilidade e no trabalho incremental.

Os objetivos principais do Scrum são garantir maior flexibilidade, habilidade para tratamento de sistemas complexos e simples, produzir sistemas sujeitos a requisitos iniciais e adicionais durante o projeto. Isso é possível devido ao fato que em todo final de um ciclo de desenvolvimento, também conhecido como Sprint, obtém-se um produto executável e testado. Essa metodologia é dividida em três fases: Planejamento, Sprints (ou Ciclos) e Encerramento.

Na fase de planejamento (*backlog*), são definidos os processos, o design da arquitetura do sistema, as equipes e seus líderes, os pacotes a serem desenvolvidos. Neste caso, há a participação dos clientes e de outros departamentos, ocorrendo o levantamento de requisitos e atribuição de prioridade dos mesmos. O cliente, na metodologia Scrum, é similar ao do método XP, em que o mesmo é integrante fundamental da equipe de desenvolvimento, para que haja maior contato entre quem necessita do serviço e quem desenvolverá. No Sprint, cada time recebe uma parte do *backlog* para desenvolver e este, por sua vez, não sofre modificações durante o desenvolvimento. Cada Sprint pode durar de uma a quatro semanas e, no final deste período, sempre é apresentado um executável.

Durante a segunda fase, de acordo com Pressman (2006), são realizadas reuniões diárias, coordenadas pelos líderes de cada equipe, essa reunião tem duração de no máximo 15 minutos, em que todos os desenvolvedores respondem a três perguntas: "O que você realizou desde a última reunião?", "Quais problemas você enfrentou?" e "Em que você trabalhará até a próxima reunião?". Por meio dessas reuniões, alguns benefícios podem ser destacados, como maior integração entre os membros da equipe, rápida solução de problemas, compartilhamento de conhecimento, e o progresso do desenvolvimento é medido continuamente, ocorrendo, assim, uma minimização significativa dos riscos. A Figura 2 destaca a fase de Sprint, na qual ocorre o planejamento (como será desenvolvido?), a implementação do produto, a revisão, a retrospectiva do mesmo e a atualização do *product backlog*. Nota-se, também, a representação das reuniões diárias, que são fundamentais nessa metodologia.

Figura 2 – Sprint (ciclos de desenvolvimento)

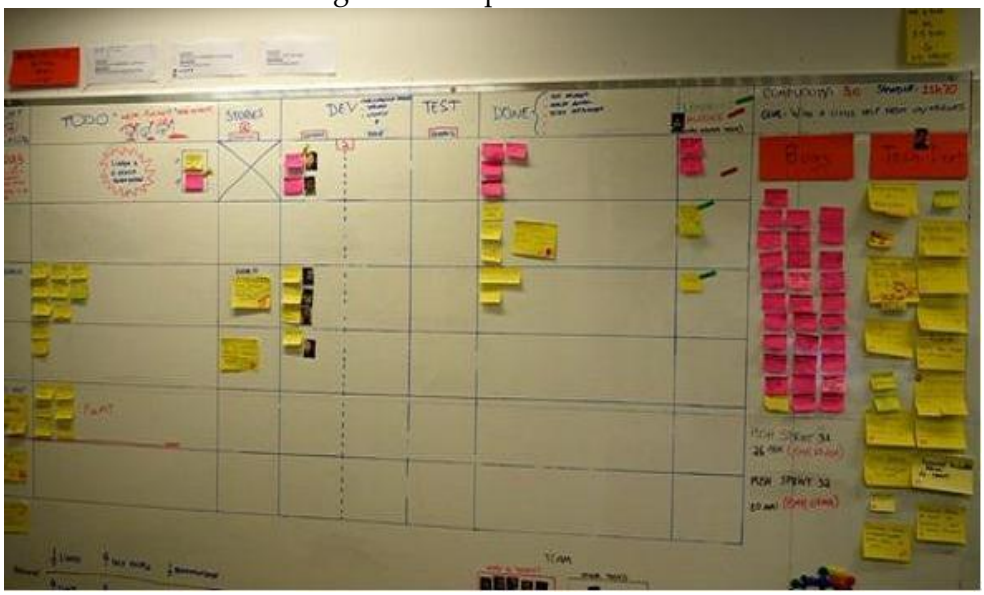


Fonte: ILLUSTRATING SCRUM, 2010.

