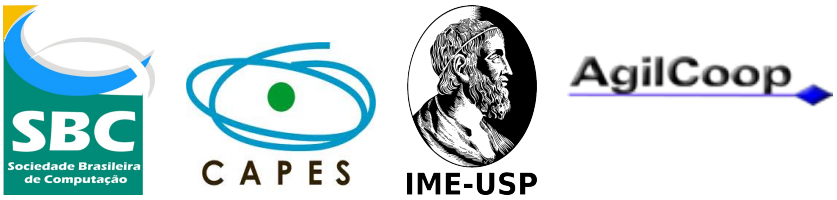
3rd Brazilian Workshop on Agile Methods (WBMA'2012)

Agile Brazil 2012
IME - USP - SP

Apoio:



ISBN: 978-85-88697-24-9
05 de Setembro de 2012

Letter from the Program Committee Chair

Welcome to the 3rd Brazilian Workshop on Agile Software Development (WBMA'2012). WBMA is a scientific venue for the presentation of academic and industrial research works in the field of Agile Methods.

This year we received 17 full paper submissions. Each paper was reviewed by 3 or 4 anonymous referees, who provided detailed guidance on how to improve the papers and helped the PC members to decide on the approval or rejection of the papers. After the first round of reviews, 3 papers were accepted, 9 were rejected and 5 were assigned a shepherd to help further improve the quality of the paper. The authors had a few weeks to interact with their shepherds and a new version of the paper was submitted and reevaluated by the shepherds and the PC chair. In the end, all shepherded papers were considered to have reached the bar for acceptance and, in total, 8 papers were accepted for publication in the proceedings and presentation in the workshop.

We would like to thank the work of our wonderful program committee members who, despite being busy international experts in the field, provided great feedback for our authors in a timely manner.

The major topics covered by the papers you will find in these proceedings are:

- User eXperience
- Organizational design and innovation
- Technical debt
- Pair programming
- Requirements
- Testing
- Dependency management
- Metrics for monitoring

We hope you have a wonderful productive and fun time at the workshop and that the interaction among the participants bring new insights, ideas, and motivation for further scientific works in the field of agile development as well as its application into the real world.

São Paulo, September, 2012
Fabio Kon, PC Chair

Proceedings of the 3rd Brazilian Workshop on Agile Methods (WBMA'2012)

- **Comitê Organizador**
  Rafael Prikladnicki (PUCRS, Brasil)
  Teresa Maciel (UFRPE, Brasil)

- **Coordenador do Comitê de Programa**
  Fabio Kon (IME-USP, Brasil)

- **Coordenadora de Publicidade**
  Claudia Melo (IME-USP, Brasil)

- **Coordenadora de Publicação**
  Viviane Santos (IME-USP, Brasil)

- **Comitê de Programa**
  Alexandre Álvaro (UFSCar-Sorocaba, Brasil)
  Alexandre Vasconcelos (UFPE, Brasil)
  Alfredo Goldman (IME-USP, Brasil)
  Angela Martin (Victoria University of Wellington, New Zealand)
  Célio Santana (UFRPE, Brasil)
  Claudia Melo (IME-USP, Brasil)
  Clovis Torres Fernandes (ITA, Brasil)
  Cristine Martins Gomes de Gusmão (UFPE, Brasil)
  Eduardo Guerra (ITA, Brasil)
  Fabio Kon (IME-USP, Brasil) - Coordenador
  Frank Maurer (University of Calgary, Canada)
  Giancarlo Succi (Free University of Bozen/Bolzano, Italy)
  Joe Yoder (The Refactory Inc., USA)
  José Carlos Maldonado (ICMC-USP, Brasil)
  Juan Garbajosa (Technical University of Madrid, Spain)
  Jutta Eckstein (IT Communication, Germany)
  Kieran Conboy (University of New South Wales, Australia)
  Marco Aurélio Gerosa (IME-USP, Brasil)
  Maria Istela Cagnin (UFMS, Brasil)
  Raul Sidnei Wazlawick (UFSC, Brasil)
  Rafael Prikladnicki (PUCRS, Brasil)
  Rodrigo de Toledo (UFRJ, Brazil)
  Teresa Maciel (UFRPE, Brasil)
  Viviane Santos (IME-USP, Brasil)
  Xiaofeng Wang (Free University of Bozen/Bolzano, Italy)

Índice

User eXperience e desenvolvimento Ágil: lições aprendidas

Tiago Silva da Silva[1], **Milene Selbach Silveira**[2]

[1]Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo - Campus São Carlos
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil

[2]PUCRS – Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681, Prédio 32 – 90619-900 – Porto Alegre – RS – Brazil

tiago.silva@icmc.usp.br, milene.silveira@pucrs.br

Abstract. *Agile development and User eXperience design have distinct cultures that at a first glance seems to conflict. However, these two methodologies present similarities and they can provide great results when combined appropriately. This paper presents lessons learned during the development of an Agile and UX integration proposal, including theory and practice.*

Resumo. *O desenvolvimento Ágil e User eXperience design possuem culturas distintas que, à primeira vista, parecem entrar em conflito. No entanto, estas duas metodologias apresentam semelhanças e podem oferecer ótimos resultados quando combinadas de forma adequada. Este artigo apresenta lições aprendidas no decorrer da construção de uma proposta de integração de UX com o desenvolvimento Ágil, compreendendo teoria e prática.*

1. Introdução

Métodos Ágeis são projetados para produzir software mais alinhado com as necessidades do cliente, realizar entregas de forma mais rápida do que em modelos de desenvolvimento tradicionais, como o modelo em cascata, por exemplo. Já o UX (*User eXperience*) *design* concentra-se na concepção de produtos interativos que apoiem o modo como as pessoas se comunicam e interagem nas suas atividades do cotidiano.

Tradicionalmente, estas duas metodologias utilizam diferentes abordagens para alocar recursos em um projeto [Fox et al. 2008]. Métodos Ágeis esforçam-se para entregar pequenos conjuntos de funcionalidades de software o mais rápido possível em pequenas iterações, ao passo que *User-Centered Design* (UCD) defende um investimento considerável de esforço em pesquisa e análise antes do início do desenvolvimento propriamente dito.

Portanto, tanto o desenvolvimento Ágil quanto o UX *design* visam a qualidade do produto de software, mesmo que seus conceitos de qualidade sejam diferentes.

Apesar de suas diferenças, principalmente em relação ao levantamento de requisitos e projeto antecipado (*up front design*), estas duas abordagens possuem semelhanças e, quando integradas, apresentam benefícios no que diz respeito a qualidade de software [Fox et al. 2008]. A principal semelhança é que ambas as abordagens são centradas no

usuário e/ou cliente. Como o nome sugere, UCD focaliza no desenvolvimento de software com o usuário em mente. Abordagens Ágeis envolvem um representante local do cliente para encurtar o ciclo de *feedback* entre a equipe de desenvolvimento e o cliente. Métodos Ágeis permitem que as equipes de desenvolvimento construam um software que é valorizado pelo cliente, abordagens centradas no usuário permitem que as equipes de desenvolvimento construam um software utilizável pelo cliente e/ou usuário.

Até alguns anos atrás, pouca atenção era direcionada à integração de UX e o desenvolvimento Ágil. Isto pode ter ocorrido devido a questões históricas, como, por exemplo, o não entendimento da necessidade de uma boa experiência do usuário por parte dos desenvolvedores e, pelo não conhecimento de métodos Ágeis de forma detalhada pelos *designers* de UX ou ainda, apenas pelo pensamento de que tais abordagens não funcionariam em conjunto devido às suas diferenças de foco.

No entanto, de acordo com [Ferreira et al. 2010], atualmente existe um crescente interesse sobre este tema e um número razoável de estudos que abordam esta integração, como pode ser visto em [Silva da Silva et al. 2011]. No entanto, sabe-se que esta integração do desenvolvimento Ágil com UX não é adequadamente tratada [Hussain et al. 2009b].

O objetivo deste trabalho é apresentar lições aprendidas no decorrer da construção de uma proposta de integração de UX com o desenvolvimento Ágil, apresentando assim, tanto práticas que devem ser adotadas como armadilhas que devem ser evitadas.

As lições aprendidas, apresentadas na Seção 3, foram extraídas de estudos exploratórios realizados em duas empresas praticantes de métodos Ágeis e que possuem a usabilidade de seus produtos como um de seus principais focos. Nestes estudos, projetos foram observados, entrevistas foram conduzidas e, descobertas e conclusões foram confirmadas com os times envolvidos. Como resultado, tem-se um conjunto de práticas e artefatos a serem utilizados para facilitar esta integração, de forma que nem a experiência do usuário fique prejudicada e nem os princípios Ágeis sejam feridos.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta conceitos básicos e trabalhos relacionados à inserção de UX em ambientes Ágeis; a Seção 3 apresenta as lições aprendidas e sua relação com a taxonomia apresentada na Seção 2; a Seção 4 traz à tona uma breve discussão sobre o tema bem como as considerações finais sobre o trabalho.

2. Conceitos Básicos e Trabalhos Relacionados

[Chamberlain et al. 2006] apresentam um *framework* a ser utilizado por equipes que objetivam integrar práticas de UX com o desenvolvimento Ágil. Os autores apresentam algumas similaridades entre UX e métodos Ágeis com base na literatura e em um estudo observacional. Baseados em seus resultados, eles sugerem cinco princípios para o sucesso da integração de UX e desenvolvimento Ágil: (*i*) os usuários devem estar envolvidos no processo de desenvolvimento de software; (*ii*) *designers* e desenvolvedores devem estar dispostos a trabalhar juntos e comunicar-se de forma muito estreita; (*iii*) designers devem estar dispostos a alimentar os desenvolvedores com protótipos e *feedback* dos usuários; (*iv*) deve ser dado tempo suficiente aos designers para que estes possam cobrir as necessidades fundamentais dos usuários antes de qualquer código; (*v*) a integração Agile e UX deve ser apoiada por uma estrutura aderente de gerenciamento.

[Ferreira et al. 2007] afirmam que a integração entre o UX *design* e o desenvolvimento Ágil não é bem entendida e apresentam um estudo qualitativo (*grounded theory*) de projetos Ágeis reais envolvendo UX *designs* de UX. Alguns resultados deste estudo são que o uso do desenvolvimento iterativo - uma prática Ágil - facilita a realização de testes de usabilidade, permitindo que os desenvolvedores incorporem os resultados destes testes em iterações subsequentes, e podem melhorar significativamente a comunicação e relacionamento entre os UX *designers* e os desenvolvedores.

[McInerney and Maurer 2005] realizaram entrevistas com UX *designers* envolvidos em projetos Ágeis e concluíram que os relatos de todos *designers* foram positivos. Estes autores também relataram que a literatura atual indica que melhorar a compreensão de como coordenar e integrar o trabalho de UX *designers* e desenvolvedores Ágeis auxilia no preenchimento da lacuna entre as disciplinas de Engenharia de Software e Interação Humano-Computador (IHC).

Ainda, de acordo com [Ferreira et al. 2011], o problema de ter desenvolvedores Ágeis e *designers* de UX trabalhando juntos em um projeto de software tem sido tipicamente caracterizado como um problema de fusão de um método com o outro. Por exemplo, [Patton 2002] explica como o Design centrado no Uso pode ser combinado com *eXtreme Programming* (XP); [Obendorf and Finck 2008] explica como *Scenario-Based Usability Engineering* (Engenharia de Usabilidade baseada em Cenários) foi combinada com XP; e [Miller 2005] explica como técnicas de Design Centrado no Usuário (UCD) foram integradas a um processo Ágil, que foi uma adaptação de *Adaptive Software Development* (ASD) e XP.

[Sy 2007] descreve adaptações realizadas em uma organização, adaptações estas que incluem ajustes de *timing* e de granularidade das investigações de usabilidade realizadas e, da forma como os possíveis problemas de usabilidade são relatados. [Sy 2007] concluiu também que a nova abordagem Ágil centrada no usuário permite a construção de produtos muito melhores do que versões projetadas utilizando abordagens em cascata. [Singh 2008] propõe um processo chamado U-Scrum (Usability-Scrum), uma adaptação do Scrum para a promoção de usabilidade, na qual existe um papel de U.P.O. (*Usability Product Owner*). [Beyer 2010] propõe um processo no qual trata a necessidade de *designers* e UX entenderem os princípios Ágeis e apresenta algumas práticas para a integração das duas áreas. Por fim, [Salah 2011] visa fornecer uma estrutura para melhoria de processo de software fornecendo diretrizes genéricas para organizações que planejam integrar UX com desenvolvimento Ágil.

Com o intuito de integrar o UX *design* e o desenvolvimento Ágil de forma harmoniosa, estudos teóricos foram realizados, os quais embasaram estudos exploratórios brevemente introduzidos na Seção 3.

A seguir, são apresentadas as principais práticas adotadas e/ou sugeridas pela literatura, e estão organizadas de acordo com a taxonomia adotada em [Silva da Silva et al. 2011].

2.1. *Little Design Up Front*

[Ambler 2006] sugere modelar a UI (*User Interface*) de forma antecipada através da utilização de ferramentas que reflitam as práticas Ágeis, como, por exemplo, *Index Cards*, esboços em quadros brancos e, protótipos de baixa fidelidade em papel, pois estes artefatos permitem iterações rápidas para coleta de informações sobre usuários.

Autores como [Hodgetts 2005], [Kollmann et al. 2009], [Chamberlain et al. 2006], [Fox et al. 2008], [Najafi and Toyoshiba 2008] e [Hudson 2003] sugerem o uso da Sprint 0 para condução de pesquisa e entrevistas.

2.2. Prototipação

Autores como [Coatta and Gosper 2011], [Sohaib and Khan 2010], [Fox et al. 2008], [Meszaros and Aston 2006], [Holzinger et al. 2005], [Detweiler 2007], [Miller 2005], [Williams and Ferguson 2007] e [Chamberlain et al. 2006] sugerem que as atividades de prototipação aconteçam nos estágios iniciais do processo de desenvolvimento. Comentam também sobre os benefícios do uso de protótipos para a melhoria da comunicação entre os UX *designers* e os desenvolvedores e, sobre o uso de tais protótipos para execução de avaliações de usabilidade, tanto por inspeção quanto com usuários. [Hussain et al. 2008b] comenta que protótipos podem ser originados de *User Stories*.

[Ungar 2008] e [Benigni et al. 2010] sugerem ainda, que os UX *designers* devem desenvolver protótipos de UI uma iteração à frente da equipe de desenvolvimento. Já [Federoff et al. 2008], sugerem que os UX *designers* trabalhem em paralelo com a equipe de desenvolvimento. No entanto, [Sy 2007] sugere que UX *designers* devem trabalhar uma iteração à frente, no que diz respeito a prototipar, mas uma iteração atrás, no que diz respeito a testar.

2.3. Teste com Usuário

[Hudson 2003], [Hussain et al. 2009a], [Meszaros and Aston 2006], [Fox et al. 2008], [Lee and McCrickard 2007], [Obendorf and Finck 2008], [Hussain et al. 2008a] e [Holzinger et al. 2005] mencionam ou sugerem a execução de testes com usuários sobre protótipos em papel.

[Miller 2005] relata a execução de testes com usuários em protótipos tanto de baixa quanto de alta fidelidade e [Illmensee and Muff 2009] mencionam que testes com usuários deve ser realizados de uma forma mais informal e não em laboratórios de usabilidade.

[Beyer et al. 2004] sugerem que a UI seja testada com usuários através de *mockups* e entrevistas pois *User Stories* são definições de funcionalidades bastante refinadas que podem ser cobertas em testes com protótipos em papel.

2.4. *User Stories*

[Jokela and Abrahamsson 2004] comentam que atividades como Análise de Tarefas de usuários deveriam contribuir para o desenvolvimento de *User Stories*.

Já [Meszaros and Aston 2006] sugerem que *User Stories* sejam originadas de testes de usabilidade sobre protótipos em papel. [Hussain et al. 2008a] e [Fox et al. 2008] comentam que *User Stories* poderiam ser refinadas para a construção de protótipos.

[Holzinger et al. 2005] sugerem que *User Stories* poderiam ser utilizadas como tarefas a serem realizadas por usuários quando da realização de testes com usuários sobre protótipos.

[Broschinsky and Baker 2008] relatam a integração de protótipos com *User Stories*. [Düchting et al. 2007] comentam que o *Product Backlog* e *User Stories* são os melhores lugares para capturar requisitos de usabilidade. Já [Singh 2008] menciona que

User Stories deveriam conter questões de usabilidade nos seus critérios de aceitação. [Beyer et al. 2004] sugerem que mockups façam parte da definição das *User Stories* e também critérios de aceitação.

