UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0002/2016

Análise de Práticas Ágeis no Apoio a Fatores Críticos de Sucesso em Projetos de Software

Denize Pimenta
Gleison Santos

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Relatórios Técnicos do DIA/UNIRIO, No. 0002/2016 Junho, 2016

Análise de Práticas Ágeis no Apoio a Fatores Críticos de Sucesso em Projetos de Software

Denize Pimenta
Gleison Santos

Programa de Pós Graduação em Informática (PPGI)
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Av. Pasteur 458, Urca – CEP 22290-240 – Rio de Janeiro – RJ – Brasil

{denize.pimenta, gleison.santos} @uniriotec.br

Abstract.

There are several agile practices and many of them are related to very specific problems associated to software projects. It is not always easy to project managers to choose the best practices to adopt in a software project. This paper presents the relationship between the challenges in developing software versus the use agile practices. Our goal is to support the choice of agile practices based on critical success factors of software projects and evidences of problems present in experience reports.

Keywords: Agile Methods, Agile Practices, Critical Success Factors

Resumo.

Há várias práticas ágeis e muitas são relacionadas a problemas específicos associados a projetos de software. Em geral, não é trivial para gerentes de projetos escolher as práticas ágeis mais indicadas para serem adotadas em um projeto de software. Este trabalho apresenta a relação entre os principais desafios no desenvolvimento de software e as práticas ágeis. O objetivo é auxiliar a escolha das práticas não ao acaso, mas baseado em fatores críticos de sucesso de projetos de software e de evidências de problemas dados por relatos de experiência.

Palavras-chave: Métodos Ágeis, Práticas Ágeis, Fatores Críticos de Sucesso

Sumário

1 Introdução

O software é muito importante em todas as áreas e, conforme o aumento desta importância, surge a necessidade do desenvolvimento de tecnologias que tornem mais fácil, mais rápido e mais barato desenvolver e manter programas de alta qualidade. Muitas práticas surgem a cada dia e o grande desafio é saber qual a melhor prática a adotar quando se tem um problema.

Este trabalho apresenta uma revisão bibliográfica que relaciona os principais fatores críticos de sucesso no desenvolvimento de sistemas às práticas ágeis mais relevantes. Para que a escolha de uma prática não seja feita por acaso, são apresentadas sua relação com fatores críticos de sucesso e trechos de relatos de bons resultados apresentados na literatura. Espera-se que esta relação facilite o entendimento do problema sendo tratado e, assim, facilite a seleção, adaptação e uso das práticas ágeis associadas.

Este trabalho está dividido em cinco seções além dessa introdução. A Seção 2 apresenta fatores críticos de sucesso no desenvolvimento de software; a Seção 3 contém uma revisão da literatura sobre as práticas ágeis mais relevantes; a Seção 4 faz o relacionamento entre os problemas e desafios apresentados e práticas; a Seção 5 expõe a importância e preocupações durante a implantação das práticas nos processos de software; por fim, a Seção 6 apresenta a conclusão do trabalho e sugestão de trabalhos futuros.

2 Fatores Críticos de Sucesso no Desenvolvimento de Software

Os métodos ágeis possuem um conjunto de boas práticas, como iterações curtas, entregas frequentes, *design* simples, programação por pares, participação constante do cliente, dentre outras [Griffiths 2012]. Antes de examinar as práticas ágeis, é preciso entender quais problemas a serem enfrentados para que se possa analisar a melhor a ferramenta para combatê-los ou evitá-los.

Chow e Cao (2007) apresentam um estudo que relaciona fatores críticos de sucesso em projetos baseados em métodos ágeis. A pesquisa feita em 109 projetos com 408 pessoas entrevistadas, relaciona um conjunto de 6 fatores críticos, baseado no resultado da análise de regressão e teste de hipóteses. Para os autores, os fatores críticos de sucesso são: (a) estratégia de entrega correta, (b) prática adequada de técnicas de engenharia de software ágeis, e (c) equipe de alto calibre. Três outros fatores que podem ser cruciais para certas dimensões de sucesso segundo os autores são [Chow e Cao 2007]: (a) bom processo de gerenciamento de projetos ágeis, (b) ambiente de equipe amigável e ágil, e (c) forte envolvimento do cliente".

Como as práticas ágeis podem ser utilizadas inclusive por projetos que utilizam metodologia de desenvolvimento tradicional, consideramos uma fonte com estudo mais abrangente realizado por Nasir e Sahibuddin (2011). Os autores apresentam uma revisão bibliográfica com 43 artigos (publicados de 1990 a 2010), que adotaram o método de análise de conteúdo e análise de frequência. Como resultado foram identificados 26 fatores críticos de sucesso para projetos de software. Estes fatores são listados na Tabela 1, ordenada pela frequência de citação da bibliografia encontrada pelos autores. A Tabela 2 acrescenta a coluna observações aos fatores críticos para conceituar e com-

plementar o fator crítico, possibilitando a identificação da prática ágil que melhor se enquadra.