No período de revisão da Sprint, a data de entrega deve ser obedecida, para que ocorra a apresentação do produto para o cliente e demais *stakeholders*, com isso sugestões de mudanças são incorporadas ao *backlog*. A revisão traz alguns benefícios, tais como: apresentação de resultados concretos ao cliente, integração e testes de uma boa parte do software, além de motivação para a equipe.

Para auxiliar no processo de criação, o Scrum adota um quadro, denominado Kanban, por meio do qual é possível saber quem está desenvolvendo o quê, quais os problemas estão ocorrendo, o que há ainda a ser feito, o que já está pronto e o que já foi testado ou que está em fase de testes. Além disso, pode-se acompanhar detalhadamente o processo de criação, controlando, assim, o tempo e evitando que duas ou mais pessoas façam uma mesma tarefa. As tarefas e os *bugs* (erros, problemas) são diferenciados por cores para facilitar a leitura e o entendimento do quadro. A Figura 3 ilustra um quadro Kanban, no qual é possível observar que cada coluna representa uma etapa de desenvolvimento. Durante a fase de desenvolvimento, um membro da equipe escolhe uma tarefa para ser desenvolvida, remove-a da coluna "para fazer (*TO DO*)" e a coloca "em desenvolvimento (*DEV*)". Após o término, são efetuados os testes, se tudo estiver satisfatório, a tarefa é concluída e deslocada para a coluna "feito (*DONE*)", senão o problema é registrado na sessão de bugs e a atividade volta para a coluna "em desenvolvimento (*DEV*)". Todo esse processo é realizado até que toda aplicação no final fique completa.

Figura 3 – O quadro Kanban



Fonte: VALENTE, 2010

A última fase do Scrum, o Encerramento, só é iniciada quando todos os aspectos (tempo, competitividade, requisitos, qualidade e custo) são satisfatórios. É nesta etapa que ocorrem os testes de integração, testes de sistema, documentação do usuário, preparação de material de treinamento e do material de marketing.

3 A METODOLOGIA HÍBRIDA NA LITERATURA

Boehm *et al.* (2004) identificaram cinco fatores críticos em um projeto para determinar o quanto o desenvolvimento do mesmo deve ser ágil ou dirigido por plano (tradicional). Os fatores identificados abrangem o tamanho do projeto, a criticidade, o dinamismo do ambiente de desenvolvimento com relação a alterações, os recursos humanos (pessoal) envolvidos e, por fim, os fatores culturais. A obra apresenta dois casos nos quais a abordagem híbrida funcionou adequadamente e destaca como vantagem do trabalho a escolha dos cinco fatores e a identificação do risco, fator principal para o balanceamento entre o ágil e o tradicional, baseada em vários projetos de diversas organizações.

Karlström e Runeson (2005) estudaram sobre a integração do método XP no modelo *stage gate* – modelo de gestão de projetos adotado na indústria, dividido em subprojetos, em um ambiente de desenvolvimento de sistemas, marketing, planejamento de produção etc. O modelo prescreve estágios definidos, pelos quais um projeto deve passar. É similar ao modelo Cascata e ao conceito de fases no RUP, dando suporte à comunicação durante o projeto e às tomadas de decisões por parte dos envolvidos. O termo *gate* refere-se à passagem de um estágio para outro.

O estudo de caso qualitativo elaborado foi baseado em entrevistas com profissionais de três grandes empresas, com isso os autores perceberam que os métodos ágeis oferecem ferramentas para planejamento, controle da rotina de trabalho e relatórios de progresso. Além disso, as equipes se comunicam de forma mais efetiva

quando se valem de software funcionando e reuniões pessoalmente, do que em documentações. Já o modelo *state-gate* oferece aos paradigmas ágeis formas de coordenação de trabalho com outras equipes e de comunicação com as demais áreas.

Ilieva *et al.* (2004) observaram o fator produtividade de dois projetos similares, um utilizando metodologias tradicionais e o outro um método híbrido baseado em PSP e XP. Os projetos possuíam tamanho aproximado de 900 pessoas-horas e foram desenvolvidos por duas equipes compostas de 4 pessoas utilizando tecnologia J2EE. Os resultados, após três iterações de medições, mostraram um aumento de 42% em produtividade [considerando, LOC (Line of Code)/hora como unidade de medida] para a equipe ágil.