2.5. Avaliação por Inspeção

Alguns autores, como [Constantine 2002], [Hudson 2003], [Hussain et al. 2009a], [Williams and Ferguson 2007], [Fox et al. 2008], [Hussain et al. 2008a], [Ungar 2008] e [Miller 2005] sugerem que avaliações de usabilidade por inspeção sejam realizadas sobre protótipos em papel, sempre com o objetivo de refiná-los para a próxima iteração.

[Albisetti 2010] relata que, em seu projeto, desenvolvedores realizaram revisões de usabilidade e que este fato mudou completamente o modo como os desenvolvedores viam o trabalho dos UX *designers*. Ter visto o trabalho dos outros a partir de uma perspectiva de alguém que não se importa o quão brilhante é o código, mas sim o que estava sendo usado pelas pessoas, parece ter tido um profundo impacto nos desenvolvedores [Albisetti 2010].

2.6. *One Sprint Ahead*

Alguns autores, como [Chamberlain et al. 2006], [Najafi and Toyoshiba 2008], [Ungar 2008], [Sy and Miller 2008] e [Williams and Ferguson 2007] sugerem que UX *designers* trabalhem uma sprint à frente do time de desenvolvimento.

[Chamberlain et al. 2006], [Najafi and Toyoshiba 2008] e [Sy and Miller 2008] também comentam que esta prática deve ter início na Sprint 0.

[Illmensee and Muff 2009] sugerem, ainda, que os UX *designers* trabalhem duas ou até mesmo três iterações à frente do resto do time, mas que devem atentar para a iteração corrente a fim de fornecer *feedback* de forma efetiva. [Cho 2009] comenta que UX é parte da estratégia do negócio, logo, é necessário que UX esteja alinhada com a análise de negócios.

3. Lições Aprendidas

As lições aqui relatadas objetivam apresentar como são tratadas, na prática, as questões levantadas por meio dos estudos teóricos resumidamente apresentados na seção anterior. Elas foram extraídas de estudos exploratórios realizados em duas organizações, como segue.

O primeiro estudo foi realizado em uma empresa de tecnologia líder mundial de fabricação e desenvolvimento de produtos colaborativos. Esta empresa adota métodos Ágeis – basicamente, Scrum – na maioria de seus projetos e tem, como um de seus principais focos, a usabilidade de seus produtos. Observações e entrevistas foram conduzidos em um time composto por sete pessoas, sendo um UX *designer*, a fim de entender o processo de desenvolvimento utilizado pela empresa e como esta tratava a integração de UX e métodos Ágeis.

Diferente do primeiro estudo, onde a intenção era identificar como a empresa como um todo tratava os aspectos do desenvolvimento Ágil com UX, o segundo estudo consistiu na análise de como as diferentes equipes – uma composta por 11 membros, sendo um *designer* gráfico e UX *designer* – de uma mesma empresa trabalhavam em

relação a esta integração. Este estudo exploratório foi realizado em uma empresa presente nos meios digitais através de grandes portais destinados a fornecer informações, serviços e oportunidades para o público das grandes cidades brasileiras, praticante de Scrum. Cabe lembrar que ambos os estudos foram guiados pela metodologia descrita em [Runeson and Höst 2009].

A seguir são apresentadas e discutidas as lições aprendidas, organizadas de acordo com a estrutura de tópicos apresentada na seção anterior.

3.1. *Little Design Up Front*

O que se viu em relação a projetar à frente do desenvolvimento foi que UX *designers*, em geral, sabem da importância de se realizar pesquisa, análise e projeto de forma antecipada ao desenvolvimento. No entanto, dificilmente isto acontece. Principalmente quando o UX *designers* possui uma carga de trabalho muito alta, trabalhando vários projetos ao mesmo tempo.

O princípio de [Chamberlain et al. 2006] sobre dar tempo suficiente aos *designers* para que estes possam cobrir as necessidades fundamentais dos usuários antes de qualquer código não casa com os princípios Ágeis. Deve-se fornecer apenas "algum" tempo para pesquisa e análise pré-desenvolvimento. Por isso os termos "little" e "some" *design* à frente do desenvolvimento. A questão é definir quanto significa este "algum" *design*. Quando questionados, UX *designers* respondem que "algum" deve ser o suficiente; mas quanto é suficiente? E a pergunta se repete, até que conclui-se que se deve pesquisar, analisar e projetar o suficiente para criar uma "Big Picture", ou seja, a visão holística do projeto; mas, ainda assim, não se deve projetar todo o *design* de forma antecipada.

Técnicas como *benchmarking*, *focus groups* e análise de concorrentes fornecem boa base para construção da *Big Picture* do projeto.

3.2. Prototipação

No que diz respeito à prototipação, todos entendem a necessidade de prototipar nas fases iniciais de processo de desenvolvimento e, inclusive, fazem uso desta prática nesta etapa.

No entanto, notou-se que a prototipação em papel, sugerida por muitos autores, pode não ser tão eficiente, principalmente quando da existência de times distribuídos. O que observou-se foi uma grande utilização de protótipos de baixa fidelidade, no entanto, digitais, e não em papel.

Observou-se também que prototipar em baixa fidelidade não pode ser uma exigência dado que UX *designers* são profissionais com formações heterogêneas; sendo alguns capazes de prototipar tanto em baixa quanto em alta fidelidade e, outros, capazes de codificar seus projetos. Logo, a técnica de prototipação a ser utilizada dependerá das habilidades do *designer*.

Ouviram-se relatos de UX *designers* de que protótipos de baixa fidelidade muitas vezes ajudam na comunicação com o time de desenvolvimento, mas às vezes não são suficientes para comunicar o *design* a alguns *stakeholders*, por exemplo.

3.3. Teste com Usuário

[Ferreira et al. 2007] comentam que a característica do desenvolvimento iterativo facilita a realização de testes de usabilidade; porém, na prática, isto não se confirmou.

A justificativa é de que testes com usuários são caros e demandam tempo para planejamento, execução e análise. Notaram-se iniciativas de execução de testes com usuários internos da organização, que não estejam envolvidos com o projeto em questão.

Entretanto, sugere-se a adoção de uma prática denominada *design partners*, proposta por [Patton 2008], a qual consiste na manutenção de uma base de possíveis usuários, parceiros com disponibilidade para execução dos testes.

3.4. *User Stories*

A inserção de questões de usabilidade como critérios de aceitação nas *User Stories*, sugerida por diversos autores, é uma ótima prática, conforme observado em um dos estudos realizados. Tal prática torna-se ainda mais eficiente quando da integração de protótipos nas *User Stories*, como sugerido por [Broschinsky and Baker 2008], facilitando, assim, o entendimento dos desenvolvedores sobre o que o *designer* está tentando expressar.

3.5. Avaliação por Inspeção

Observou-se a utilização de protótipos para realização de avaliações por inspeção, principalmente revisões por pares. Tais revisões aconteciam entre UX *designers*, ou com *Products Owners* ou ainda, com desenvolvedores, sempre com o intuito de validar idéias antes da definição do *design* e sua implementação. Não se observou realização de avaliações por inspeção sobre o produto já desenvolvido.

3.6. *One Sprint Ahead*

Observou-se que UX *designers* sabem da necessidade e importância de se trabalhar uma iteração à frente da equipe de desenvolvimento. No entanto, tal prática deve ser adotada desde o início do processo de desenvolvimento pois, uma vez que o processo tem seu início com os UX *designers* trabalhando na mesma iteração da equipe de desenvolvimento ou até mesmo, atrás da equipe de desenvolvimento, apenas testando interfaces já implementadas, não é possível implantar a prática de trabalho uma iteração à frente.

Notou-se que muitas vezes os UX *designers* tentam trabalhar uma iteração à frente da equipe de desenvolvimento. Entretanto, por estarem ocupados com outros projetos/times, não conseguem adotar esta prática.

Em um dos estudos, observou-se que aquele "algum" *design* realizado antes do início do desenvolvimento fornece uma boa base para que o UX *designer* consiga trabalhar uma iteração à frente do time de desenvolvimento. Pois tendo a visão holística, o UX *designer* consegue, pelo menos, ter idéia, do que está por vir na próxima iteração.

Para isso, é importante que o UX *designer* esteja disponível para a equipe de desenvolvimento, tanto para esclarecer as dúvidas dos desenvolvedores quanto para ter um melhor entendimento do projeto.

4. Discussão e Considerações Finais

Através dos relatos anteriores, pode-se observar que, ter o UX *designer* dedicado exclusivamente a um projeto torna-se extremamente importante. Notou-se através dos estudos que um UX *designers* trabalhando em vários projetos ao mesmo tempo, causa até atrasos na entrega, pois muitas vezes, o *designer* bloqueia o time de desenvolvimento por não estar disponível.

Um outro aspecto importante é a co-localização dos UX *designers*. Notou-se que a comunicação e colaboração entre os *designers* e os desenvolvedores é imensamente melhor quando eles trabalham no mesmo ambiente, conforme já havia sido comentado por [Ferreira et al. 2010].

É importante lembrar que em se tratando de desenvolvimento Ágil, sempre há algo a ser feito. O desenvolvimento deve ser "puxado" e não "empurrado". Partindo deste princípio e com base nos estudos realizados, sugere-se que o UX *designer* esteja sempre atento em quatro direções, como apresentado na Figura 1.

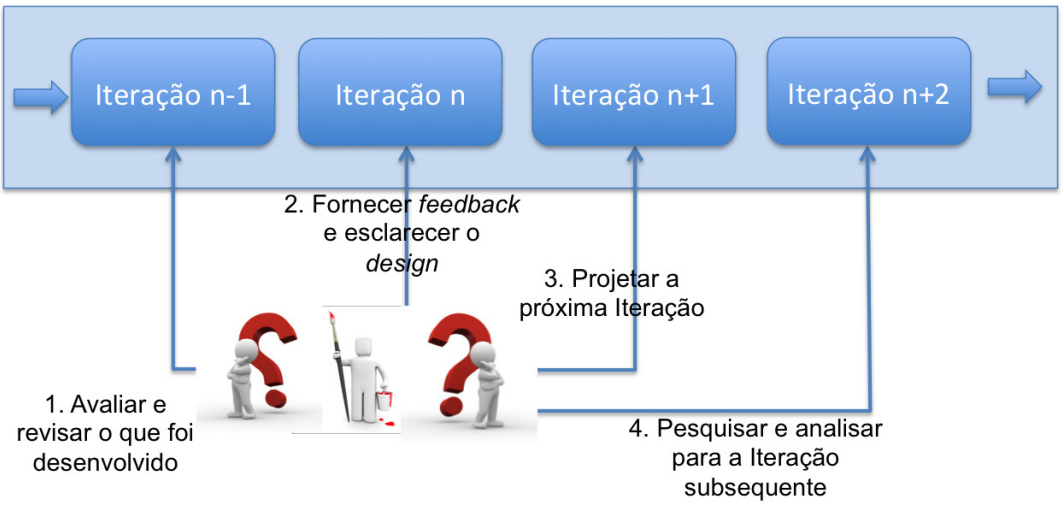


Figura 1. UX *designer* atento em quatro direções

1. Avaliar e revisar o que foi desenvolvido na Iteração *n-1*.
2. Fornecer *feedback* e esclarecer o *design* da iteração atual Iteração *n*.
3. Projetar para a próxima iteração (Iteração *n+1*).
4. Pesquisar e analisar para iteração subsequente (Iteração *n+2*).

É possível observar, também, através dos relatos da seção anterior, que todos os aspectos, apesar de apresentados separadamente, estão relacionados.

Por exemplo, a definição de algum *design* de forma antecipada está relacionada com a prototipação, que pode ser utilizada tanto para avaliação - por inspeção ou com usuários - quanto para definição de *User Stories*. Tal *design* antecipado também auxilia na manutenção da visão holística do projeto, permitindo com que UX *designers* trabalhem uma iteração à frente do time de desenvolvimento.

Este fluxo vai ao encontro das afirmações de [Chamberlain et al. 2006] de que *designers* e desenvolvedores devem estar dispostos a trabalhar juntos e comunicar-se de

forma muito estreita e, *designers* devem alimentar fornecer protótipos e *feedback* dos usuários aos desenvolvedores.

Por conseguinte, a comunicação e relacionamento entre UX *designers* e desenvolvedores melhoram, conforme mencionado por [Ferreira et al. 2007], confirmando assim, o relato de [Sy 2007], que afirma que a integração de UX com o desenvolvimento Ágil permite a construção de produtos muito melhor projetados.

Assim como toda adoção de ou migração para o desenvolvimento Ágil, a grande mudança está nas pessoas e na cultura adotada. Pôde-se notar que em todos os aspectos, a *Close Collaboration* esá envolvida. Portanto, se não houver uma colaboração muito próxima entre todos os membros envolvidos, principalmente desenvolvedores, UX *designers* e analistas de negócios, dificilmente ocorrerá uma integração eficiente entre UX e desenvolvimento Ágil.

Por fim, conclui-se que, apesar da existência de vários estudos visando definir metodologias para a integração de UX e o desenvolvimento Ágil, não é possível a definição de um padrão para tal integração, dado que determinadas práticas funcionam em certos contextos. Pode-se sugerir diretrizes e melhores práticas para que as equipes e/ou organizações possam adaptar tais práticas para o seu contexto de trabalho.

Agradecimenos

Gostaríamos de agradecer ao Dr. Frank Maurer e à Dra. Angela Martin pela colaboração e, às empresas onde os estudos foram realizados pela oportunidade.

Referências

[Albisetti 2010] Albisetti, M. (2010). Launchpad's quest for a better and agile user interface. In Sillitti, A., Martin, A., Wang, X., and Whitworth, E., editors, *XP*, volume 48 of *Lecture Notes in Business Information Processing*, pages 244–250. Springer.

[Ambler 2006] Ambler, S. (2006). Bridging the gap - agile software development and usability. *Dr.Dobb's - The World of Software Development*.

[Benigni et al. 2010] Benigni, G., Gervasi, O., Passeri, F. L., and Kim, T. H. (2010). Usabagile: A web agile usability approach for web site design. *Computational Science and Its Applications–ICCSA 2010*, 6017:422–431.

[Beyer 2010] Beyer, H. (2010). *User-Centered Agile Methods*. Morgan & Claypool Publishers.

[Beyer et al. 2004] Beyer, H., Holtzblatt, K., and Baker, L. (2004). An agile customer-centered method: Rapid contextual design. In *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, pages 527–554.

[Broschinsky and Baker 2008] Broschinsky, D. and Baker, L. (2008). Using persona with xp at landesk software, an avocent company. In *Proceedings of the Agile 2008*, pages 543–548, Washington, DC, USA. IEEE Computer Society.

[Chamberlain et al. 2006] Chamberlain, S., Sharp, H., and Maiden, N. (2006). Towards a framework for integrating agile development and user-centred design. In Abrahamsson, P., Marchesi, M., and Succi, G., editors, *Extreme Programming and Agile Proces-*

ses in Software Engineering: 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006. Proceedings, pages 143–153. Springer Berlin.

[Cho 2009] Cho, L. (2009). Adopting an agile culture. In *Proceedings of the 2009 Agile Conference*, AGILE '09, pages 400–403, Washington, DC, USA. IEEE Computer Society.

[Coatta and Gosper 2011] Coatta, T. and Gosper, J. (2011). Ux design and agile: a natural fit? *Commun. ACM*, 54:54–60.

[Constantine 2002] Constantine, L. L. (2002). Process agility and software usability: Toward lightweight usage-centered design. *Information Age*, 1.

[Detweiler 2007] Detweiler, M. (2007). Managing ucd within agile projects. *interactions*, 14:40–42.

[Düchting et al. 2007] Düchting, M., Zimmermann, D., and Nebe, K. (2007). Incorporating user centered requirement engineering into agile software development. In *Proceedings of the 12th international conference on Human-computer interaction: interaction design and usability*, HCI'07, pages 58–67, Berlin, Heidelberg. Springer-Verlag.

[Federoff et al. 2008] Federoff, M., Villamor, C., Miller, L., Patton, J., Rosenstein, A., Baxter, K., and Kelkar, K. (2008). Extreme usability: adapting research approaches for agile development. In *CHI '08 extended abstracts on Human factors in computing systems*, CHI EA '08, pages 2269–2272, New York, NY, USA. ACM.

[Ferreira et al. 2007] Ferreira, J., Noble, J., and Biddle, R. (2007). Agile development iterations and ui design. In *Proceedings of the AGILE 2007*, pages 50–58, Washington, DC, USA. IEEE Computer Society.