Tabela 1. Fatores Críticos de Sucesso para Projetos de Software [Nasir and Sahibuddin 2011]

Fator Crítico de Sucesso	Frequência de citação
Requisitos e especificações claras	26
Objetivos e metas claros	24
Prazos realistas	23
Habilidades eficazes de gestão de projeto / metodologias (Gerente do projeto)	23
Apoio da alta administração	22
Usuário / envolvimento do cliente	20
Comunicação eficaz e *feedback*	20
Orçamento realista	19
Pessoal qualificado e suficiente	18
Requisito congelado	17
A familiaridade com a tecnologia / metodologia de desenvolvimento	15
Planejamento adequado	15
Processos adequados de desenvolvimento / metodologias (processo)	14
Relatório de progresso atualizado	12
Monitoramento e controle efetivo	12
Recursos adequados	11
Boa liderança	11
Gerenciamento de riscos	10
Complexidade, tamanho do projeto, duração, número de organizações envolvidas	10
Mudança efetiva e gerenciamento de configuração	10
Ferramentas de apoio e boa infraestrutura	9
Equipe comprometida e motivada	9
Bom gerenciamento da qualidade	9
Designação clara de papéis e responsabilidades	7
Bom desempenho de fornecedores / empreiteiros / consultores	4
Fornecimento de treinamento ao usuário final	2

Tabela 2. Fatores Críticos de Sucesso

Fator Crítico de Sucesso	Observações
Requisitos e especificações claras	Este fator crítico é uma ameaça que à incompreensão dos requisitos, ou seja, não ter requisitos completamente definidos antes de iniciar o desenvolvimento, ou não entender o esforço verdadeiro do trabalho [Schmidt et al. 2001].
Objetivos e metas claros	A alteração de escopo / objetivos do projeto ocorre quando há mudanças significativas ou reorganização do negócio no meio do projeto [Schmidt et al. 2001].
Prazos realistas	Trata a possibilidade da ocorrência prazos irrealistas ou artificiais [Schmidt et al. 2001].
Habilidades eficazes de gestão de projeto / metodologias (Gerente do projeto)	Este fator crítico é ameaçado devido à ausência de eficácia de gerenciamento de projetos, causado quando gerente do projeto não tem poder ou habilidades [Schmidt et al. 2001].
Apoio da alta administração	A falta de comprometimento da alta direção com o projeto é uma grande ameaça ao projeto, pois não havendo supervisão dos executivos e visibilidade de seu compromisso, os recursos necessários ficam comprometidos [Schmidt et al. 2001].

Fator Crítico de Sucesso	Observações
Usuário / envolvimento do cliente	Usuários devem participar ativamente da equipe do projeto, e se comprometer com as suas entregas, de acordo com suas responsabilidades. O usuário deve dedicar seu tempo aos objetivos do projeto [Schmidt et al. 2001].
Comunicação eficaz e *feedback*	A ameaça identificada na ausência de comunicação é a incapacidade de gerir as expectativas do usuário final, ou seja, expectativas descasadas com entrega - muito altas ou muito baixas - causam problemas [Schmidt et al. 2001].
Orçamento realista	A ameaça a este fator crítico é a falta de instrumento eficaz ou técnicas estruturadas para estimar corretamente escopo do trabalho [Schmidt et al. 2001].
Pessoal qualificado e suficiente	Schmidt et al. (2001) afirmam que funcionários insuficientes ou inadequados, com habilidades erradas, competências insuficientes de acordo com o projeto, formam uma grande ameaça [Schmidt et al. 2001].
Requisito congelado	Uma grande ameaça é a mudança constante nas necessidades dos usuários, consequentemente, o sistema nunca será implantado em produção porque nenhum dos requisitos será concluído. Alternativamente, o congelamento de um subconjunto da funcionalidade permite a conclusão do sistema e versões de atualização, conforme necessário [Schmidt et al. 2001].
A familiaridade com a tecnologia / metodologia de desenvolvimento	A ameaça a este fator crítico é utilizar tecnologia nova, quando não há domínio sobre a mesma, ou quando grande mudança tecnológica ocorre durante o projeto [Schmidt et al. 2001].
Planejamento adequado	A ameaça ocorre quando não há planejamento ou o planejamento elaborado é inadequado [Schmidt et al. 2001].
Processos adequados de desenvolvimento / metodologias (processo)	A escolha da estratégia de desenvolvimento pode ser errada para o tipo do projeto, por exemplo, cascata, prototipagem, etc. [Schmidt et al. 2001].
Relatório de progresso atualizado	Uma das maiores ameaças para o desenvolvimento de sistemas é ter relatório de status defasado ou com dados falsos [Jones 2006].
Monitoramento e controle efetivo	A ameaça a este fator crítico ocorre quando há controle pobre ou inexistente do projeto, ou seja, nenhuma sinalização, nenhuma metodologia de acompanhamento de projetos, ou deixar os envolvidos sem conhecimento do estado geral do projeto [Schmidt et al. 2001].
Recursos adequados	Ausência de recursos para desempenhar as funções, por exemplo, tecnologia, conhecimento do negócio, e experiência [Schmidt et al. 2001].
Boa liderança	A ausência de habilidades em liderança, ou seja, o gerente do projeto tenta "controlar" horários, tecnologia, requisitos, etc. [Schmidt et al. 2001].
Gerenciamento de riscos	O principal impedimento a este fator crítico é a má gestão do risco, ou seja, ignorar, ou combater os riscos errados [Schmidt et al. 2001].
Complexidade, tamanho do projeto, duração, número de organizações envolvidas	O fator crítico é afetado pelo aumento do número de linhas de comunicação e potencial chance de ocorrer conflito [Schmidt et al. 2001].
Mudança efetiva e gerenciamento de configuração	Fator de risco no projeto é a equipe não empregar adequadamente controle de mudanças, ou outros processos necessários [Schmidt et al. 2001].