Alves (2011) apresentou uma proposta de um processo híbrido denominado SCRUM-RUP, que consistiu na integração de algumas práticas de Scrum em um processo de desenvolvimento baseado em RUP. O estudo de caso foi realizado em uma empresa localizada na cidade de Uberlândia-MG, experiente na utilização de um método customizado do RUP (a partir daqui, será designado somente o termo RUP), e tinha como intuito avaliar como o processo SCRUM-RUP impactou a produtividade de desenvolvimento em comparação ao paradigma já utilizado. Foram analisados, para comparação, seis projetos desenvolvidos utilizando RUP e oito utilizando SCRUM-RUP. Os profissionais envolvidos, funcionários da empresa, em sua maioria possuíam um grau de senioridade mais alta (plenos e seniores).

Para auxiliar na análise de resultados, Alves (2011) aplicou aos participantes do estudo um questionário para coletar pontos de vista de cada profissional participante a respeito da metodologia em questão. Os resultados da experiência apontaram que houve aumento de produtividade nos projetos realizados com o método proposto, só não esclareceram até qual marco o processo foi responsável pelo sucesso. O autor do estudo afirmou que não se pode inferir conclusões sobre o ganho teórico do processo SCRUM-RUP em comparação ao RUP, devido à falta de números maiores em relação à quantidade de projetos analisados para representarem uma amostra significativa para generalização estatística. Além do processo, os fatores predominantes nas metodologias ágeis (maior colaboração entre os participantes, diminuição da documentação e micro gerenciamento) exerceram papel significativo pelo parecer satisfatório.

O trabalho realizado por Costa *et al.* (2010) visou descrever uma metodologia híbrida voltada para o desenvolvimento de um software para fins educacionais. O método foi denominado MHDCU (Metodologia Híbrida de Desenvolvimento Centrado no Utilizador). A metodologia foi utilizada na elaboração de um jogo lúdico (*Courseware Sere*) direcionado, primeiramente, a alunos do primeiro e segundo ciclos do Ensino Básico, com o propósito de facilitar o ensino sobre preservação da natureza e recursos naturais. Durante o processo de criação, houve grande participação e interação dos usuários finais, tanto alunos quanto professores, na fase de testes e validações, o que fez bastante jus ao nome do paradigma em questão. A equipe de desenvolvimento foi formada por profissionais das áreas de educação e tecnologia, desde designers gráficos e gerentes de projetos até programadores e testadores. A documentação elaborada, seja o conjunto de manuais de usuário e outros documentos,

passou por constantes revisões e atualizações, na medida em que novos requisitos surgiam ou quando problemas eram detectados durante os testes.

A MHDCU teve como base os seguintes princípios dos métodos ágeis: simplicidade, correção e melhoria contínua do código do software e entrega incremental. Costa *et al.* (2010), por fim, afirmaram que os processos iterativos e incrementais associados aos procedimentos de prototipagens utilizados, incluindo as ferramentas de avaliação e monitorização, foram uma forma eficiente da MHDCU se adaptar às mudanças contínuas de requisitos.

4 CONCLUSÃO

Após a análise dos trabalhos referenciados anteriormente, conclui-se que a adoção de uma metodologia híbrida deve levar em consideração vários fatores além dos procedimentais (Requisitos, Divisão de Papéis, Erros/Falhas, Qualidade e Documentação) para obter resultados satisfatórios. Boehm *et al.* (2004) atribuíram como principais: o tamanho do projeto, a criticidade, o dinamismo do ambiente de desenvolvimento com relação a alterações, os recursos humanos envolvidos e os fatores culturais. O fator risco foi considerado como referência para o balanceamento entre práticas tradicionais e ágeis.

Dentre os fatores mencionados alguns foram comumente destacados nos trabalhos dos autores que pesquisaram sobre desenvolvimento híbrido. Os recursos humanos exercem forte influência, visto que as estratégias de coordenação de trabalho oferecidas pelos métodos tradicionalistas acrescidas da maior colaboração dos participantes sejam em reuniões face a face, compartilhando conhecimento, ou interagindo diretamente com a equipe, como visto na aplicação do MDHCU por Costa *et al.* (2010), contribuem significativamente para obter sucesso com métodos híbridos. Tanto Karlström *et al.* (2005) quanto Alves (2011) têm essa mesma opinião. Além disso, acrescentaram a diminuição da documentação e o micro gerenciamento como pontos principais para o êxito, reforçando, assim, os fatores criticidade, adaptação às alterações e tamanho do projeto.