[Ferreira et al. 2010] Ferreira, J., Sharp, H., and Robinson, H. (2010). Values and assumptions shaping agile development and user experience design in practice. In *XP*, pages 178–183.

[Ferreira et al. 2011] Ferreira, J., Sharp, H., and Robinson, H. (2011). User experience design and agile development: managing cooperation through articulation work. *Softw. Pract. Exper.*, 41:963–974.

[Fox et al. 2008] Fox, D., Sillito, J., and Maurer, F. (2008). Agile methods and user-centered design: How these two methodologies are being successfully integrated in industry. In *Agile, 2008. AGILE '08. Conference*, pages 63 –72.

[Hodgetts 2005] Hodgetts, P. (2005). Experiences integrating sophisticated user experience design practices into agile processes. In *Proceedings of the Agile Development Conference*, pages 235–242, Washington, DC, USA. IEEE Computer Society.

[Holzinger et al. 2005] Holzinger, A., Errath, M., Searle, G., Thurnher, B., and Slany, W. (2005). From extreme programming and usability engineering to extreme usability in software engineering education (xp+ue→xu). In *Proceedings of the 29th annual international conference on Computer software and applications conference*, COMPSAC-W'05, pages 169–172, Washington, DC, USA. IEEE Computer Society.

[Hudson 2003] Hudson, W. (2003). Adopting user-centered design within an agile process: a conversation. *Cutter IT Journal*.

[Hussain et al. 2008a] Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., Vlk, T., and Wolkerstorfer, P. (2008a). User interface design for a mobile multimedia application: An iterative approach. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, ACHI '08, pages 189–194, Washington, DC, USA. IEEE Computer Society.

[Hussain et al. 2008b] Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., and Wolkerstorfer, P. (2008b). Agile user-centered design applied to a mobile multimedia streaming application. In *Proceedings of the 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for Education and Work*, USAB '08, pages 313–330, Berlin, Heidelberg. Springer-Verlag.

[Hussain et al. 2009a] Hussain, Z., Milchrahm, H., Shahzad, S., Slany, W., Tscheligi, M., and Wolkerstorfer, P. (2009a). Integration of extreme programming and user-centered design: Lessons learned. *Agile Processes in Software Engineering and Extreme Programming*, 31:174–179.

[Hussain et al. 2009b] Hussain, Z., Slany, W., and Holzinger, A. (2009b). Current state of agile user-centered design: A survey. In *Proceedings of the 5th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society on HCI and Usability for e-Inclusion*, USAB '09, pages 416–427, Berlin, Heidelberg. Springer-Verlag.

[Illmensee and Muff 2009] Illmensee, T. and Muff, A. (2009). 5 users every friday: A case study in applied research. In *Proceedings of the 2009 Agile Conference*, AGILE '09, pages 404–409, Washington, DC, USA. IEEE Computer Society.

[Jokela and Abrahamsson 2004] Jokela, T. and Abrahamsson, P. (2004). Usability assessment of an extreme programming project: Close co-operation with the customer does not equal to good usability. In Bomarius, F. and Iida, H., editors, *PROFES*, volume 3009 of *Lecture Notes in Computer Science*, pages 393–407. Springer.

[Kollmann et al. 2009] Kollmann, J., Sharp, H., and Blandford, A. (2009). The importance of identity and vision to user experience designers on agile projects. In *Proceedings of the 2009 Agile Conference*, AGILE '09, pages 11–18, Washington, DC, USA. IEEE Computer Society.

[Lee and McCrickard 2007] Lee, J. C. and McCrickard, D. S. (2007). Towards extreme(ly) usable software: Exploring tensions between usability and agile software development. In *Proceedings of the AGILE 2007*, pages 59–71, Washington, DC, USA. IEEE Computer Society.

[McInerney and Maurer 2005] McInerney, P. and Maurer, F. (2005). Ucd in agile projects: dream team or odd couple? *interactions*, 12:19–23.

[Meszaros and Aston 2006] Meszaros, G. and Aston, J. (2006). Adding usability testing to an agile project. In *Proceedings of the conference on AGILE 2006*, pages 289–294, Washington, DC, USA. IEEE Computer Society.

[Miller 2005] Miller, L. (2005). Case study of customer input for a successful product. In *Proceedings of the Agile Development Conference*, pages 225–234, Washington, DC, USA. IEEE Computer Society.

[Najafi and Toyoshiba 2008] Najafi, M. and Toyoshiba, L. (2008). Two case studies of user experience design and agile development. In *Proceedings of the Agile 2008*, pages 531–536, Washington, DC, USA. IEEE Computer Society.

[Obendorf and Finck 2008] Obendorf, H. and Finck, M. (2008). Scenario-based usability engineering techniques in agile development processes. In *CHI '08 extended abstracts on Human factors in computing systems*, pages 2159–2166, New York, NY, USA. ACM.

[Patton 2002] Patton, J. (2002). Designing requirements : Incorporating usage-centered design into an agile sw development process. *Reading*, pages 95–102.

[Patton 2008] Patton, J. (2008). Twelve emerging best practices for adding ux work to agile development. http://agileproductdesign.com/blog/emerging_best_agile_ux_practice.html.

[Runeson and Höst 2009] Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Softw. Engg.*, 14(2):131–164.

[Salah 2011] Salah, D. (2011). A framework for the integration of user centered design and agile software development processes. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 1132–1133. ACM.

[Silva da Silva et al. 2011] Silva da Silva, T., Martin, A., Maurer, F., and Silveira, M. (2011). User-centered design and agile methods: A systematic review. In *Proceedings of the Agile 2011*, pages 77–86.

[Singh 2008] Singh, M. (2008). U-scrum: An agile methodology for promoting usability. In *Proceedings of the Agile 2008*, pages 555–560, Washington, DC, USA. IEEE Computer Society.

[Sohaib and Khan 2010] Sohaib, O. and Khan, K. (2010). Integrating usability engineering and agile software development : A literature review. *Computer Design and Applications ICCDA 2010 International Conference on*, 2.

[Sy 2007] Sy, D. (2007). Adapting usability investigations for agile user-centered design. *Journal of Usability Studies*, 2(3):112–132.

[Sy and Miller 2008] Sy, D. and Miller, L. (2008). Optimizing agile user-centred design. In *CHI '08 extended abstracts on Human factors in computing systems*, pages 3897–3900, New York, NY, USA. ACM.

[Ungar 2008] Ungar, J. (2008). The design studio: Interface design for agile teams. In *Proceedings of the Agile 2008*, pages 519–524, Washington, DC, USA. IEEE Computer Society.

[Williams and Ferguson 2007] Williams, H. and Ferguson, A. (2007). The ucd perspective: Before and after agile. In *AGILE 2007*, pages 285 –290.

Studying agile organizational design to sustain innovation

Celina Maki Takemura[1], **Claudia de O. Melo** [2]

[1]Brazilian Agriculture Research Corporation - Embrapa
Satellite Monitoring Research Center, Campinas, Brazil

[2]Department of Computer Science, University of São Paulo, São Paulo, Brazil

celina@cnpm.embrapa.br, claudia@ime.usp.br

Abstract. *Innovation is a core part of software development companies, frequently determined by organizational design variables including structure, capacity for learning, for change and adaptation. Agile software methods have evolved as approaches to promote agility and innovativeness in software development organizations. However, little research has examined organizational innovativeness and its relationship with organizational design and adoption of agile methods. In this work, we propose a conceptual framework to characterize innovation's prone and averse patterns on organizational design in agile companies by measuring diffusion and integration of technologies and practices within individual, team, organizational, and environmental levels.*

1. Introduction

Innovation is fundamental to economic growth, the creation of new industries and businesses, and competitive advantage of organizations (Damanpour and Aravind, 2012). Innovation is more than a creative process. It includes implementation and application. Its scope takes in not only product and service, but process, marketing, organizational design and practices. Therefore, it is not only related to Research and Development (R&D) activities. Thus, innovation comprises internally conceived and externally adopted innovation. Moreover, it is relative, i.e., an innovation may be commonplace in other organizations, but would still be considered innovation if it were a novelty to the researched subject.

The extent of software contribution to innovation is widely studied (Pikkarainen et al., 2011). Although software development, *per se*, is not considered an innovation activity, it copes with the various dimensions of innovation. Software development is a R&D activity if its completion is determined by a scientific and/or technological advance. Software acquisition or deployment may also be related to a new process or a new marketing methodology, for example. Moreover, it may represent an innovation on organizational design.

Propensity to innovation or to the adoption of innovations, a dynamic capability, i.e, **innovativeness**, is a non-tangible variable, however it is dependent on a number of innovation determinants. In literature we find innovation determinants expressed over three levels, i.e., individual level, team level and organizational level that may bond with innovative processes surrounding research. Thus, there is an environmental aspect underlying innovation (economic, political, etc.).

On the other hand, by definition, **agility** is the ability to both create and respond to change in order to profit in a turbulent business environment. Agile software development

recognizes the value of team member competencies and diversity when bringing agility and innovativeness to development processes (Nerur and Balijepally, 2007). Abrahamsson et al. (2009) points that supporting evidence relating the true ability of agile methods and practices and innovation is much needed.

However, while most agile methods advocate their adoption to facilitate innovation, there is a lack of rigorous research evaluating innovation in an agile context (Moe et al., 2012), particularly within the organizational design.

The intent of this research is to establish the basis of innovation and organizational design analysis on agile software development companies on grounds of the present consensus on innovation, agility and organization design research. It is not our purpose that our ideas represent an absolute renewal or evidentiate outright originality. We present our ideas as a complete formulation of concepts that researchers of these three areas have individually elaborated progressively and spontaneously.

Therefore, the aim of this paper is to develop a conceptual framework to characterize innovation's prone and averse patterns on organizational design in agile software development companies. The main issue is to recognize the multiple dimensions of innovation and normalize language and practices. Our framework is a guideline to evaluate innovation in agile companies at the organizational level by measuring diffusion and integration of technologies and practices, i.e., individual, team, organizational, and environmental forms combined together.

The remainder of this paper is organized as follows. Section 2 presents a review of innovation and innovation activities concepts; Section 3 synthesizes research on how agile methods foster innovation; Section 4 summarizes how organizational design is related to innovation and agility; Section 5 presents a conceptual framework with links among its components, enabling research on agile organizational design to sustain innovation; Section 6 concludes and provides suggestions for further work.

2. The innovation and innovation activities concepts

According to the Oslo Manual (Oecd, 2005), **innovation** is the implementation of a new or significantly improved product (good or service), or process, a new marketing method, or a new organizational method in business practices, workplace organization or external relations. **Innovation activities** are all scientific, technological, organizational, financial and commercial steps which actually lead, or are intended to lead, to the implementation of innovations. Some innovation activities are themselves innovative, others are not novel activities but are necessary for the implementation of innovations. Innovation activities also include R&D that is not directly related to the development of a specific innovation. Moreover, an innovative firm is one that implemented an innovation (Oecd, 2005).

Software development is classified as R&D, and therefore an innovation activity, if its completion is dependent on a scientific and/or technological advance, and the aim of the project is the systematic resolution of a scientific and/or technological uncertainty. Furthermore, services development is classified as R&D if it results in new knowledge or involves the use of new knowledge to devise new applications (Oecd, 2002).

Finally, it is possible to visualize two different types of innovation processes. The first one, the *generation* of innovation, results in an outcome - a product, service, or tech-

nology - that is at least new to an organizational population. The second one is innovation *adoption*, which results in the assimilation of a product, service, or technology that is new to the adopting organization (Oecd, 2002).

3. Agile software development practices fostering innovation

Agile software development is defined as a business of innovation, emphasizing practices such as feature planning and dynamic prioritization; feedback and change; and focus on teamwork (Highsmith and Cockburn, 2001). Agile assumes that change is not only inevitable, but also necessary to foster innovation and adaptation (Vinekar et al., 2006).

Conboy et al. (2011) presented a review of the current state of innovation in agile development focusing on the creativity aspect, that is the ability to produce work that is considered novel, appropriate and adaptive. Creativity has been advocated by agilists as "the only way to manage complex software development problems"(Highsmith and Cockburn, 2001). However, there is a lack of understanding of what constitutes creativity and innovation in software development in general, and to what extent agile methods actually facilitate these processes. Creativity by individuals and teams is the starting point for innovation; creativity might be necessary, but is not sufficient for innovation to occur (Amabile, 1988).

Conboy et al. (2011) argue that agile teams need to include multiple stakeholders outside the business unit in order to promote intra-organizational innovation, identifying new concepts for products, processes, marketing methods or organizational changes. The whole team practice propitiates communication and interaction among the different roles, favouring the identification of "new".

Moreover, the role of the agile coach is part embedded trainer, part consultant – an advisor. Even the best agile training courses cannot cover every detail or eventuality a team will encounter. The coach is there to continue the training after the formal classes are over. Consequently, agile development promote human skills by means of development (through internal training) or purchase (by hiring). Tacit and informal learning - "learning by doing" - are also involved in agile practices as pair programming, continuous peer review, job rotation, TDD (immediate feedback after mistakes).

Agile promotes double-loop learning, intertwining thought and action, critical reflection and learning after action. This "generative" learning process increases both learning and the ability to innovate and use change to one's advantage (Nerur and Balijepally, 2007). Practices such as continuous integration, refactoring, retrospectives and stand-up promote the feedback from team members and stakeholders, fostering the double-loop learning. As a counterpoint, Moe et al. (2012) found that simply adopting agile practices is not sufficient to maintain innovative edge. Agile practices were found to support only two antecedents of innovation: empowerment and knowledge management.

4. Organizational determinants of innovation capabilities in agile software companies

As argued by Lam (2004), there are three innovation determinants over organization design: a) the relationship between organizational structural forms and innovativeness; b) innovation as a process of organizational learning and knowledge creation; and c) organizational capacity for change and adaptation. Environmental determinants might mediate

the relationship between organizational determinants and innovation, as well as directly influence innovativeness (Crossan and Apaydin, 2010).

4.1. Organizational structural forms and innovativeness

Different organizational arrangements are suited to different types of competitive environments and differing types of innovation. For instance, Japanese firms are said to have gained a competitive advantage because of their superior organizational capacity for integrating shop-floor workers and enterprise networks, enabling them to plan and coordinate specialized divisions of labour and innovative investment strategies (Nonaka and Takeuchi, 1995). However, this Japanese model of organizational integration ('**J-form**') works well in established technological fields in which incremental innovation is important, but not necessarily in rapidly developing new fields where radical innovation is vital for competitiveness, i.e., focusing mainly on the innovation generation process.

By contrast, **adhocracy** Mintzberg (1980) focuses on the adoption process, thriving on information acquisition and fostering risk taking. Moreover, adhocracies tend to rely more upon individual specialist expertise organized in flexible market-based project teams capable of speedy responses to changes in knowledge and skills, and on integrating new kinds of expertise to generate radical new products and processes. Both the J-form and adhocracy are learning organizations with strong innovative capabilities, but they differ markedly in their structural forms, patterns of learning and in the type of innovative competences generated (Lam, 2004).

To compete, companies must continually pursue many types of innovation aimed at incremental and radical innovations as well as existing and new customers. **Ambidextrous organizations** encompass these two profoundly different types of businesses - those focused on exploiting existing capabilities for profit and those focused on exploring new opportunities for growth. Therefore, ambidextrous organizations encompass the generation and adoption processes of innovation.

Other structural archetypes and their innovative potentials are studied by Mintzberg (1980) and retrieved by Lam (2004) are simple structure, machine bureaucracy, professional bureaucracy and divisionalized form. They classify as simple structure an organic, centrally controlled firm that can respond quickly to changes in the environment. On the other hand, machine bureaucracy is a mechanistic organization characterized by a high level of specialization, standardization and centralized control. It is designed for efficiency and stability, thus good at dealing with routine problems, but highly rigid and unable to cope with novelty and change.

The **professional bureaucracy** is characterized by a decentralized mechanistic form which accords a high degree of autonomy to individual professionals, and by functional specialization, with a concentration of power and status in the "authorized experts'. The **divisionalized form** is composed by quasi-autonomous entities loosely coupled together by a central administrative structure. It is typically associated with larger organizations designed to meet local environmental challenges, in which have the ability to concentrate on developing competency in specific niches.

4.2. Innovation as a process of organizational learning and knowledge creation

Some organizational researchers regard innovation as a process of bringing new, problem-solving, ideas into use (Amabile, 1988, Lam, 2010). An innovative organization is one

that is intelligent and creative, capable of learning effectively (Argyris and Schön, 1978) and creating new knowledge (Nonaka and Takeuchi, 1995).