Fator Crítico de Sucesso	Observações
Equipe comprometida e motivada	Beynon-Davies argumenta que o compromisso por parte dos participantes do projeto é um fator determinante necessário para o sucesso de um projeto [Beynon-Davies 1999].
Bom gerenciamento da qualidade	De acordo com Jones (2006), o controle efetivo da qualidade do software é o fator mais importante que separa projetos bem-sucedidos de atrasos e desastres [Jones 2006].
Designação clara de papéis e responsabilidades	O maior impedimento para este fator crítico é pela definição inadequada de funções e responsabilidades, ocorre quando a equipe do projeto e/ou da organização não possuem funções e responsabilidades claramente estabelecidas [Schmidt et al. 2001].
Bom desempenho de fornecedores / empreiteiros / consultores	A ameaça está nas dependências externas não atendidas, ou seja, consultores ou vendedores do projeto não entregam ou saem do negócio [Schmidt et al. 2001].
Fornecimento de treinamento ao usuário final	Usuários identificarem erros por não saberem utilizar o sistema [Beynon-Davies 1999].

3 Métodos Ágeis

Os métodos de desenvolvimento tradicionais dão mais ênfase à documentação do que à geração do produto final, pois refletem o ato de fazer certo da primeira vez. Por outro lado, os métodos ágeis são mais flexíveis e adaptativos, com isso, possuem a habilidade de sobreviver em um ambiente em constante mudança, tão comum nas empresas atuais. Alguns métodos ágeis se tornaram comuns, como *Scrum*, *Extreme Programming*, *Feature Driven Development, Dynamic Systems Development Method, Adaptive Software Development, Crystal*, e *Test Driven Development* [Carvalho e Mello 2012].

Com um conjunto de novos métodos, surgiu também um grande conjunto de práticas ágeis. A revisão bibliográfica apresentada por Abrantes e Travassos (2011), aponta que as práticas ágeis mais comumente abordadas são: Desenvolvimento orientado por teste, integração contínua, programação em par, jogos de planejamento, cliente presente, propriedade coletiva do código, pequenas liberações, metáfora, refatoração, ritmo sustentável, design simples, padrões de código, equipe completa, visibilidade do projeto, reuniões diárias, ambiente aberto e *backlog* do produto.

Abrantes (2012), a partir de uma revisão sistemática da literatura, identificou 17 práticas ágeis. Para priorizar esta lista por relevância, o autor aplicou um *survey* a 25 pesquisadores e profissionais de engenharia de software [Abrantes 2012]. Esta pesquisa foi reexecutada por Mello et al. (2014), envolvendo 292 participantes, aplicando técnicas baseadas em teoria dos grafos e análise multivariada para estratificação de grupos. O *survey* realizado em 2014 refinou o conjunto preliminar elaborado por Abrantes, que dentre outros pontos refuta a hipótese de o conjunto de práticas estabelecido inicialmente estar completo, pois novas práticas são descobertas, o que denota a natureza dinâmica da aplicação das práticas [Mello et al. 2014].

Nesse *survey* atualizado, além da relevância, foi analisada a pertinência das práticas. Os autores apontam que a prática "metáfora" foi descartada, enquanto que outras práticas pertinentes surgiram, sendo elas: Desenvolvimento orientado a comportamento, Programação em Pares e Revisão de Software. Estas práticas, oriundas dos métodos ágeis, são detalhadas nos itens a seguir por ordem de relevância.

1. Integração Contínua (*continuous integration*) - Quando vários desenvolvedores trabalham junto no mesmo código deve haver sincronização dos trabalhos para que o

trabalho de um não interfira no trabalho de outro, ao mesmo tempo em que todos precisam evoluir rapidamente. A integração contínua visa ao trabalho de forma isolada, porém juntando o que produzem com a versão mais recente do código de produção, diversas vezes ao dia. Isto é, os pares se sincronizam com frequência à medida que terminam pequenas atividades de codificação [Teles 2005]. Traz problemas à tona antes de mais código ser construído [Griffiths 2012].

2. *Backlog* do Produto (*product backlog*) - Conjunto completo dos requisitos conhecidos do produto, com os itens priorizados pelo cliente [Cunha e Andrade 2014]. É uma lista ordenada de tudo que pode ser necessário para o produto, como: funcionalidades, funções, requisitos, atributos de qualidade, melhorias e correções a serem construídas. É a única fonte de requisitos, é dinâmica e evolui como o produto [Griffiths 2012].