Por fim, conclui-se que utilizar método híbrido não garante sucesso em todos os projetos, pois, em primeiro lugar, deve-se levar em consideração o fator cultural da organização que pretende adotá-lo, porque qualquer mudança gera resistência. Vencida esta etapa, o ideal é planejar criteriosamente o que adaptar de melhor dos dois paradigmas (tradicional e ágil) para satisfazer as necessidades reais do processo de desenvolvimento de software da organização.

REFERÊNCIAS

ALVES, Nelio Muniz Mendes. *Integração de princípios de desenvolvimento ágil de software ao RUP – um estudo empírico*. 138 f. Tese (Doutorado em Ciências da Computação) - Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia – UFU, Uberlândia, 2011.

BASSI FILHO, Dairton Luiz. *Experiências com desenvolvimento ágil.* 170 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Matemática e Estatística da Universidade de São Paulo. USP, São Paulo, 2008.

BOEHM, B.; TURNER, R. *Balancing agility and discipline:* a guide for the perplexed. Reading: Adisson-Wesley, 2004.

CANEZ, Adonai Silveira. *Processo Unificado (PU) - Unified Process Fases do Processo Unificado.* Disponível em: <http://www.adonai.eti.br/wordpress/2011/04/processo-unificado-pu-unified-process/fases-pu/>. Acesso em: 13 de Maio de 2013.

CARVALHO, W. C. S. *et al*. Um estudo sobre produtividade na adoção de um processo híbrido de desenvolvimento de software. *Revista do CCEI*, v. 15, p. 1, 2011.

COSTA, António Pedro; LOUREIRO, Maria João; REIS, Luís Paulo. Metodologia Híbrida de Desenvolvimento Centrado no Utilizador aplicada ao Software Educativo. *Revista Ibérica de Sistemas e Tecnologias de Informação*, p. 10, 2010.

ILIEVA, S.; IVANOV, P.; STEFANOVA, E. *Analyses of an Agile Methodology Implementation.* Euromicro Conference. Rennes, France: IEEE Computer Society Press. 2004.

Illustrating Scrum - A new and improved Scrum Diagram 3moths Blog. Disponível em: <http://blog.3months.com/2010/01/10/illustrating-scrum-a-new-and-improved-scrum-diagram/> Acesso em: 22 de Maio de 2013.

KARLSTRÖM, D.; RUNESON, P. *Combining Agile Methods with Stage-Gate Project Management.* IEEE Software. v. 22. n. 3. p. 43-49, 2005.

LARMAN, Craig. *Utilizando UML e padrões:* uma introdução à análise e ao projeto orientados a objetos a o processo unificado. 2. ed. Porto Alegre: Bookman, 2004.

MAINART, Domingos de A.; SANTOS, Ciro M. *Desenvolvimento de Software:* processos ágeis ou tradicionais? Uma visão crítica**.** Faculdade Presidente Antônio Carlos de Teófilo Otoni; Universidade Federal dos Vales do Jequitinhonha e Mucuri - UFVJM, Teófilo Otoni - MG, 2010. Disponível em: <http://www.enacomp.com.br/2010/cd/artigos/completos/enacomp2010_4.pdf >. Acesso em: 7 de Agosto de 2013.

MOTA, Rodrigo L. M.; LIMA, Pablo B. S.; ROMANO; Breno L. *Um modelo para definição de metodologia de desenvolvimento de software baseado em pessoas.* Departamento de Computação e Matemática - Instituto de Ciências Exatas - Universidade Federal de Itajubá (UNIFEI), Itajubá - MG, 2011. Disponível em: <http://www.cafw.ufsm.br/eati/2011/anais/artigos/91321.pdf >. Acesso em: 4 de Setembro de 2013.

PRESSMAN, Roger S. *Engenharia de software.* 3. ed. São Paulo: Pearson Education do Brasil, 1995.

PRESSMAN, Roger S. *Engenharia de software.* 6. ed. São Paulo: Pearson Addison Wesley, 2006.

SOMMERVILLE, Ian. *Engenharia de software.* 6. ed. São Paulo: Pearson Addison Wesley, 2003.

VALENTE, Pedro. *Product Owner na prática.* 2010. Disponível em: <http://www.slideshare.net/pedrovalente/product-owner-na-prtica> Acesso em: 23 de Maio de 2013.