Gassmann and Enkel (2004) define three core innovative capabilities: (a) Absorptive capability related to the outside-in process; (b) Multiplicative capability related to inside-out process; (c) Relational capacity related to coupled process - an ambidextrous organization.

The *absorptive capability* covers all efforts and activities aimed at creating new ideas and getting them to work. Considering organizations, the learning goal is to gather new, unrelated knowledge to create untapped future opportunities (Cohen and Levinthal, 1990). The *multiplicative capability*, on the other hand, is usually a problem-solving process in which an existing idea is adapted to address the recognized needs and identified problems within an organization. The learning goal for organizations here is to gain new knowledge that is related to current areas of expertise, in order to advance the organization's existing technologies and products (Cohen and Levinthal, 1990).

The innovation-generating organization depends more heavily on its technological knowledge and market capabilities to develop and commercialize innovations. The innovation-adopting organization relies more on its managerial and organizational capabilities to select and assimilate innovations. Therefore, although organizational units pursuing exploration are expected to be small and decentralized with loose process, organizational units that pursue exploitation are expected to be larger, more decentralized, and with tight processes.

Tushman and Smith (2002) describe incremental innovations as exploitative and radical innovations as explorative. Ambidexterity is the ability to simultaneously pursue both incremental and discontinuous innovation, and has been proposed as a viable solution to balance agile and traditional systems' development while maintaining the necessary organizational cultures for each approach (Vinekar et al., 2006). Most of the solutions to cope with ambidexterity are related to two basic underlying concepts: spatial separation (at the business unit or corporate level) and parallel structures.

Agility is the ability to both create and respond to change in order to profit in a turbulent business environment; it is the ability to balance flexibility and stability (Highsmith, 2004). According to Lyytinen and Rose (2006), it is the ability to sense and respond swiftly to technical changes and new business opportunities; it is enacted by exploration-based learning and exploitation-based learning. By definition, agile teams would be able to learn in both ways.

However, agile software development methods are mostly limited to the micro-context of software product design and delivery (exploitation), whereas higher-level innovation capabilities are needed during exploratory phases (such as new base technology adoption) (Lyytinen and Rose, 2006). Kettunen (2009) noted that the latest trends in agile software development seek for combinations of exploitative and explorative methods to reach ambidexterity in larger scale, i.e., in the agile organizational level. He exemplifies it describing the synthesis of agile methods principles and lean principles to support innovative capabilities.

4.3. Organizational capacity for change and adaptation

A third strand of research concerns organizational change and adaptation, and the processes underlying the creation of new organizational forms. Its main focus is to understand whether organizations can overcome inertia and adapt in the face of radical environmental shifts and technological changes, and whether organizational change occurs principally at the population level through selection (Lam, 2010).

According to Lam (2010), there are two broad theoretical views on how organizations adapt when facing the need of change. The first view is to spin out new business ventures in an environmental selection process. The second perspective views organizational change as a product of an actor's decisions and learning, rather than the outcome of a passive environmental selection process. In this work, we adopt Lam's second view.

Organizational design theories focus predominantly on the link between structural forms and the propensity of an organization to innovate (Mintzberg, 1980). Although the unit of analysis is the organization and the main research aim is to identify the structural characteristics of an innovative organization, or to explore the effects of organizational structural variables on product and process innovation, we consider two levels of innovativeness: i) creativity at the individual and team levels, and ii) innovation at the organizational unit and organizational levels.

4.3.1. Individual and team levels predictors for innovation

Creativity is an input to the innovative outcome or a part of the innovation process. Creativity requires freedom and a climate of support where individuals are unrestricted in their search for solutions (Amabile, 1988).

Individual and team creativity feed organizational innovation (Damanpour and Aravind, 2012). Amabile (1988) describes individual and team creativity determinants as: expertise, creativity skills and intrinsic task motivation. Although the two skill components determine what an individual is capable of doing in a given domain, it is the task motivation component that determines what a person will actually do.

Team processes have strong relationship with creativity and innovation. Team interaction processes like exchanging information, learning, motivating, and negotiating, as well as composition and structure, i.e. functional heterogeneity and frequency of meetings, are more strongly related to creativity and innovation measured at the team than at the individual level.

One core value of agile software development methodologies is: "individuals and interactions over processes and tool" (Beck et al., 2001). To facilitate interactions, agile methods rely on frequent inspect-and-adapt cycles. These cycles can range from every few minutes with pair programming, to every few hours with continuous integration, to every day with a daily stand-up meeting, to every iteration with a review and retrospective. To foster respect for the worth of every person, truth, transparency, trust and commitment to and within the team and to the team's goals, agile management must provide a supportive environment, team coaches must facilitate member inclusion, and team members must exhibit their commitment.

Agile methodologies facilitate commitment by encouraging teams to pull from a

prioritized work list, manage their own work, and focus on improving their work practices. This practice is the basis of self-organization, which is the driving force for achieving results in an agile team. In addition, agile principles (Beck et al., 2001) such as "close, daily co-operation between business people and developers", "face-to-face conversation is the best form of communication", "projects are built around motivated individuals, who should be trusted" and "self-organizing teams" attempt to communication issues on software development. On the other hand, the principle of "continuous attention to technical excellence and good design" is related to individual expertise. Agile methods, therefore, provide specific values, principles, and practices that reinforce team climate and freedom to be creative.

However, because of the concept of sprints/iterations and the truly customer-driver aspect of agile methodologies, it is possible that technical cutting-edge knowledge may be not used. Higman et al. (2002) approach to deal with that and the issue of individual recognition is the Gold Card System that grants the developer who has it, one day of work on a topic of their choice.

4.3.2. Organizational Design level predictors for innovation

Organizational structure is the necessary division of work and establishment of communications channels which allows an organization to reach its goal. Organizational structure is a combination of the number of layers in hierarchy, the relationship between employees and managers, the level of participation of the employees in the decision-making process, and finally the interactions between services and participants of vertical and horizontal integration. In other words, organization is the way in which a company structures itself, its partnerships and its employee roles and responsibilities.

Organizational structure innovation, i.e. innovation at the organizational level. Organizational innovation often involves rethinking the scope of the companies activities as well as redefining the roles, responsibilities and incentives of different business units and individuals (Sawhney et al., 2006).

Organizational structure and innovation generation and adoption. Understanding how social interaction and group dynamics within organizations shape collective intelligence, learning and knowledge generation yields important insights into the innovative capability of organizations. Indeed, certain organizational structures facilitate the creation of new products and processes, especially in terms of fast changing environments.

4.4. Environmental Determinants

Environment has a strong impact on an organization's ability to adapt and innovate (Damanpour and Gopalakrishnan, 1998). Different segments operate under different environmental conditions imposed on **economic**, **technological** and **political-legal** aspects related to the industry type. The degree of **uncertainty** refers to the extent of change (rate and velocity) in the business environment. Finally, **complexity** refers to the extent of complexity in the business environment.

5. Assessing the influence of organizational design and agile practices on innovativeness

To achieve our research goal, we propose a guideline to evaluate innovation in agile companies at the organizational level by measuring diffusion and integration of technologies

and practices, i.e., skill, technologies and organizational forms combined together, thus, adequately measuring innovation, incorporating both technological and non-technological dimensions. The key challenge is the need for a consensus among researchers on measures, scales, and methods of inquiring innovation, particularly on agile software development companies. The first step it to assess **innovation**, frequently measured by the protocols suggested by OECD (Oecd, 2002, 2005) .

In our work, we adapted the conceptual framework on organizational innovation developed from a systematic review conducted by Crossan and Apaydin (2010), who suggested an overarching framework that links different theoretical units into a coherent whole: individual, team, organizational, and environmental levels impacting on innovation. We also considered results from the meta-analysis conducted by Damanpour and Aravind (2012), which analyzed organizational design determinants for innovation, as well as Lam (2010) work on innovative organizations in the 21st century. We aim to describe a more coherent framework for agile organizational design's influence on innovation. Figure 1 summarizes our conceptual framework.

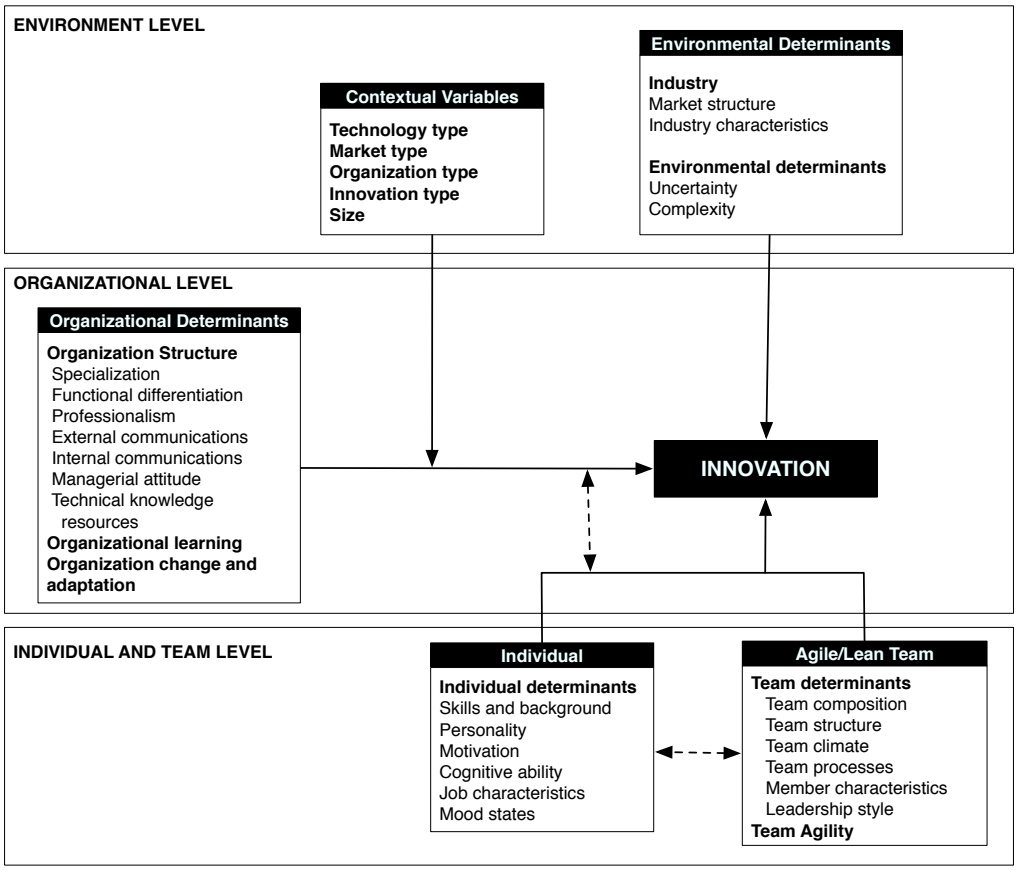


Figure 1. Conceptual framework of Agile Organizational Design's influence on Innovation

5.1. Individual and Team Level

Individual Level. It is clear that special types of structure/organization design attracts and are more prone to develop certain types of people. In addition, some tasks motivate and are better carried out by particular people. Apart from the task's level of interest, engagement and challenge, other variables like condence, clarification of personal responsibility, financial and non-nancial incentives, sense of significance are motivational items

to be evaluated. Since the key element in agile software development are people, human behavior factors may be evaluated to assess their influence on innovativeness. Here we presume that skill, background, personality, motivation, cognitive ability, job characteristics, and mood states variables can model the individual innovativeness degree. Therefore, **skills and background** may be measured as amount of education, age, tenure, diversity of background, experience, extra-industry ties (Crossan and Apaydin, 2010).

Personality is a complex set of relatively stable behavioral and emotional characteristics that can be used to uniquely identify a person. Thus, personality represents those characteristics of the person that account for consistent patterns of behavior. By means of the Myers-Briggs Type Indicator (Myers et al., 1985), it is possible to assess **cognitive style**, mental set, self-efficacy, assertiveness, tolerance to anxiety and ambiguity, etc.

Motivation. Intrinsic motivation is animated by personal enjoyment, interest, or pleasure. Researchers often contrast intrinsic motivation with extrinsic motivation, which is governed by reinforcement contingencies. In general, motivation on an industry context is analysed as **job and mood** dependent. A review on job characteristics, motivation and psychological states of software engineers team members can be found in Beecham et al. (2008). Motivation is frequently assessed using either self-report measures or rating scales completed by teachers or parents (Broussard and Garrison, 2004). Such instruments usually include questions organized under several subscales, such as interest, attributions, self-perception and self-efficacy, preference for challenge, curiosity, mastery orientation, persistence, and enjoyment of learning.

Team Level. At the team level, team design is of foremost importance, because the resources (knowledge, skills, abilities) to be innovative mainly reside with the team members. However, team processes will determine the extent to which the innovative potential of the team is fully realized. Team processes are a not tangible variable, however they depend on team structure, climate and leadership, and also on variables in the wider organizational context (Anderson et al., 2004).

Campion et al. (1993) examined 19 group-mensurable characteristics within social psychology, socio-technical theory, industrial engineering and organizational psychology variables while attempting to analyse relationships between design characteristics and effectiveness. The effectiveness criteria is given by means of productivity, satisfaction and manager judgement degrees. Our hypothesis is that the measurements developed by them are extensible to innovativeness analysis from that define the length of team composition, team structure, team processes and leadership style measurements. Moreover, they use the organization's opinion survey as measurement for employee satisfaction (over it, it is possible to derive team climate).

Qumer and Henderson-Sellers (2006) suggest agility can be described and possibly measured by the attributes: flexibility, speed, leanness, responsiveness and learning. As a further matter, **team agility** may be assessed using proposed agility taxonomies (e.g., Conboy (2009), Qumer and Henderson-Sellers (2006)) and a corresponding set of metrics. Moreover, Conboy et al. (2011) describes a set of recommendations to overcome a broad range of problems from recruitment of agile staff, to training, motivation and performance evaluation, among others.

5.2. Organizational Level

Organizational structure. Structural theories of innovation usually aim to specify organizational design characteristics that lead to innovation (Damanpour and Gopalakrishnan, 1998). Mintzberg (1980) synthesized much of the work on organizational structure and proposed a series of archetypes presented in Section 4.1. We aim to describe a company's structural configuration using his taxonomy. Burton et al. (2011) provided a step-by-step approach for assessing the organizational design based on the multi-contingency approach. Their questionnaire is one possible instrument to define organizational structure.

Organizational learning. Cohen and Levinthal (1990) argue that innovative outputs depend on the prior accumulation of knowledge that enables innovators to assimilate and exploit new knowledge. From this perspective, understanding the role of organizational learning in fostering or inhibiting innovation becomes crucially important (Lam, 2010). Assessing organizational learning is complex because of its multidimensional nature. Recent research, e.g. Jerez-Gomez et al. (2005), have proposed scales to measure Organizational Learning that can be useful to concretize our framework.

Organization capacity to change and adapt. As we discussed before, agility is the capacity to change and adapt in turbulent environments. Kettunen (2012) suggests a reference framework with a prototype tool called *Agility Profiler* for exploring and exploiting agility in software-intensive product development organizations. The tool can be used to assess the organizational degree of agility.

5.3. Environmental Level

Contextual variables. OECD Manuals (Oecd, 2002, 2005) provide a protocol to gather data regarding technology type, market type, organization type, innovation type, and company size. **Market structure and industry type.** Every country has a classification of industry type. Moreover, interest rates expectation, inflation rates, expenditure on R&D, total expenditure, patent protection, new products, antitrust regulations, protection law, tax law, special incentives, foreign trade regulation, labor law, lifestyle changes, career, consumer, rate of family formation, population growth rate, age distribution may all affect innovation drivers (Morck and Yeung, 2001).

Uncertainty. Kotha and Nair (1995) suggest four dimensions of environment to capture uncertainty: resource availability, competitive interdependence, technological change, and industry concentration. Past research implied that a mechanistic structure is more suitable for conditions of certainty when, under conditions of uncertainty, an organic structure would be more responsive to changes (Kotha and Nair, 1995).

Complexity. Some measurements of degree complexity are: geographic concentration of competitors, industry sales, and labor availability; level of products/services differentiation; geographic concentration of customers; and technological diversity used in the industry (Kotha and Nair, 1995).

6. Conclusion

In this work, we synthesized research on innovation, agility and organization design, and presented a guideline to evaluate innovation in agile companies at the organizational level. Future work is needed to both confirm and extend the usefulness of our conceptual model through empirical tests.