3. Visibilidade do Projeto (*project visibility*) - Pode ser feita por um ou mais painéis (na web ou não) para manter a qualquer tempo o status e as medições do progresso do projeto. O avanço do projeto em relação às histórias de usuários, as quais as equipes se comprometeram a entregar no final das iterações, deve ser incluído. Modelos devem se tornar acessíveis para todas as equipes [Abrantes e Travassos 2011]. Várias ferramentas apóiam esta prática, como: gráfico *burndown*, quadro de tarefas *kanban*, dentre outros.

4. Pequenas Liberações (*small releases*) - Entrega em ciclos curtos, chamados de *sprint* ou iteração, aumenta a participação dos envolvidos e possibilita *feedback* mais cedo [Schwaber e Sutherland 2013]. Estas entregas são frequentes e devem ser pequenas, tanto na iteração para demonstrar o progresso e aumentar a visibilidade para o cliente, quanto na *release* para implementar rapidamente uma versão do software para o público-alvo [Griffiths 2012].

5. Equipe Completa (*whole team*) - Inclui todos os perfis necessários, como clientes, usuários e demais interessados, para que a equipe possa ter bom desempenho, enfatizando o espírito de equipe com todos os seus membros compartilhando um propósito e apoiando-se mutuamente [Abrantes 2012].

6. Propriedade Coletiva de Código (*collective code ownership*) - Qualquer membro da equipe pode melhorar ou corrigir qualquer código. Isto significa que várias pessoas trabalham no código todo, o que resulta em melhoria de visibilidade e conhecimento da base do código [Griffiths 2012].

7. Ritmo Sustentável (*sustainable pace*) - Trabalhar por longos períodos com carga extra acaba tornando o desenvolvimento insustentável e contraprodutivo. Esta prática mantém um ritmo sustentável de desenvolvimento, otimizando entregas de longo prazo [Griffiths 2012].

8. Reuniões diárias (*stand-up meetings*) - Reunião onde cada participante informa o que fez, o que fará e quais são os impedimentos que estão atrapalhando a execução das atividades [Cunha e Andrade 2014]. O objetivo é avaliar o progresso do trabalho [Griffiths 2012].

9. Refatoração (*refactoring*) - É o processo de melhorar o design do código existente sem alterar o comportamento externo ou incluir novas funcionalidades. Mantendo o design eficiente, mudanças e novas funcionalidades podem facilmente ser aplicadas no código. O foco é remover código duplicado, diminuir o acoplamento e aumentar a coesão [Griffiths 2012].

10. Design Simples (*simple design*) - A técnica tem foco em manter o design simples, mas adequado, permitindo o desenvolvimento rápido e adaptável às necessidades. O design é mantido apropriado para o que o projeto requer no momento. Deve ser revisado iterativamente e incrementalmente para garantir que continua apropriado [Griffiths 2012].

11. Desenvolvimento Orientado a Testes (*test driven development - TDD*) - A equipe escreve o teste antes de desenvolver o código, desta forma os problemas são destacados no início do desenvolvimento, permitindo *feedbacks* mais cedo [Griffiths 2012].

12. Padronização de Código (*coding standards*) - Como todos da equipe podem alterar o código, é imprescindível seguir um padrão de codificação consistente para que todo o código pareça ser escrito por um único programador experiente. As especificidades do padrão que cada equipe usa não são importantes; o que importa é que a equipe tenha uma abordagem consistente para escrever o código [Griffiths 2012].

13. Jogos de Planejamento (*planning game*) - São as atividades envolvidas para o planejamento do que conterá uma versão do software (release) ou uma iteração, onde desenvolvedores e clientes atuam juntos. O cliente prioriza os requisitos mais importantes a serem incluídos em uma entrega curta e incremental. Os desenvolvedores estimam o custo das funcionalidades candidatas, conforme a priorização estabelecida pelo custo e valor agregado para o negócio. Os desenvolvedores então dividem as histórias em tarefas, mas sem envolver o cliente com detalhes de implementação. A meta do jogo de planejamento é balancear os interesses do cliente com a capacidade da equipe, de forma contínua e progressiva [Abrantes 2012].

14. Cliente Presente (*on-site customer*) - Significa colocar cliente e desenvolvedores para trabalhar junto, com objetivo de esclarecer e validar requisitos, estabelecer prioridades, responder perguntas, fazer testes de aceitação e assegurar que o desenvolvimento tenha o progresso esperado. Esta prática também leva o cliente a mudar mais prontamente os requisitos, ajudando a equipe a mudar o foco dos esforços de desenvolvimento para as necessidades primordiais [Abrantes 2012].

15. Desenvolvimento orientado a comportamento (*behavior driven development - BDD*) – Esta técnica tem por objetivo a integração das regras de negócio com a codificação do sistema, dando ênfase no comportamento e ainda escrevendo os testes antes do código funcional [Matté 2011].

16. Programação em Pares (*pair programming*) - O código é produzido simultaneamente por dois desenvolvedores, para escrever e prover revisão de software ao mesmo tempo [Griffiths 2012].

17. Revisão de Software (*software review*) - Realização de atividades que incluem a revisão de código e de modelos de design, o que antecipará a correção de defeitos antes da apresentação/entrega de um módulo. Estas revisões devem ser frequentes e podem ser realizadas em grupo, em pares ou individualmente [Mello et al. 2014].

4 Desafios e Práticas Ágeis

A Tabela 3 unifica a lista de problemas apresentados na Seção 2 com as práticas listadas na Seção 3, com intuito de permitir o entendimento de qual técnica auxilia em qual Fator Crítico.

O fator crítico "requisito congelado" não é apresentado na Tabela 2 por ser contrário ao princípio ágil de "Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento", conforme citado no manifesto ágil [Griffiths 2012].

São apresentadas na última coluna Tabela 3 evidências de relatos de experiências que melhor embasar a discussão sobre a utilidade, necessidade e implantação de cada prática considerada.

Tabela 3. Relacionamento entre Fatores Críticos de Sucesso e Práticas Ágeis

Fator Crítico de Sucesso	Prática Ágil	Evidências / Relatos de Experiência
Requisitos e especificações claras	***Backlog* do Produto**	"O *backlog* foi difícil de gerir, o que causou uma situação em que os requisitos não eram compreensíveis para o Desenvolvedores e o Engenheiro da Qualidade (cliente externo) " [Pikkarainen et al. 2008].
	Equipe Completa	"O documento de requisitos de alto nível é aberto à interpretação, mas podíamos sempre encontrar com o engenheiro de sistemas, quando precisávamos de ajuda". [Rising and Janoff 2000].
	Cliente Presente	Cao e Ramesh (2008) informam que com o acesso imediato aos clientes e o envolvimento deles no projeto, os requisitos ficam mais claros e mais compreensíveis. Por outro lado, alertam para um risco que prática pode ocasionar quando a interação entre clientes e desenvolvedores não for de alta qualidade [Cao and Ramesh, 2008].
	Pequenas Liberações	Teles (2005) pôde observar a "importância da utilização de iterações curtas e *feedback* rápido, especialmente onde os requisitos eram inovadores e o requerente não possuía profunda experiência sobre o domínio do problema". Por outro lado, é necessário ter atenção quando mais de um grupo de cliente está envolvido e cada grupo com considerações diferentes do sistema, pois encontrar consenso ou compromisso em ciclo de desenvolvimento curto é um desafio. [Cao and Ramesh 2008].
Objetivos e metas claros	***Backlog* do Produto**	Na experiência relatada por Carvalho e Mello (2012), o *backlog* do produto foi totalmente implementado e controlado pelo dono do produto e estava sempre atualizado e facilmente acessível para todos os membros do time. Entretanto, na experiência de Leal e Santos (2015), foi relatada dificuldade em encontrar um responsável, dificultando na resolução do escopo e *feedback* do produto.
Prazos realistas	**Jogos de Planejamento**	Ao fazer a revisão da primeira iteração, a equipe apontou vários erros, o maior deles foi o planejamento mal feito com a criação de um *backlog* muito grande para a iteração [Carvalho e Mello 2012].
	Equipe Completa	"As equipes completas podem apresentar mais capacidade para buscar a fatia de tempo mais adequada para dividir um processo do projeto, e fazer a divisão do trabalho em múltiplas entregas" [Abrantes 2012].

Fator Crítico de Sucesso	Prática Ágil	Evidências / Relatos de Experiência
Usuário / envolvimento do cliente	**Pequenas Liberações**	"Os projetos pilotos relataram ter enfrentado dificuldades em convencer os clientes a realizar publicações parciais do sistema" (implantação gradual) [Melo e Ferreira 2010].
	Cliente Presente	Teles (2005) relata que para que a troca de *feedback* possa ocorrer e o cliente possa obter o máximo de valor do projeto, é essencial que ele participe ativamente do processo de desenvolvimento. Além disso, a sua presença viabiliza a simplicidade em diversos aspectos, especialmente na comunicação [Teles 2005].
Comunicação eficaz e *feedback*	***Backlog* do Produto**	A falta de documentação escrita é percebida no relato de um testador: "Precisaríamos de uma melhor documentação para a equipe de engenharia de qualidade trabalhar" [Pikkarainen et al. 2008].
	Reuniões Diárias	Um relato de melhoria na comunicação entre os membros da equipe com a adoção das *stand-up meetings* pode ser consultado em [Ramos et al. 2013].
	Visibilidade do Projeto	Costinhas e Santos (2014) relatam o benefício de facilitação da comunicação para todos os envolvidos com o quadro *kanban*, mas também apresentam dificuldade de resistência inicial com a implantação da ferramenta.
	Cliente Presente	Teles (2005) ressalta a importância da comunicação e *feedback* com a presença e participação do cliente, aumentando o aprendizado da equipe e possibilitando a correção de erros mais cedo.
	Propriedade Coletiva do Código	Abrantes (2012) informa que a propriedade coletiva de código, se efetivamente exercitada pela equipe, pode apoiar a comunicação e eventualmente melhorar produtividade e qualidade.
Familiaridade com a tecnologia / metodologia de desenvolvimento	**Pequenas Liberações**	Carvalho e Mello (2012) citam a inexperiência da equipe como causa de ter tido alguns *Sprints* de duração muito longa (inexperiência dos envolvidos).
Planejamento adequado	***Backlog* do Produto**	Alguns relatos de experiência apresentam dificuldade do uso do *product backlog*, como não ter sido devidamente organizados e, consequentemente, não ter sido priorizados [Pikkarainen et al. 2008]. Carvalho e Mello (2012) informam que a dificuldade encontrada foi em elaborar a primeira versão do *backlog* que geralmente é muito extensa, mas também colocam que a adoção desta prática melhorou o planejamento do time.
	Jogos de Planejamento	O estudo realizado por Pikkarainen et al. (2008) informam que esta prática foi útil devido a situação do projeto ter sido bem gerida e toda a equipe do projeto estar ciente dos planos de projeto e metas das iterações. Porém, há um relato neste mesmo estudo que afirma as reuniões de planejamento de *sprint* eram muito curtas, considerando a enorme quantidade de características e requisitos.
Relatório de progresso atualizado	**Visibilidade do projeto**	Alguns relatos defendem o benefício desta prática na visualização do progresso do projeto, são elas: (i) (*Story/task board*) permitiu a todos visualizar o status do projeto em uma página [Pikkarainen et al. 2008]; (ii) As retrospectivas e seus cartazes ajudam o time a entender o andamento do projeto