According to (Anderson et al., 2004), there are two important directions for future research on innovation. First, to progress our understanding of innovation as a quintessentially multi-level phenomenon, researchers should use multi-level theory. The multi-level nature of innovation is first and foremost important because different variables will influence innovative behavior at different levels. Our proposal is a multi-level conceptual framework to analyze agile software companies considering four levels: individual, team, organizational, and environmental. Second, cross-cultural differences and the international generalizability should be considered when studying innovation. We aim to conduct a multiple-case study in Brazil to explore the multi-level relationship between agile methods, organizational design and innovation using the proposed conceptual framework as a guideline.

Finally, since innovation is an ongoing process, its consequences cannot be truly detected unless theoretical models explain composite effects of innovations over time. Damanpour and Aravind (2012) recommend that research on innovation must rely on longitudinal analytical methods to examine these conceptual models. Thus, we suggest that our framework should be applied in longitudinal studies, both exploratory and confirmatory.

7. Acknowledgements

This research is supported by FAPESP, Brazil, proc. 2009/10338-3.

References

Abrahamsson, P., Conboy, K., and Wang, X. (2009). 'lots done, more to do': the current state of agile systems development research. *EJIS*, 18(4):281–284.

Amabile, T. M. (1988). A model of creativity and innovation in organizations. In Staw, B. M. and Cummings., L. L., editors, *Research in organizational behavior: an annual series of analytical essays and critical reviews*, volume 10. Greenwich, Conn.: J.A.I. Press.

Anderson, N., De Dreu, C. K. W., and Nijstad, B. A. (2004). The routinization of innovation research: a constructively critical review of the state-of-the-science. *Journal of Organizational Behavior*, 25(2):147–173.

Argyris, C. and Schön, D. (1978). *Organizational learning: a theory of action perspective*. Addison-Wesley series on organization development. Addison-Wesley Pub. Co.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. http://agilemanifesto.org/.

Beecham, S., Baddoo, N., Hall, T., Robinson, H., and Sharp, H. (2008). Motivation in software engineering: A systematic literature review. *Information & Software Technology*, 50(9-10):860–878.

Broussard, S. C. and Garrison, M. E. B. (2004). The relationship between classroom motivation and academic achievement in elementary-school-aged children. *Family and Consumer Sciences Research Journal*, 33(2):106–120.

Burton, R., Obel, B., and DeSanctis, G. (2011). *Organizational Design: A Step-By-Step Approach*. Cambridge University Press.

Campion, M. A., Medsker, G. J., and Higgs, A. C. (1993). Relations between work group characteristics and effectiveness: Implications for designing effective work groups. *Personnel Psychology*, 46(4):823–847.

Cohen, W. M. and Levinthal, D. A. (1990). Absorptive capacity: A new perspective on learning and innovation. *Administrative Science Quarterly*, 35(1):128–152.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, 20(3):329–354.

Conboy, K., Coyle, S., Wang, X., and Pikkarainen, M. (2011). People over process: Key challenges in agile development. *Software, IEEE*, 28(4):48 –57.

Crossan, M. M. and Apaydin, M. (2010). A multi-dimensional framework of organizational innovation: A systematic review of the literature. *Journal of Management Studies*, 47(6):1154–1191.

Damanpour, F. and Aravind, D. (2012). Organizational structure and innovation revisited: From organic to ambidextrous structure. In *Handbook of Organizational Creativity*. Academic Press.

Damanpour, F. and Gopalakrishnan, S. (1998). Theories of organizational structure and innovation adoption: the role of environmental change. *Journal of Engineering and Technology Management*, 15(1):1–24.

Gassmann, O. and Enkel, E. (2004). Towards a theory of open innovation : Three core process archetypes. *Innovation*, 1:1–18.

Highsmith, J. (2004). *Agile Project Management - Creating Innovative Products*. Pearson Education.

Highsmith, J. and Cockburn, A. (2001). Agile software development: The business of innovation. *IEEE Computer*, 34(9):120–122.

Higman, J., Mackinnon, T., Moore, I., and Pierce, D. (2002). Innovation and sustainability with gold cards. In Marchesi, M., Succi, G., Wells, D., and Williams, L., editors, *Extreme Programming Perspectives*. Addison-Wesley Professional.

Jerez-Gomez, P., Cspedes-Lorente, J., and Valle-Cabrera, R. (2005). Organizational learning capability: a proposal of measurement. *Journal of Business Research*, 58(6):715 – 725. Special Section: The Nonprofit Marketing Landscape.

Kettunen, P. (2009). *Agile software development in large-scale new product development organization: team-level perspective*. PhD thesis, Helsinki University of Technology.

Kettunen, P. (2012). Systematizing software-development agility: Toward an enterprise capability improvement framework. *Journal of Enterprise Transformation*, 2(2):81–104.

Kotha, S. and Nair, A. (1995). Strategy and environment as determinants of performance: Evidence from the japanese machine tool industry. *Strategic Management Journal*, 16(7):497–518.

Lam, A. (2004). *Organizational Innovation*, pages 115–147. Oxford University Press.

Lam, A. (2010). *Innovative Organizations: Structure, Learning and Adaptation*, pages 163–175. BBVA, Spain.

Lyytinen, K. and Rose, G. M. (2006). Information system development agility as organizational learning. *EJIS*, 15(2):183–199.

Mintzberg, H. (1980). Structure in 5's: A synthesis of the research on organization design. *Management Science*, 26(3):322–341.

Moe, N. B., Barney, S., Aurum, A., Khurum, M., Wohlin, C., Barney, H. T., Gorschek, T., and Winata, M. (2012). Fostering and sustaining innovation in a fast growing agile company. In *PROFES*, pages 160–174.

Morck, R. and Yeung, B. (2001). *The economic determinants of innovation*. Industry Canada Research Publications Program.

Myers, I., McCaulley, M., and Most, R. (1985). *Manual: A guide to the development and use of the Myers-Briggs Type Indicator*. Consulting Psychologists Press Palo Alto, CA.

Nerur, S. and Balijepally, V. (2007). Theoretical reflections on agile development methodologies. *Communications of ACM*, 50:79–83.

Nonaka, I. and Takeuchi, H. (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press.

Oecd (2002). *Frascati Manual: Proposed Standard Practice for Surveys on Research and Experimental Development*, volume 1. OECD.

Oecd (2005). *Oslo manual: guidelines for collecting and interpreting innovation data.*, volume Third edition. Organisation for Economic Co-operation and Development: Statistical Office of the European Communities.

Pikkarainen, M., Codenie, W., Boucart, N., and Heredia, J. A. (2011). *The Art of Software Innovation - Eight Practice Areas to Inspire your Business*. Springer.

Qumer, A. and Henderson-Sellers, B. (2006). Crystallization of agility back to basics. In *ICSOFT (2)*, pages 121–126.

Sawhney, M., Wolcott, R. C., and Arroniz, I. (2006). The 12 different ways for companies to innovate the 12 different ways for companies to innovate. *MIT Sloan Management Review*, 47(3):128143.

Tushman, M. and Smith, W. (2002). *Blackwell Companion to Organizations*, chapter Organizational Technology: Technological Change, Ambidextrous Organizations and Organizational Evolution, pages 386–414. Blackwell Publishers.

Vinekar, V., Slinkman, C. W., and Nerur, S. P. (2006). Can agile and traditional systems development approaches coexist? an ambidextrous view. *IS Management*, 23(3):31–42.

Uma Análise de Dívida Técnica em uma Empresa de Tecnologia com Desenvolvimento baseado em Scrum

Graziela S. Tonin[1], Rogério Chaves[2], Alfredo Goldman[1], Viviane Santos[1].

[1]Instituto de Matemática Estatística e Ciência da Computação – Universidade de São Paulo (USP)
Rua do Matão, 1010 Cidade Universitária – São Paulo – SP – Brazil

[2] Instituto de Pesquisas Tecnológicas do Estado de São Paulo (IPT)

{grazzi, gold, vsantos}@ime.usp.br, rcpires@uoldiveo.com

Resumo. Até o momento existem muitas lacunas nas pesquisas científicas sobre Dívida Técnica. Assim como sobre o seu comportamento ao longo dos ciclos de vida dos projetos de software. O que as pesquisas vêm destacando é que Dívida Técnica existe e precisa ser considerada no momento de tomada de decisão. Por outro lado, tem ficado claro a eficiência e eficácia dos princípios e valores ágeis se aplicados e seguidos no gerenciamento de desenvolvimento de software. Por isso, o objetivo deste trabalho é verificar como se dá o comportamento de Dívida Técnica e seu impacto no decorrer de um projeto, em uma empresa com alto grau de maturidade em Métodos Ágeis, no caso Scrum. Para isso, realizou-se um estudo de caso em uma empresa líder de mercado no setor de tecnologia e algumas Dívidas Técnicas de impacto em determinado projeto foram identificadas e analisadas. Como resultado, ficam claras as razões pelas quais Dívidas Técnicas são contraídas, suas influências, motivações e impactos. Um modelo foi gerado e pode ser utilizado em outros projetos como base no momento de tomada de decisão para que a área de negócio não tenha que dispensar recursos com problemas recorrentes, podendo assim redirecionar os investimentos para a evolução do sistema.

Palavras Chaves: *Dívida Técnica; Tomada de Decisão Estratégica; Scrum.*

Abstract. So far there are few scientific studies about Technical Debt and their behavior over the software project life cycles. However, some works are outlining that Technical Debt exists and needs to be considered at the time to business on decision making. Moreover, it has become clear the efficiency and effectiveness of agile principles and values when applied and followed in managing software development. Therefore, the goal of this paper is to verify how is the behavior of Technical Debt and its impact over a project lifecycle in a company leader in the technology sector with a high degree of maturity in Agile Methods, Scrum in this specific case. For this, we performed a case study in a market leader in the technology sector and some Technical Debt impact on a project were identified and analyzed. As a result there are clear reasons why Technical Debts are contracted, influences, motivations and impacts. An important model that can be used in other projects based at the time of decision making so that the business does not have to dispense resources with recurring problems, so you can redirect investment to the evolution of the system.

Keywords: *Technical Debt; Strategic Decision Making; Scrum.*

1. Introdução

Dívida Técnica é uma metáfora criada por Cunningham (1992) que em seu relato de experiência do OOPSLA 1992 criou a primeira definição para o termo:

> *"... A primeira vez que a qualidade do código é comprometida é como se estivesse incorrendo em Dívida Técnica. Uma pequena Dívida Técnica acelera o desenvolvimento até que seja paga através da reescrita do código. O perigo ocorre quando a Dívida Técnica não é paga. Cada minuto em que o código é mantido em inconformidade, juros são acrescidos na forma de reimplementação..."*

Esta metáfora faz inferência a dívida financeira e vem ganhando força nos últimos anos. Onde, por uma decisão estratégica durante o processo de evolução de um sistema a qualidade do código é potencialmente comprometida devido a restrições impostas ao processo, como por exemplo, tempo/recursos. Também muitas vezes os desenvolvedores optam por comprometer a qualidade do software, para atender alguma demanda urgente em outra dimensão do sistema, como por exemplo, *time to market*. É como se estivessem contraindo uma Dívida, que precisará ser paga no futuro. Na analogia, isso implica em juros que podem crescer na forma de tempo, esforço ou custo extra de futuras alterações [Guo 2009].

A mesma tem sido tema de muitas discussões em *blogs*, fóruns, palestras e eventos. Por um lado, há vários profissionais da área de desenvolvimento de software tentando descobrir uma forma de trabalhar este conceito em suas empresas. Por outro lado, algumas pesquisas estão sendo realizadas na área, visando formalizar uma teoria a respeito [Brown 2010 e Gomes 2011]. Além disso, outras pesquisas tentam encontrar formas que possam auxiliar a identificá-la e geri-la [Guo 2009; Guo 2011; Guo 2012 e Tonin 2011], porém, ainda faltam evidências formais sobre como usar de forma eficaz o conceito de Dívida Técnica. O que pesquisas recentes mostram, é que se gastou com Dívida Técnica em 2010 aproximadamente U$500 bilhões e estima-se que até em 2015 o gasto chegará a U$1 trilhão [Gartner 2012].

Em contrapartida, pesquisas relatam que cada vez mais, nos últimos 10 anos, empresas têm adotado Metodologias Ágeis, como forma de gerir projetos de desenvolvimento de software. E que sua eficácia e benefícios têm sido significativos, onde os resultados apresentados relatam equipes mais produtivas, menos estressadas e clientes mais satisfeitos com os produtos entregues [VersionOne 2012 e Williams 2010]. Outras pesquisas evidenciam que agilidade é chave para o sucesso organizacional, onde estudos do MIT revelam que empresas Ágeis aumentam a receita 30% mais rápido e geram 37% mais lucros do que as não ágeis [HighSmith 2011]. Mas segundo HighSmith (2011), qualidade ou a falta dela continua ainda sendo a questão central para a agilidade eficaz.

Nas próximas seções será discorrido sobre os objetivos da pesquisa (Seção 2). Na seção 3, é relatado sobre a metodologia utilizada e como a mesma foi aplicada na coleta e análise dos dados. Na seção 4 os resultados são explanados onde um modelo foi criado para auxiliar a identificação e monitoração de Dívida Técnica. Na quinta seção os resultados são discutidos e comparados com os estudos existentes, coletados até o momento da escrita do artigo.

2. Objetivos da Pesquisa

Esta pesquisa foi realizada em uma empresa de grande porte que adota *Scrum* como Metodologia de Gerenciamento de Projetos há mais de quatro anos. É o maior portal de internet do Brasil e provê uma gama de produtos e serviços que atendem o mercado de internet. A empresa foi fundada em 1996 como provedor de serviços em internet focando em conteúdo e serviços discados de internet. A partir daí foi diversificando a sua gama de produtos e serviços e evoluindo juntamente com as tecnologias e práticas adotadas no mercado de internet. Possui uma área de Pesquisa e Desenvolvimento (P&D) que é responsável pela concepção, desenvolvimento e manutenção de todos os produtos e serviços providos pelo portal. Todo o desenvolvimento de software é baseado em Métodos Ágeis, com foco principal em *Scrum*.

Levando em conta que a empresa investigada segue esses princípios, buscou-se investigar como o conceito de Dívida Técnica é considerado no desenvolvimento de software. Sendo estes conceitos utilizados, procurou-se evidências para responder as seguintes questões:

(1) A empresa em estudo conhece o conceito de Dívida Técnica? Caso sim, como é considerado na gestão do projeto?

(2) Decisões Estratégicas implicaram na contração de Dívida Técnica?

(3) Qual foi o impacto ao longo do tempo?

3. Metodologia Utilizada

Realizou-se um estudo de caso exploratório, pois, segundo Merriam (2009) e Robson (2002) pesquisas qualitativas tem como foco principal entender o significado e/ou natureza de um fenômeno.

3.1. Seleção da Amostra e Caso Estudado

O caso estudado foi selecionado com base no método de amostragem intencional [Yin, 2003]. Esta abordagem é comumente utilizada, quando o objetivo da pesquisa é explorar, entender ou encontrar evidências sobre algo. Nesse método deve-se buscar por casos que possuam informações relevantes e abundantes para o problema de pesquisa. O mesmo tipo de amostragem foi utilizado para a seleção de pessoas que foram convidadas a realizar a entrevista. Logo, primeiro foi necessário encontrar um caso onde acreditava-se que haviam evidências de Dívidas Técnicas e que fosse passível de acesso devido a restrições dos negócios da empresa. Para isso, realizou-se conversas informais com o cliente de alguns projetos e com base em relatos do mesmo, definiu-se o caso a ser estudado.

3.2 Entrevistas

Optou-se pela realização de entrevistas, visto que em pesquisas qualitativas é um dos principais métodos utilizados na coleta de dados. Também não era permitido o acesso aos documentos do projeto, por se tratar de um projeto confidencial. Como foram investigados eventos passados, não seria possível observá-los. Por isso, foram realizadas

entrevistas informais com um membro do time que participou do projeto desde o início. Nestas entrevistas foram identificados momentos de solicitações de grandes mudanças e/ou mudanças de grande impacto, ao longo do ciclo de vida do projeto. De posse desses casos, foi criado um roteiro de entrevista com perguntas fechadas e abertas, seguindo os passos sugeridos por Merriam (2009), realizando assim uma entrevista semiestruturada. Essa mesma metodologia tem sido utilizada em outros trabalhos [Guo 2012; Gomes 2011 e Tonin 2011].

O roteiro foi dividido em duas partes. Em um primeiro momento explicava-se para o entrevistado o motivo da pesquisa. Em seguida, uma breve explanação era feita sobre um determinado momento do projeto para verificar se a pessoa recordava do mesmo. Em um segundo momento, eram apresentadas as questões que foram divididas em três contextos:

- Caracterizar a função/papel da pessoa naquele dado momento do projeto.
- Identificar o que a pessoa recordava sobre aquele dado momento no projeto.
- Perguntas específicas sobre decisões tomadas, problemas enfrentados e impactos gerados, naquele momento e ao longo da evolução do sistema na visão do entrevistado.

Algumas perguntas eram adicionadas ao longo das entrevistas caso o pesquisador achasse necessário para deixar clara alguma informação adicional.

3.2.1. Análise dos Dados

Segundo [Strauss 2008], basicamente há três componentes principais na pesquisa qualitativa. Primeiro há os ***dados***, que podem vir de várias fontes, tais como entrevistas, observações, documentos, registros e filmes. Depois, há os procedimentos, que os pesquisadores podem usar para interpretar e organizar os dados. Eles geralmente consistem em conceitualizar e reduzir os dados, elaborar *categorias* em termos de suas propriedades e dimensões e relacioná-las por meio de uma série de declarações preposicionais. Conceitualizar, reduzir, elaborar e relacionar sempre são referidos como *codificação*. Esses passos e princípios citados por Strauss foram seguidos para a análise e sumarização dos resultados. As entrevistas foram transcritas e conceitualizadas segundo suas propriedades e dimensões.

3.3. Casos Estudados

Os casos selecionados para serem investigados foram os seguintes:

CASO 1: *Upgrade da versão do JQuery.*

JQuery é uma biblioteca JavaScript *cross-browser* desenvolvida para simplificar os scripts *client side* que interagem com o HTML. *JQuery* é a mais popular das bibliotecas JavaScript utilizadas no mundo (http://trends.builtwith.com/javascript/JQuery). No início do projeto era utilizada a versão 1.1 do *JQuery* e com a evolução no desenvolvimento e implementação de novas funcionalidades, foram encontrados diversos *bugs* e limitações na biblioteca para atender a todos os requisitos. Os técnicos em *interface* do time fizeram algumas análises e verificaram que migrando para a versão 1.3 do *JQuery* seriam sanadas

a grande maioria dos *bugs* e seria possível implementar novas funcionalidades para aprimoramento do sistema.

CASO 2: *Definição da forma de persistência de eventos de monitoração na base de dados.*

O sistema de monitoração desenvolvido baseava-se no *framework Nagios* como plataforma de *backend* (http://www.nagios.org/). Porém, o *Nagios* tem limitações arquiteturais que limitavam a escalabilidade da ferramenta. Havia a necessidade de monitorar uma escala de milhares de dispositivos e nesses valores, não era possível utilizar somente um *backend* da ferramenta. Como forma de otimizar o desempenho do *Nagios*, optou-se por somente persistir as mudanças de status da monitoração e também não persistir as métricas coletadas nos agentes de monitoração.

CASO 3: Utilização do *MON* (https://mon.wiki.kernel.org/) como agente de monitoração na ferramenta.

"MON" é uma ferramenta que trabalha como um agente para a monitoração de disponibilidade de serviços e envio de alertas exclusivamente em servidores. Tomou-se a decisão de utilizar esse agente, pois, os principais entregáveis do projeto diziam respeito à monitoração de servidores e serviços ao qual o agente é compatível. É um componente pronto (oriundo do projeto *OpenSource*), de fácil manutenção e configuração e a curva de aprendizado para desenvolvimento de novos *plug-ins* de monitoração é pequena.

CASO 4: Ter o CMDB (*Configuration Management Database*) como mandatório na inserção de informações na ferramenta.

O desenvolvimento da ferramenta de monitoração e automação foi realizado paralelamente à implementação do ITIL (*Information Technology Infrastructure Library*) na empresa [Bon 2000]. Implementou-se uma ferramenta de gestão de problemas, mudanças e gestão de configuração (CMDB). A ferramenta de monitoração necessita de toda uma base de CI (Componentes de Configuração) e relacionamentos. Esses CI's são basicamente servidores, *racks, switches*, portas de *switches*, roteadores entre outros. A forma de se alimentar a ferramenta era exclusivamente via integração do CMDB.

4. Resultados

Em um primeiro momento com o auxílio de um profissional que participou do projeto, desde o seu início, identificou-se que a empresa, embora tenha noção do conceito de Dívida Técnica, não o considera na prática no momento de tomada de decisão estratégica e não o utiliza na gestão dos projetos. Logo, o passo seguinte foi a realização das entrevistas com cinco pessoas da equipe do projeto, das quais, apenas três estavam no projeto desde o início. Outras três pessoas consideradas chaves no momento de tomada de decisão, não participam mais do projeto e por isso, não foi possível realizar entrevistas com as mesmas. O resultado da análise dos dados e categorização dos mesmos será apresentado abaixo.

Após a conceitualização e caracterização de cada caso chegou-se ao diagrama abaixo que apresenta de forma generalista os conceitos e categorias em comum nos quatro casos estudados.

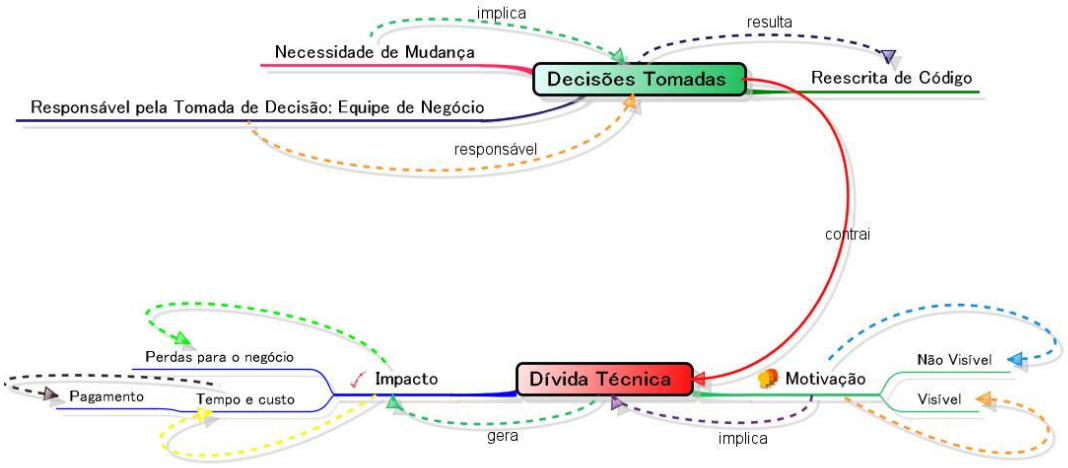


Figura 1 – Modelo resultante da caracterização e conceitualização dos casos.

Como se pode observar na **Figura 1**, todas as Dívidas estudadas foram originadas pelo surgimento de uma necessidade de mudança, seja ela oriunda por problemas de desempenho do sistema, ou por restrições do próprio negócio. Onde após essa necessidade de mudança, poucos estudos foram realizados para identificar qual seria a melhor solução. Comumente um membro da equipe com maior poder de arguição e/ou por dominar determinada tecnologia, convencia tomadores de decisão a seguir por determinado caminho. Esse processo implicou nas Dívidas Técnicas estudadas sem a percepção das pessoas envolvidas no projeto e que ao longo do tempo foram sofrendo impactos por isso, os primeiros impactos foram os atrasos na entrega como pode ser observado na **Figura 2.**

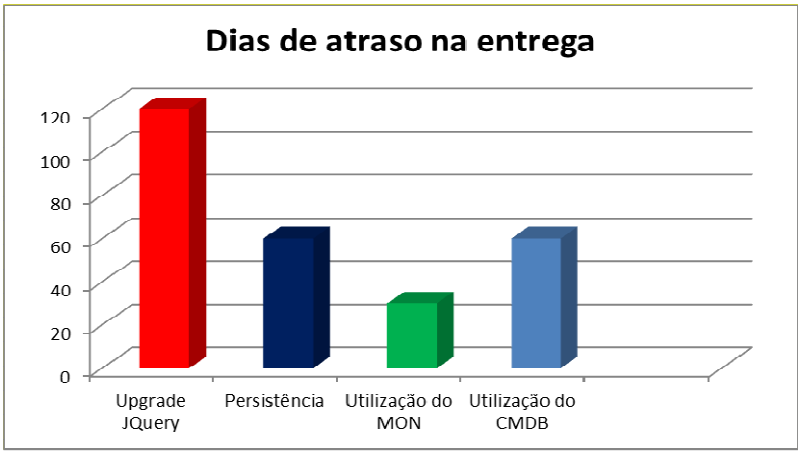


Figura 2 – Impacto em dias de atraso na entrega de cada caso.

Fica claro na **Figura 2**, que a entrega foi atrasada em muitos dias. Além disso, observou-se também, que impactos menos visíveis vêm sendo pagos até hoje na forma de reescrita de código e readequação de processo, onde muitas vezes o negócio precisa ser adaptado à ferramenta/tecnologia.

A partir do modelo criado na **Figura 1,** foi realizada a classificação de cada caso:

***Caso 1*:**

Através das evidências coletadas, identificou-se que no momento da Tomada de Decisão do *Caso 1* contraiu-se uma Dívida Técnica que ficou categorizada da seguinte forma:

- **Decisão Tomada:** *Fazer upgrade da Versão do JQuery*.
- **Influências dessa Decisão:** Colaborador com grande conhecimento técnico convenceu a todos de que seria algo rápido de se fazer e o ganho seria grande.
- **Motivação:**
  - *Visível*
    - Poderiam corrigir alguns *bugs* e ganhar em desempenho.
  - *Não Visível*
    - Técnicos convenceram que aquela era a melhor forma de fazer;
      - *Primeiro,* fizeram uma análise baseada apenas no documento de *benchmark* divulgado no site da ferramenta.
      - *Segundo, r*esolveram mudar a arquitetura de todo o *frontend* do projeto, porém, não deixaram isso claro aos outros membros da equipe e para o cliente.

- **Impactos:**
  - *Direto:*
    - Um planejamento inicialmente estimado para ser executado em cerca de 2 meses, demorou mais de 6 meses.
    - Equipe com alto grau de pressão, estresse e desgaste.
    - Cliente cobrando resultados.
  - *Indireto:*
    - Muitas outras novas funcionalidades eram desenvolvidas e não podiam ser colocadas no ambiente de produção visto que ainda não se tinha um *frontend* funcional por causa do *upgrade*.
    - Cliente final insatisfeito.

<u>Caso 2</u>:

Identificou-se que no momento da Tomada de Decisão, contraiu-se uma Dívida Técnica que ficou categorizada da seguinte forma:

- **Decisão Tomada:** *Definição da forma de persistência de eventos de monitoração na base de dados.*
- **Influências dessa decisão:** Membro da equipe aponta como melhor solução no momento, pois detinha conhecimento de tal tecnologia.
- **Motivação:**
  - *Visível:* Necessidade de mudança da forma de persistência de eventos de monitoração.
    - O uso desta ferramenta atendia o único requisito solicitado no momento.
  - *Não Visível:* Era de conhecimento do profissional que sugeriu usá-la e pouco ou nenhuma busca foi realizada para encontrar outra alternativa.
- **Impactos:**
  - *Direto:* Tempo de entrega de novos componentes demoravam mais do que esperado, pois foi preciso se adaptar a tecnologia implantada.
  - *Indireto:* A falta de um conhecimento mais detalhado do negócio, como: onde se pretendia chegar? Pode levar a equipe a cometer erros no momento da escolha de determinada tecnologia, implicando assim na necessidade de

mudança da mesma em um futuro breve e/ou na limitação de possíveis soluções que podem ser oferecidas ao cliente.

Caso 3:

Identificou-se que no momento da Tomada de Decisão contraiu-se uma Dívida Técnica que ficou categorizada da seguinte forma:

- **Decisão Tomada:** Utilização do *MON* (https://mon.wiki.kernel.org/) como agente de monitoração na ferramenta.
- **Influências dessa decisão:** Cliente externo, coloca como restrição mandatória não utilizar outra alternativa cogitada, que no caso era *SNMP* (http://www.snmplink.org/).
- **Motivação:**
  - *Visível*
    - Restrição do time de segurança. Provável falta de conhecimento técnico dos mesmos por impor esta limitação.
  - *Não Visível*
    - Falta de questionamento e também provável conhecimento técnico para fazer valer tal alternativa por parte da equipe do projeto.
    - Falta de análise mais profunda levando em conta o que produtos da concorrência já estavam disponibilizando no mercado.
- **Impactos:**
  - *Direto:* Reescrita de código para que novas funcionalidades se adaptassem a solução existente, até chegar ao momento em que uma nova solução foi adquirida e adicionada a solução anterior.
  - *Indireto:* Muitas estimativas sobre dados de infraestrutura foram perdidos.

Caso 4:

Identificou-se que no momento da Tomada de Decisão, contraiu-se uma Dívida Técnica que ficou categorizada da seguinte forma:

- **Decisão Tomada:** *Ter o CMDB (Configuration Management Database) como mandatório na inserção de informações na ferramenta.*
- **Influências dessa decisão:** Mandatória da diretoria da empresa.
- **Motivação:** Dados espalhados em vários lugares e muitas vezes inconsistentes.
- **Impactos:**
  - *Direto:* Muitas funcionalidades ficaram atrasadas pois dependiam de dados advindos desse sistema e não estavam atualizadas.
  - *Indireto:* Às vezes o que é apontado como um *bug* do sistema, na verdade é apenas inconsistência nos dados advindos do CMDB e dispende-se certo tempo até se chegar a esta conclusão e/ou provar isso para o relator do *bug*.

5. Discussão dos resultados

Em todos os casos aqui estudados pode-se dizer que as Dívidas Técnicas contraídas foram de curto prazo e inseridas de forma Não Intencional, como classificam as pesquisas de [McConnel 2007; McConnel 2008 e Fowler 2009].

Em relação ao estudo de Kaiser (2011), observou-se que a aderência dos desenvolvedores pode ser uma barreira para se pagar as Dívidas Técnicas, bem como o volume de negócios, onde novos requisitos surgem a todo instante.

As pesquisas de McConnel (2008) destacam que o primeiro passo a ser dado é aumentar a visibilidade da Dívida Técnica. Nos *casos 1, 2 e 3* ficou claro que a Dívida Técnica foi contraída devido à falta de visibilidade da equipe sobre o real problema do negócio.

Mar (2010) ressalta que para saber se um sistema possui Dívida Técnica ou está em *'design death'*[1] é preciso prestar atenção em 3 indicadores: (1) se os componentes do sistema legado estão interligados de tal forma que não é possível isola-los; (2) quando não há nenhum indício de testes no código – sem testes unitários abrangentes é impossível refatorar o código para um estado mais gerenciável e (3) se o conhecimento do código do sistema está segmentado em apenas uma ou duas pessoas da empresa. Podemos observar que se olhássemos para o sistema analisando apenas esses três fatores, já identificaríamos que o mesmo possuía Dívida Técnica, pois, nos *Casos 1, 2 e 3* a decisão foi tomada devido ao poder de convencimento de um membro da equipe com maior conhecimento técnico em determinada tecnologia, no *caso 4* não se aplica essa classificação pois o mesmo foi oriundo de uma decisão externa mandatória. No *Caso 3* os componentes do sistema estão completamente dependentes um do outro.

Nos estudos de Guo (2009) e Guo (2011) são propostos mecanismos para o gerenciamento da Dívida Técnica e sugerem que Dívida Técnica deve ser considerada como um tipo particular de risco em manutenção de software. Não se aplicam a este estudo, pois, ficou claro que o conceito de Dívida Técnica era desconhecido pela equipe de desenvolvimento, logo a mesma não foi gerida, apenas inserida no sistema. O mesmo vale para a pesquisa de Izurieta (2012) que propõe um modelo embasado nas abordagens existentes que visa facilitar a organização, visualização e identificação de Dívida Técnica.

Em um estudo recente publicado por [Allman 2012] destaca que Dívida Técnica é inevitável, onde o objetivo não deve ser eliminá-la, mas sim administrá-la. E comunicação, compreensão e gestão da Dívida Técnica podem fazer uma enorme diferença a longo e curto prazo para o sucesso do projeto. O estudo realizado enfatiza tal importância, deixando claro que se a Dívida Técnica for inevitável é preciso ter ciência dessa contração para que o seu impacto possa ser administrado e utilizado em benefício do projeto.