Fator Crítico de Sucesso	Prática Ágil	Evidências / Relatos de Experiência
		[Sato 2007]; (iii) A respeito do gráfico *burndown* Carvalho e Mello (2012) afirmam que é uma ferramenta visual interessante, que deixa claro o que os números já mostravam, porém também afirmam que tende a ser abandonado, pois, à primeira vista, parece ser informação redundante.
	Reuniões diárias	Pikkarainen et al. (2008) argumentam que esta prática é uma boa maneira de manter desenvolvedores, líder do projeto e cliente (em alguns casos) cientes do status do projeto, mas alertam que nem todas as reuniões são interessantes ao cliente, pois em alguns casos são muito demoradas.
Monitoramento e controle efetivo	**Reuniões diárias**	Carvalho e Mello (2012) apresentaram dificuldade ao implantar esta prática, pois houve desencontros de horários dos membros da equipe, mas também puderam observar o ganho afirmando que a utilização da prática aumentou muito o controle do projeto e os riscos foram minimizados.
Boa liderança	**Equipe Completa**	A equipe trabalhou bem, colaborando entre si, e tinham grande capacidade de desempenhar várias funções simultâneas, mas como não trabalhavam lado a lado (turnos diferentes), tiveram dificuldade em se autogerenciar [Carvalho e Mello 2012].
	Visibilidade do projeto	"O status atualizado do projeto, com seus artefatos, disponíveis para as equipes pode contribuir para que as melhores maneiras de se trabalhar sejam identificadas para completar os itens de trabalho (apóia a auto-organização)." [Abrantes 2012].
Gerenciamento de riscos	***Backlog* do Produto**	Foi criado o backlog de impedimento serve para todo o time conhecer os riscos e cobrar mitigação de riscos conhecidos [Carvalho e Mello 2012].
Complexidade, tamanho do projeto, duração, número de organizações envolvidas	**Programação em Pares**	"Foi relatada melhoria na resolução de casos complexos com o uso da programação em pares" [Ramos et al. 2013]. Os resultados dos estudos mostraram que, pela cooperação, os programadores podem concluir tarefas e atingir metas que seriam difíceis ou impossíveis se serem feitas individualmente [Dyba et al. 2007].
	Cliente Presente	Teles (2005) relata o ganho de produtividade com os envolvidos próximos e a comunicação presencial por meio do diálogo, principalmente devido à complexidade dos detalhes do projeto.
Mudança efetiva e gerenciamento de configuração	**Refatoração**	Refatoração altera a estrutura interna de software para torná-lo mais fácil de entender e mais barato para modificar sem alterar seu comportamento observável. Entretanto, a refatoração depende de muitos fatores, como experiência do desenvolvedor e pressão de prazo. Ocasionalmente, a única alternativa é jogar o código fora e construir novamente, devido à arquitetura se tornar inadequada ou inapropriada [Cao e Ramesh 2008].
	Pequenas Liberações	"As liberações frequentes, com funcionalidades de maior valor sendo priorizadas podem apoiar a capacidade do processo ser adaptado para atender mudanças de última hora nos requisitos e/ou no ambiente de desenvolvimento" [Abrantes 2012].
	Cliente Presente	No final de cada ciclo de desenvolvimento, há a avaliação dos testes e das características implementadas. Clientes fornecem

Fator Crítico de Sucesso	Prática Ágil	Evidências / Relatos de Experiência
		feedback e podem solicitar mudanças importantes se suas expectativas não forem alcançadas [Cao e Ramesh 2008].
Equipe comprometida e motivada	**Pequenas Liberações**	Carvalho e Mello (2012) relatam que a divisão do projeto em *Sprints* aumentou a motivação do time.
	Ritmo Sustentável	No *survey* apresentado por Laanti (2013) indica que 82% dos entrevistados afirmam trabalhar com ritmo sustentável e que 64% consideraram que seu desempenho tinha aumentado.
Bom gerenciamento da qualidade	**Integração Contínua**	Foi relatado no estudo de Pikkarainen et al. (2008) que ter sempre uma nova versão disponível para testes ajuda os Engenheiros de Qualidade a obter informações sobre o status do produto final.
	Desenvolvimento Orientado por Testes	O estudo realizado por Feitosa (2007) indica a melhoria da qualidade utilizando TDD, pois a quantidade de defeitos foi 2,6 vezes menor e em outro departamento foi 4,2 vezes menor. O relato realizado por Sato (2007) informa que os programadores mais experientes tendem a demonstrar maior resistência a práticas ágeis como TDD, Código Compartilhado e Programação em Pares, pois eles precisam mudar drasticamente seu modo de trabalho, alterando algo que já estão acostumados há diversos anos.
	Programação em Pares	Pikkarainen et al. (2008) relatam que esta prática é uma forma eficiente de implementar a revisão do código, mas é difícil e problemática para uso diário. Já Dyba et al. (2007) informam o ganho com a alta qualidade de código na metade do tempo.
	Refatoração	Refatorações podem ser classificadas de acordo com robustez, extensibilidade, reutilização e desempenho, permitindo melhorar a qualidade de software, aplicando as refatorações relevantes nos lugares certos [Mens and Tourwe 2004].
Bom desempenho de fornecedores / empreiteiros / consultores	**Design Simples**	"A equipe técnica destacou ainda que o projeto simples, os testes automatizados e o código padronizado foram práticas que favoreceram a subcontratação da evolução do software de forma mais eficiente que a tradicional documentação do sistema" [Melo e Ferreira 2010].
	Padronização de Código	

5 Discussão

De acordo com Shull et al. (2001) "Um processo bem definido pode ser observado e medido, e, assim, melhorado". Os autores indicam nesse artigo uma abordagem metodológica para evolução de processos de software de forma gradual, com experimentos iniciais, medições, análise antes da implantação em massa e indicam também a análise de melhorias constantes.

O relato das experiências apresentado anteriormente na Tabela 3 mostra que a mesma prática implantada pode apresentar resultado, dificuldades e percepções diferentes, para tal, alguns fatores podem influenciar no resultado do uso da prática, como: (1) conhecimento prévio das práticas e experiência dos envolvidos, (2) ferramenta que apoia a prática, (3) implantação gradual e (4) melhoria contínua.

Fatores críticos de sucesso não abordados por práticas

As práticas ágeis focam nas ações para desenvolvimento, controle e qualidade do produto final, sendo assim, os fatores críticos de sucesso citados que estão além do processo de desenvolvimento do produto, como, por exemplo: "Apoio da alta administração", "Pessoal qualificado e suficiente", "Recursos adequados", "Processos adequados de desenvolvimento / metodologias (processo)" ou "Fornecimento de treinamento ao usuário final" não encontram práticas ágeis para apoiá-los.

As atividades relacionadas ao gerenciamento do projeto que estabelecem o desenvolvimento orientado a planejamento como nos casos dos fatores críticos "Orçamento realista" e "Habilidades eficazes de gestão de projeto / metodologias (gerente do projeto)" também não tiveram relacionamento com nenhuma prática ágil.

Para o fator crítico "Designação clara de papéis e responsabilidades", apesar de não ter sido identificada nenhuma prática para minimizar os problemas que possam afetá-lo, há relato de dificuldades encontradas com o uso de métodos ágeis que influenciam este fator.

Práticas ágeis não abordadas ou pouco abordadas

"Propriedade coletiva do código" é uma prática ágil que auxilia apenas o fator crítico "Comunicação eficaz e *feedback*". A "Integração contínua" possui forte conotação de qualidade e foi relacionada somente ao fator crítico "Bom gerenciamento da qualidade". Não foi encontrado nenhum relato de experiência que apresentasse pontos positivos ou negativos no uso das práticas "BDD" e "revisão de software", apesar de possuírem uma forte conotação de qualidade. As demais práticas possuem relacionamentos com dois ou mais fatores críticos de sucesso de projetos de software.

6 Conclusão

Esse trabalho apresentou uma análise das práticas ágeis no apoio a fatores críticos de sucesso em projetos de software. O processo de software deve ser adaptado ao ambiente (projeto, empresa, cultura), de acordo com suas características e necessidades, para tal é importante conhecer o problema a ser enfrentado e as ferramentas antes de aplicá-las. As necessidades devem ser entendidas previamente, para que seja constatado o sucesso ou validade das práticas implantadas. Uma prática não pode ser usada indiscriminadamente, os relatos de experiência citados neste trabalho mostram pontos de sucesso e de insucesso e as dificuldades ao utilizar uma prática.

As limitações dos resultados obtidos neste trabalho são influenciadas diretamente: (i) pela qualidade da definição e variedade dos fatores críticos de sucesso em projetos de software; (ii) pelo filtro estabelecido de listar apenas as práticas mais relevantes; (iii) e a abrangência dos artigos de relatos de experiências identificados.

Como próximo passo é sugerido o estudo sistematizado da bibliografia, para a consulta mais ampla, é sugerido também o estudo do relacionamento entre as práticas ágeis ou o aprofundamento de soluções para problemas pouco explorados, como Envolvimento dos Usuários, e novas técnicas, como, por exemplo, a aplicação de jogos sérios no desenvolvimento de sistemas.