Seaman (2012) propõe formas de como tomar decisões considerando Dívida Técnica, uma vez que a Dívida Técnica já tenha sido identificada e medida. Como supracitado, nos casos estudados as Dívidas Técnicas foram ignoradas, não sendo possível utilizar tais abordagens sugeridas por Seaman. Porém, cabe ressaltar que o alto impacto gerado pela contração da Dívida Técnica, enfatiza a necessidade de considerá-la no momento de tomada de decisão como sugere Seaman.

Os casos também foram classificados seguindo as propriedades de Dívida Técnica: decisão, benefícios, dívida, juros e pagamento, criadas por Cunningham (1992), como pode ser observado na **Figura 3**.

---

[1] Quando um código possui tantas Dívidas Técnicas que o esforço necessário para alterar qualquer parte dele é muito oneroso, dizemos que o mesmo está com 'd*esign morto*'. – Informação retirada do artigo "*Technical Debt and Design Death*", Kane Mar e Michael James.

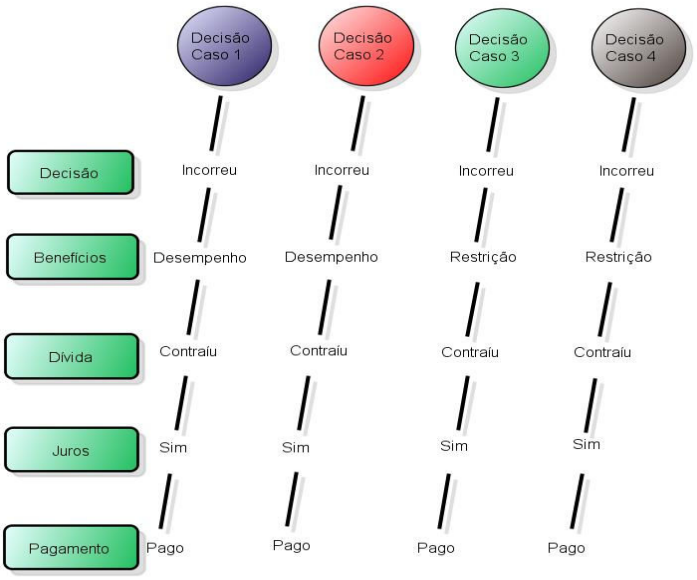


Figura 3 – Classificação dos casos segundo as propriedades criadas por Cunningham (1992).

Na **Figura 3** é possível identificar que as decisões tomadas incorreram em Dívida Técnica em prol de alguns benefícios e que as mesmas foram pagas ao longo do tempo a custos altos como: clientes insatisfeitos, concorrência oferecendo soluções mais completas no mercado; altos custos para fazer refatoração, negócio tendo que ser adaptado a tecnologia, pois a mesma não atendia às necessidades, equipe com alto grau de estresse e sem motivação para dar continuidade ao projeto.

Por fim, este estudo não só corrobora a importância de se considerar Dívida Técnica como deixa claro que Dívidas não geridas se transformam em grandes problemas que podem perdurar por toda a vida útil do sistema. E que quanto mais o tempo passa maior o custo que se paga por ignorá-las. Destaca também a importância de se haver comunicação clara entre membros de equipe e também entre equipe técnica e área de negócio. Pois, se a equipe não sabe aonde se pretende chegar com o negócio, determinada solução pode sim atender aquele requisito em dado momento, mas se tornar em uma grande Dívida no momento seguinte.

Acredita-se que foi dado um importante passo, identificando, monitorando e caracterizando as Dívidas Técnicas e a partir disso criando um modelo que de forma generalista mostra onde elas surgem, quais suas influências, motivações e para onde tem seguido. Assim hoje é possível passar a monitorá-las e ir pagando as mesmas ou frações das mesmas ao longo do tempo de forma estratégica ou até mesmo é possível se chegar ao entendimento de que é melhor não pagá-las. Bem como este exemplo pode ser replicado em outros projetos em andamento buscando outras Dívidas Técnicas que podem estar causando impactos negativos no negócio. Este mesmo conceito pode ser replicado para outros projetos que não somente de tecnologia. E as motivações e influências de inserção de Dívida aqui encontradas podem ser consideradas em novos projetos no momento de tomada de decisão para que os mesmos erros e consequentes impactos não se repitam.

6. Ameaças a Validade

Os dados da pesquisa foram coletados através de entrevistas, logo, para uma versão futura seria interessante realizar triangulação de dados buscando confrontar as evidências com

dados de documentos do projeto. Não foi possível fazer um cálculo detalhado sobre custo, pois, não foi possível ter acesso a variáveis como: valor homem/hora, custo da manutenção do sistema enquanto a nova versão não era disponibilizada e impacto/perdas para o negócio pelo atraso das entregas. Por isso, o gráfico de impacto, que pode ser visualizado na **Figura 2** possui apenas os dias de atraso da entrega do projeto.

7. Trabalhos Futuros

Objetiva-se dar continuidade a esta pesquisa, replicando o estudo para outros projetos da empresa, bem como buscar acesso a documentos para corroborar as evidências encontradas.

Em relação à Dívida Técnica, muitas ainda são as lacunas existentes em como identificar, gerir e utilizá-la de forma estratégica ao longo do projeto. Cabe ressaltar que até o momento as pesquisas realizadas vêm sendo embasadas na metáfora criada por Cunningham (1992), pois não existe uma teoria formada sobre Dívida Técnica.

8. Agradecimentos

Agradecemos em especial à empresa que nos propiciou acesso a seu ambiente, dados e colaboradores. E a todos os colaboradores que prontamente dispuseram seu tempo para contribuir com a pesquisa. Agradecemos também a Fapesp agência financiadora dos projetos número; 2011/23470-7 e 2009/16354-0.

Referências

Allman, E. (2012) “Managing Technical Debt”. Publications of the ACM, Volume 10, Issue 3, Março, 2012.

Bon, V.J., et.al (2000) “Introduction to ITIL”. The Stationery Office. U.K.ISBN 0113309732.

Brown, N. et al. (2010). Managing technical debt in software-reliant systems. The FSE/SDP workshop on Future of software engineering research. New Mexico: [s.n.]. p. 47-52.

Cunningham, W. (1992) “The WyCash Portfolio Management System”. Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92). Vancouver: [s.n.].

Fowler, M. (2009) “Technical Debt Quadrant.” Disponível em <http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html>. Acesso em: 26 maio. 2012, 10:37.

Gartner. (2012). “Gartner Estimates Global 'IT Debt' to Be $500 Billion This Year, with Potential to Grow to $1 Trillion by 2015.” http://www.gartner.com/it/page.jsp?id=1439513 . Último acesso, em abril/2012.

Gomes, R. (2011) “Caracterização e Conceituação Teórica da Metáfora de Débito Técnico Através de um Estudo Exploratório.” Dissertação de Mestrado Universidade Federal de Pernambuco, Setembro, 2011.

Gomes, R. Cavalcanti, A. Tonin, G. et.al. (2011) “An Extraction Method to Collect Data on Defects and Effort Evolution in a Constantly Modified System.” MTD '11 Proceedings of the 2nd Workshop on Managing Technical Debt. ACM, New York, USA, 2011.

Guo, Y. (2009) “Measuring and Monitoring Technical Debt.” 4th International Doctoral Symposium on Empirical Software Engineering. [S.l.]: [s.n.]. p. 25-46.

Guo, Y. and Seaman, C. (2011) “A portfolio approach to technical debt management.” Proceeding of the 2nd working on Managing technical debt. [S.l.]: [s.n.].

Guo, Y, Seaman, C. Gomes, R. Cavalcanti, A. Tonin, G. et.al (2012) “Tracking Technical Debt - An Exploratory Case Study”. Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM’ 11), 528-531, 2011.

Highsmith, J. (2011). “Accelerating Enterprise Agility Adaptive Leadership.” Agile Conference 2011.Salt Lake City, Utah, Agosto.

Izurieta, C. Vetro, A. Zazworka, N. Cai, Y. Seaman, C and Shull, F. (2012) “Organizing the Technical Debt Landscape.” Third International Workshop on Managing Technical Debt in conjunction with ICSE 2012.

Mar, K e James, M. (2010) “The Technical Debt and Design Death.” CollabNet.

McConnel, S. (2008) “Managing Technical Debt.” Construx Software, Version 1, 2008. Disponível em <http://www.construx.com/File.ashx?cid=2797>. Acesso em: 26 maio.2012, 10:28.

McConnel, S. (2007) “Technical Debt.” Construx Software,Version 1. Disponível em <http://www.construx.com/File.ashx?cid=2797>. Acesso em: 26 maio. 2012, 10:28.

Merriam, S. B. (2009) “Qualitative Research: A Guide to Design and Implementation.” 3. ed. [S.l.]: Jossey-Bass.

Robson, C. (2002) “Real World Research.” Blackwell, (2nd edition).

Seaman, C., et.al. (2012) “Using Technical Debt Data in Decision Making: Potential Decision Approaches.” MTD2012: 3rd Workshop on Managing Technical Debt, at ICSE 2012, Zurich, Switzerland.

Strauss, A. and Corbin, J. (2008) “Pesquisa qualitativa: Técnicas e procedimentos para o desenvolvimento de teoria fundamentada nos dados.” Tradução de Luciane de Oliveira da Rocha. 2. ed. Porto Alegre: Artmed.

Tonin, G.S (2011). “Decisões em Projetos de Desenvolvimento de Software e a Metáfora do Débito Técnico: Um Estudo De Caso Exploratório”. Dissertação de Mestrado Universidade Federal de Pernambuco, Setembro, 2011.

VersionOne. (2011). “State of Agile Development Survey Results” http://www.versionone.com/state_of_agile_development_survey/11/. Último acesso, abril/2012.

Williams, L. (2010) “Agile Software Development Methodologies and Practices”. Advances in Computers, vol. 80, pp. 1-44.Yin, R. K. (2003) “Case Study Research: Design and Methods.” 3. ed. [S.l.]: Sage.

Investigação experimental e Práticas ágeis:
ameaças à validade de experimentos envolvendo a prática ágil Programação em Par.

Vagner C. M. Lima[1], Adolfo G. S. Seca Neto[1], Maria Claudia F. P. Emer[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Departamento Acadêmico de Informática (DAINF)
Universidade Tecnológica Federal do Paraná (UTFPR) – Curitiba, PR – Brasil

vagnercml@hotmail.com, {adolfo,mclaudia}@dainf.ct.utfpr.edu.br

Abstract. *Even though the process of investigation by experimentation, in the context of Software Engineering, is not something new, it still lacks new quality studies. In studies that adopt the experiment as a method of research, a complete and detailed evaluation of the threats to the internal, external, construct and conclusion validity is an extremely important quality factor. Based on a detailed analysis of two empirical studies, this work presents a list of 21 possible threats to the validity of an experiment involving the Pair Programming agile practice, giving an exemplification of 17 possible threats.*

Resumo. *A investigação por experimentação no contexto da Engenharia de Software não é algo recente, mas ainda carece de novos estudos de qualidade. Em estudos que adotam o experimento como método de pesquisa, uma avaliação completa e detalhada das ameaças à validade interna, externa, de construção e de conclusão é um fator de qualidade importantíssimo. A partir de uma análise detalhada de dois estudos empíricos, este trabalho apresenta uma lista de 21 possíveis ameaças à validade de um experimento envolvendo a prática ágil Programação em Par, proporcionando a exemplificação de 17 possíveis ameaças.*

1. Introdução

O interesse da indústria de software em métodos ágeis para o desenvolvimento de sistemas, na última década, vem crescendo ano após ano (DYBÅ e DINGSØYR, 2008). Os estudos sugerem que a academia deve aumentar o número e a qualidade de estudos envolvendo tais métodos (DYBÅ e DINGSØYR, 2009). Esses métodos, diferente das abordagens tradicionais de desenvolvimento de software, valorizam mais as pessoas e interações que processos e ferramentas (AGILE MANIFESTO, 2011).

Buscando contribuir com este cenário, começamos no final do ano passado um estudo que busca investigar por meio de experimentos o desempenho da prática ágil Programação em Par perante diferentes tarefas de modelagem e construção de software. O desempenho será medido a partir da satisfação, motivação e produtividade dos participantes e também da qualidade do software por eles produzido. E, na época, buscávamos a eventual existência de uma forma diferenciada de avaliação de desempenho das duplas versus esforço individual.

A Programação em Par é uma prática de Engenharia de Software, conforme Beck e Andres (2004), primária do método ágil *Extreme Programming*, ou XP. De

acordo com VersionOne (2011) e Melo *et al.* (2012), a XP é um dos métodos ágeis mais adotados pela indústria de software.

A investigação por experimentação no contexto da Engenharia de Software não é algo recente, mas ainda carece de novos estudos de qualidade. O Experimento, o Estudo de caso e o S*urvey* são as três principais formas primárias de se conduzir uma investigação experimental. No contexto deste trabalho, destaca-se o primeiro. O Experimento caracteriza-se pela manipulação de algumas variáveis e a observação de outras em um ambiente controlado. Em geral, essa forma traz um maior controle sobre a execução, medição, investigação e facilidade de repetição. Porém, o custo e o risco são altos (MAFRA e TRAVASSOS, 2006; TRAVASSOS, 2011; TRAVASSOS, GUROV e AMARAL, 2002).

As etapas de uma investigação experimental, independente da forma adotada, são Definição, Planejamento, Avaliação, Execução, Análise e Interpretação e, por fim, realizada em paralelo, Empacotamento. Entretanto, Definição e Planejamento são referências para as etapas seguintes. É na etapa de planejamento de um experimento onde o pesquisador preocupa-se em como o estudo experimental será conduzido. Por exemplo, qual o contexto e o desenho do experimento (MAFRA e TRAVASSOS, 2006).

É também especificamente na etapa de planejamento da investigação experimental que o pesquisador deve analisar e definir as ameaças à validade do seu experimento segundo quatro tipos de validade: interna, externa, de construção e de conclusão. A eventual existência de uma determinada ameaça que não foi previamente definida e devidamente tratada pode comprometer os resultados do experimento, e com isso invalidar o estudo como um todo (MAFRA e TRAVASSOS, 2006).

Há obras que descrevem as ameaças mais comuns para os tipos de validade citados anteriormente (TRAVASSOS et. al., 2002; WAINER, 2007). É por meio dessas descrições que o pesquisador definirá as ameaças específicas à validade de seu experimento. Contudo, logo que chegamos à etapa de planejamento do estudo experimental descrito no início desta seção, principalmente no passo de análise e definição das ameaças, levantaram-se os seguintes questionamentos: as ameaças à validade de um experimento envolvendo a prática ágil Programação em Par limitam-se as ameaças apontadas como as mais comuns? E, se as descrições das ameaças mais comuns são genéricas e em geral sem exemplos práticos, como avaliar essa possível limitação?

Perante esse contexto, este artigo apresenta o resultado de uma identificação preliminar de exemplos práticos de ameaças à validade de um experimento envolvendo a prática ágil Programação em Par. Espera-se com o resultado deste trabalho auxiliar outros pesquisadores orientando seus esforços para avaliações mais abrangentes e detalhadas das ameaças à validade de seus experimentos. E, também, busca-se contribuir com a qualidade do estudo experimental que está em andamento.

Nas seções a seguir, apresentamos um resumo sobre Programação em Par, conceitos pertinentes à validade de um experimento, a análise e identificação de exemplos práticos de ameaças à validade de experimentos envolvendo PP, uma discussão dessas análises e, finalmente, as conclusões.

2. Programação em Par (PP)

A Programação em Par é uma prática que dois desenvolvedores trabalham lado a lado em um computador colaborando continuamente em um mesmo projeto, algoritmo, código ou teste (WILLIAMS e KESSLER, 2000). Nela, os dois desenvolvedores podem ser vistos como um único organismo pensante, o qual é responsável por todos os aspectos do artefato que está sendo elaborado. Um desenvolvedor, o piloto, controla o computador e escreve o código. O outro, chamado de navegador, contínua e ativamente observa o trabalho do piloto, procurando por defeitos, pensando em alternativas, buscando recursos e considerando implicações estratégicas. E, durante o processo, eles deliberadamente e periodicamente trocam de papéis (WILLIAMS *et al.*, 2000).

Alguns dos benefícios esperados com o uso PP no Desenvolvimento de Software são: (a) melhoria da qualidade do projeto e do código, o trabalho em equipe, a comunicação e a disseminação do conhecimento e, combinada com outras práticas, tais como Desenvolvimento Orientado a Teste (*Test Driven Development* - TDD), é uma prática chave para melhoria da qualidade do código (SFETSOS e STAMELOS, 2010); (b) aumento da produtividade, em geral, no médio e longo prazo, segundo Poppendieck e Poppendieck (2011), pois a qualidade e a robustez do código são melhoradas quando "dois conjuntos de olhos" continuamente inspecionam o código.