Referências Bibliográficas

Abrantes, J. F. (2012) "**Estudos Experimentais sobre Agilidade no Desenvolvimento de Software e sua Utilização no Processo de Teste**". Tese de Doutorado COPPE/UFRJ.

Abrantes, José Fortuna, e Travassos, Guilherme Horta (2011) "Common Agile Practices in Software Processes". **International Symposium on Empirical Software Engineering and Measurement**.

Beynon-Davies, Paul. (1999) "**Human error and information systems failure: the case of the London ambulance service computer-aided despatch system Project**". Computer Studies Department, University of Glamorgan, UK.

Cao, Lan and Ramesh, Balasubramaniam. (2008). "Agile Requirements Engineering Practices: An Empirical Study". **IEEE Software**.

Carvalho, Bernardo V. e Mello, Carlos H. P. (2012). "**Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pena empresa de base tecnológica**". Universidade Federal de Itajubá – UNIFEI.

Chow, Tsun and Cao, Dac-Buu. (2007) "A survey study of critical success factors in agile software projects". **The Journal of Systems and Software**.

Costinhas, C. E. A e Santos, G. (2014) "Adoção de Práticas Ágeis em um Ambiente Tradicional: Um Estudo de Caso", **Simpósio Brasileiro de Qualidade Software - SBQS 2014**, Blumenau - SC.

Cunha, T. F. V. e Andrade, R. M. C. (2014) "Agile DMAIC: Um Método para Avaliar e Melhorar o Uso do Scrum em Projetos de Software". **Simpósio Brasileiro de Qualidade Software - SBQS 2014**, Blumenau – SC.

Dyba, Tore, Arisholm, Erik, Sjoberg, D I K, Hannay, Jo E. and Shull, Forrest (2007). "Are Two Heads Better than One? On the Effectiveness of Pair Programming". **IEEE Software**.

Feitosa, D S (2007) "**Um estudo sobre o impacto do uso de Desenvolvimento Orientado por Testes na melhoria da qualidade de software**". Universidade Federal da Bahia.

Jones, Capers. (2006). "Why Projects Fail – Social and Technical Reasons for Software Project Failures". **CrossTalk – The Journal of Defense Software Engineering**.

Griffiths, M. (2012). PMI-ACP Exam Prep. RMC Publications, Inc.

Laanti, Maarit (2013). "Agile and Wellbeing — Stress, Empowerment, and Performance in Scrum and Kanban Teams". **46th Hawaii International Conference on System Sciences**.

Leal, Tainá e Santos, Gleison (2015) "Um Survey sobre Métodos Ágeis e o Pós-Agilismo". **Congresso Ibero-Americano de Engenharia de Software - CIBSE 2015**, Lima – Peru.

Matté, Marcio Angelo (2011) "**Testes de Software: Uma Abordagem da Atividade de Teste Software em Metodologias Ágeis Aplicando a Técnica *Behavior Driven Development* em um Estudo Experimental**". Universidade Tecnológica Federal do Paraná – UTFPR.

Mello, R. M, da Silva, P. C. e Travassos, G. H. (2014). Agilidade em Processos de Software: Evidências Sobre Características de Agilidade e Práticas Ágeis. **Simpósio Brasileiro de Qualidade de Software – SBQS 2014**, Blumenau - SC.

Melo, Claudia de O, Ferreira, Gisele R. M. (2010). "Adoção de métodos ágeis em um Instituição Pública de grande porte – um estudo de caso". **Workshop Brasileiro de Métodos Ágeis** (Agile Brasil 2010). Centro de Eventos da PUCRS (CEPUC), Porto Alegre – Brasil.

Mens, Tom and Tourwe, Tom (2004) "A Survey of Software Refactoring". **IEEE Transactions On Software Engineering**, Vol. 30, No. 2.

Nasir M H N and Sahibuddin S (2011) "**Critical success factors for software projects: A comparative study**" Faculty of Computer Science and Information Technology and Advanced Informatics School.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson P. and J. Still (2008) "The impact of agile practices on communication in software development". **Empirical Software Engineering** Volume 13, Issue 3, 13, pp 303–337.

Rising, Linda and Janoff, Norman S. (2000). "The Scrum Software Development Process for Small Teams". **IEEE Software**.

Ramos, A L B M, Lima, T L L, Cunha, LM R V, Tavares, R L A (2013). "Práticas Ágeis Aplicadas a um Processo de Manutenção de Software: Um Relato de Experiência".

Sato, Danilo Toshiaki (2007) "**Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**". Dissertação de Mestrado, Universidade de São Paulo.

Schmidt, Roy, Lyytinen, Kalle, Keil, Mark And Cule, Paul (2001). "Identifying Software Project Risks - An International Delphi Study". **Journal of Management Information Systems**.

Schwaber, Ken e Sutherland, Jeff (2013). **Scrum Guide**. Disponível em: http://www.scrumguides.org/, último acesso em: 16/06/2016.

Shull, Forrest, Carver, Jeffrey, and Travassos, Guilherme H. (2001). "An Empirical Methodology for Introducing Software Process". http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/vienna.pdf.

Teles, Vinicius Manhães (2005). "**Um Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming**". Dissertação (Mestrado em Informática) – UFRJ.