Embora os benefícios citados anteriormente sirvam de grande incentivo para o uso da PP, tanto na indústria de software quanto em outros contextos, há alguns desafios a serem enfrentados. Alguns destes podem inviabilizar a adoção da prática na indústria de software, são eles: o relacionamento humano, que é um dos grandes desafios da PP (TELES, 2004; BECK e ANDRES, 2004) ou a "guerra" de ferramentas de desenvolvimento e padrões de codificação e pares do tipo "professor-estudante" (SHORE e WARDEN, 2008).

3. Validade de um experimento

Em um experimento o pesquisador possui controle sobre a quantidade e o tipo de grupos de participantes, qual grupo fará o quê, em que momento haverá a intervenção e, em alguns casos, quem participará de cada grupo (WAINER, 2007). Os tipos de variáveis possíveis são: (a) variáveis independentes (ou fatores), as quais apresentam a causa que afeta o processo de experimentação e (b) variáveis dependentes, as quais se referem a saída de tal processo. O valor de (a) se chama 'tratamento' e o valor de (b) se chama 'resultado' (MAFRA e TRAVASSOS, 2006; TRAVASSOS *et al.*, 2002).

E mais, há vários desenhos experimentais – em linhas gerais, como os grupos de participantes serão arranjados (ver seção 3.5). Ressalta-se que a presença ou não de determinadas ameaças à validade do experimento dependerá do desenho experimental escolhido pelo pesquisador (WAINER, 2007).

A validade de um experimento está relacionada ao nível de confiança que se pode ter no processo de investigação experimental como um todo. Ou seja, o quão confiáveis são os elementos envolvidos nesse processo – desde a base teórica adotada até os resultados obtidos, inclusive a forma como são apresentados, considerando validade de construção, interna, de construção e externa (TRAVASSOS *et al.*, 2002; WAINER, 2007).

3.1 Validade de construção (VCS)

A validade de construção, para Travassos *et al.* (2002), "considera os relacionamentos entre a teoria e a observação, ou seja, se o tratamento reflete a causa bem e o resultado reflete o efeito bem". As ameaças mais comuns a este tipo de validade estão relacionadas com: **Projeto do experimento:** em geral, má definição da base teórica ou da definição do processo de experimentação; **Fatores humanos (ou sociais):** por exemplo, os participantes podem basear seus comportamentos nas hipóteses de pesquisa ou eles podem estar envolvidos em outros experimentos.

3.2 Validade interna (VI)

A validade interna, para Travassos *et al.* (2002), "define se o relacionamento observado entre o tratamento e o resultado é causal, e não é resultado da influência de outro fator – não controlado ou medido". Para Wainer (2007), as ameaças mais comuns a este tipo de validade são: **Instrumentação:** a diferença nos resultados é consequência de uma medição incorreta ou instrumento inadequado; **Testagem:** o desenho proporciona com que os participantes aprendam com seus próprios erros; está relacionada à sequência da coleta dos resultados versus o momento da intervenção, por exemplo; **Maturação:** os sujeitos podem tornar-se mais capazes ou ficarem desmotivados com o passar do tempo; **História:** possibilidade de que o resultado do processo é consequência de um evento externo ao experimento.

Continuando a lista de ameaças, **seleção:** os participantes não foram selecionados de maneira aleatória ou os grupos não foram divididos de maneira igualitária tanto no aspecto quantitativo quanto qualitativo; **Mortalidade seletiva:** a evasão de participantes com características específicas e relevantes durante o experimento; c**ontaminação:** por exemplo, participantes que fazem parte de um grupo ensinam, ou aprendem, com participantes do outro grupo; **Comportamento competitivo:** os membros do grupo de controle se sintam preteridos frente aos do outro grupo, e podem se mostrar motivados a competir com o grupo experimental; **Comportamento compensatório:** alguma pessoa, ou entidade, externa ou não ao experimento, sente que o grupo de controle foi preterido e, com isso, cria medidas compensatórias para ele.

E mais, **regressão à média**: fenômeno que pode trazer ameaça ao experimento. Em geral, conforme Bandeira (2012), essa ameaça está presente quando se trabalha apenas com um grupo (somente o grupo experimental) e seus integrantes são selecionados de maneira deliberada com base nos piores ou nos melhores escores do pré-teste, ou seja, não há um grupo de controle e seleção aleatória de participantes; **Efeito da expectativa do sujeito:** os participantes esperam ou buscam um determinado resultado – positivo ou negativo – para o experimento, ou ainda, pode haver um efeito positivo pelo simples fato dos participantes saberem que estão sendo observados – conhecido também por efeito *Hawthorne*.

Por fim, **efeito da expectativa do experimentador:** o pesquisador interage intensamente com os participantes, as crenças dele causam um efeito no sujeito ou nos testes realizados por ele, ou ainda, o simples fato do pesquisador querer que o experimento "dê certo"; **influência de parte da intervenção:** aparece em diferentes domínios com diferentes nomes, logo, este efeito não tem um nome padrão, mas a ideia

é que o efeito observado não é devido à intervenção como um todo, mas devido a apenas parte dela.

3.3 Validade de conclusão (VCC)

A validade de conclusão está, segundo Travassos *et al.* (2002), "relacionada à habilidade de chegar a uma conclusão correta a respeito dos relacionamentos entre o tratamento e o resultado do experimento", e isso envolve a correta **análise e interpretação estatística** do resultado (por exemplo, teste estatístico versus tamanho da amostra), **confiabilidade das medidas** e **confiabilidade da implementação dos tratamentos**.

3.4 Validade externa (VE)

A validade externa, conforme Travassos *et al.* (2002), "define condições que limitam a habilidade de generalizar os resultados de um experimento para a prática industrial", as ameaças mais comuns a este tipo de validade são: **Participantes:** selecionar participantes que não possuem relação ou refletem o comportamento da população, ou, trabalhar com uma amostra não representativa quantitativamente e qualitativamente da população; **Tempo:** por exemplo, impor ou suprimir restrições de tempo; **Configuração do experimento:** realizar o experimento em um ambiente muito diferente do ideal, ou adotar matérias ou instrumentos distantes da realidade.

3.5 Ameaças à validade versus Desenho experimental

Dentre os desenhos experimentais, existe um desenho que é considerado verdadeiramente experimental, pois a seleção dos participantes entre o grupo experimental e de controle é realizada de maneira aleatória. Dessa forma, algumas ameaças são neutralizadas ou minimizadas (Tabela 1), algo que não acontece em desenhos como os citados no final do parágrafo seguinte (WAINER, 2007). Destaca-se que os estudos selecionados e avaliados na próxima seção adotaram esse desenho.

O desenho comentado anteriormente é o seguinte: dois-grupos, pós-teste e seleção aleatória (veja sua representação a seguir). Entretanto, quando a seleção não é aleatória, mas há grupo de controle o desenho é chamado de *quase experimento*, e quando não há grupo de controle o desenho é chamado de pré-experimental – ou não-experimental em outras obras (WAINER, 2007).

Grupo 1: *A* ***X*** *O*
Grupo 2: *A* *O*

O Grupo 1 é o que sofre a intervenção, ele é o grupo experimental. O Grupo 2 é o que não sobre a intervenção, ou seja, o grupo de controle. Os sujeitos são selecionados e divididos (*A*) entre os grupos de forma aleatória e a observação (*O*) é realizada somente após o teste, ou intervenção (*X*). Segundo Wainer (2007), a análise estatística apropriada seria algo como a comparação entre o resultado do pós-teste do Grupo 1 versus Grupo 2.

Tabela 1 – Situação de cada ameaça à validade versus experimento controlado

Validade/Ameaças	Situação da ameaça versus experimento controlado	
	Permanece	**Neutraliza/Minimiza**
Validade interna (VI)		
Instrumentação	X	
Testagem		X
Maturação		X
História		X
Seleção		X
Mortalidade seletiva		X
Contaminação	X	
Comportamento competitivo	X	
Comportamento compensatório		X
Regressão à média		X
Efeito da expectativa do sujeito	X	
Efeito da expectativa do experimentador	X	
Influência de parte da intervenção	X	
Validade externa (VE)		
Participantes	X	
Tempo	X	
Configuração do experimento	X	
Validade de construção (VCS)		
Projeto do experimento	X	
Fatores humanos (ou sociais)	X	
Validade de conclusão (VCC)		
Análise e interpretação estatística	X	
Confiabilidade das medidas	X	
Confiabilidade da implementação dos tratamentos	X	

4. Experimentos envolvendo PP

4.1 Seleção dos estudos para análise

A prática Programação em Par vem sendo investigada em inúmeros estudos como, por exemplo (separados conforme o principal método de pesquisa): **(a) experimento**: Balijepally *et al.* (2009), Choi *et al.* (2008), Hannay *et al.* (2010), Walle e Hannay (2009) e na academia Salleh *et al.*, (2009), Salleh *et al.* (2011b), Sfetsos *et al.* (2008) e Williams *et al.* (2000); **(b) revisão sistemática da literatura:** Sfetsos e Stamelos (2010) e na academia Salleh *et al.* (2011a); **(c) estudo de caso:** Melo *et al.* (2011), Williams e Kessler (2000) e na academia Fila e Loui (2010) e Han *et al.* (2010).

Dentre os estudos citados no parágrafo anterior, foram selecionados para este trabalho os estudos que atenderam os seguintes critérios ou diretrizes: (a) o método de pesquisa adotado pelo estudo foi a investigação experimental, especificamente, o experimento; (b) o estudo deveria ter seções e subseções específicas para as ameaças à validade; ou, caso não tivesse este detalhamento, o estudo deveria apresentar formas diferenciadas (ou mais rigorosas) de avaliar o desempenho das duplas versus esforço individual.

Além disso, procuramos selecionar no mínimo um estudo feito com participantes da indústria e outro com participantes da Academia. Os estudos que atenderam os critérios descritos anteriormente e que foram selecionados para a análise são os

seguintes: **(a) Sfetsos *et al.* (2008)** [Estudo A]: nesse estudo há uma seção e várias subseções que descrevem a avaliação da validade da investigação experimental e **(b) Balijepally *et al.* (2009)** [Estudo B]: nesse estudo há uma modesta seção que descreve a avaliação da validade da investigação experimental, entretanto, foi o único estudo que apresentou uma forma diferenciada ou mais rigorosa de avaliar o desempenho das duplas versus esforço individual.

4.2 Método de identificação e classificação dos exemplos de ameaças à validade

Para identificar e classificar os exemplos de ameaças à validade de um experimento envolvendo PP, um dos autores deste trabalho leu os estudos selecionados e gerou uma tabela intermediária onde as ameaças relatadas nos estudos foram analisadas, resumidas e classificadas originalmente segundo os quatro tipos de validade definidos anteriormente.

Na elaboração da tabela intermediária foi considerado o seguinte: para o Estudo A foi mantido o nome que os pesquisadores definiram para cada ameaça. Para o Estudo B ver "Correlação por Definição (CD)" descrita a seguir. Ressalta-se que cada ameaça foi identificada e classificada mesmo se o autor do estudo justificou a ausência dela no experimento ou se ele definiu um mecanismo de proteção para ela.

Na sequência, a tabela intermediária junto com restante do trabalho foi enviada aos outros dois autores. Eles revisaram o artefato e, em seguida, cada um enviou suas considerações para o relator. Com as considerações em mãos buscando um maior nível de detalhe e compreensão o relator elaborou uma nova tabela [re]classificando novamente as ameaças da seguinte forma**: (a) Correlação Total (ou CT):** caso o tipo de validade, a identificação (nome) e a definição da(s) ameaça(s) relatada pelo estudo em análise coincidiram com a literatura sobre ameaças à validade utilizada neste trabalho, ou somente literatura de referência; **(b) Correlação Parcial (ou CP):** se somente o tipo de validade e a definição da(s) ameaça(s) relatada pelo estudo em análise coincidiram com a literatura de referência.

Ou ainda, **(c) Tipo de validade não correspondente - reclassificadas (RC):** caso o tipo de validade da(s) ameaça(s) relatada pelo estudo em análise não coincidiu com a literatura. Nessa situação a ameaça foi [re]classificada a partir da sua descrição segundo a literatura de referência; **(d) Correlação por Definição (CD):** não havia descrição explicita da ameaça no estudo em questão. Logo, foi necessário definir as ameaças a partir do conteúdo das seções "Avaliação da Validade" ou "Limitações" do estudo em questão conforme as possibilidades descritas na literatura de referência.

Continuando o processo, essa nova tabela foi enviada novamente aos outros dois autores. Eles revisaram o artefato e, em seguida, cada um enviou suas considerações para o relator. A partir dessas considerações o relator atualizou a nova tabela e, na sequência, dividiu essa nova tabela em uma tabela para cada tipo de validade[1]. Por fim, para facilitar a busca pela descrição ou exemplo da ameaça, gerou-se uma tabela consolidada a partir da Tabela 1, mas com o conteúdo das quatro tabelas elaboradas durante todo o processo de análise (ver Tabela 2).

[1] Estão disponíveis em: https://docs.google.com/open?id=0B-lo3hrzjKFXWWRqZ204NlB1LVk ou http://vagnercml.files.wordpress.com/2012/08/wbma2012_lima_secaneto_emer_tabelas_intermediarias_v1-0.pdf

4.3 Identificação e classificação dos exemplos de ameaças à validade

Na tabela a seguir pode-se identificar em qual estudo há a descrição ou exemplificação de uma determinada ameaça, e pode-se identificar também em qual nível de correlação com a literatura de referência.

Tabela 2 – Consolidação: resultado da análise dos estudos experimentais

Validade/Ameaças	**Estudo onde a ameaça está descrita/exemplificada versus nível correlação com literatura**			
	Correlação total (CT)	**Correlação parcial (CP)**	**Correlação por definição (CD)**	**Tipo de validade não correspondente - reclassificadas (RC)**
Validade interna (VI)				
Instrumentação	Estudo A		Estudo B	
Testagem				
Maturação	Estudo A			
História	Estudo A			Estudo A (2*): originalmente uma ameaça a VE e outra a VCC.
Seleção	Estudo A		Estudo B	
Mortalidade seletiva		Estudo A	Estudo B	
Contaminação		Estudo A		
Comportamento competitivo		Estudo A		
Comportamento compensatório		Estudo A		
Regressão à média				
Efeito da expectativa do sujeito				
Efeito da expectativa do experimentador				Estudo A: originalmente uma ameaça a VCS.
Influência de parte da intervenção				
Validade externa (VE)				
Participantes		Estudo A	Estudo B	
Tempo			Estudo B	Estudo A: originalmente uma ameaça a VI.
Configuração do experimento		Estudo A	Estudo B	
Validade de construção (VCS)				
Projeto do experimento		Estudo A(7*)		
Fatores humanos (ou sociais)		Estudo A(2*)	Estudo B	
Validade de conclusão (VCC)				
Análise e interpretação estatística	Estudo A(4*)			
Confiabilidade das medidas	Estudo A			
Confiabilidade da implementação dos tratamentos	Estudo A			

* Quantidade de diferentes descrições ou exemplos por ameaça.

5. Discussão

Ao analisarmos os resultados obtidos e presentes na tabela da seção anterior, vemos que foi possível identificar exemplos práticos para mais de 80% (17 das 21) das ameaças mais comuns relatadas por Wainer (2007) e Travassos *et al.* (2002).

Uma ameaça à validade interna é, segundo Wainer (2007), "uma outra possível explicação para o efeito observado que não a ação ou a manipulação". Então, caso exista essa outra possível explicação, há uma ameaça e a validade do estudo pode ser questionada. Sendo assim, independente do tipo de validade, ou o pesquisador cita a ameaça e justifica o porquê de ela não ocorrer no seu estudo, ou ele cita a ameaça, assumindo sua presença, e descreve algum mecanismo de proteção que minimiza ou neutraliza seu efeito. Caso contrário, a validade do estudo poderá ser questionada.

Por exemplo, como ficariam as ameaças Testagem, Regressão à média, Efeito da expectativa do sujeito e Influência de parte da intervenção não citadas em ambos os estudos? A primeira e a segunda ameaça o estudo não precisaria necessariamente citar, pois, conforme Wainer (2007), elas são neutralizadas pelo desenho experimental escolhido. Porém, os autores do estudo poderiam ser questionados sobre as outras duas.

O Estudo A apresentou uma quantidade alta de ameaças correlacionadas parcialmente em Projeto do experimento e Análise e interpretação estatística, fato que nos chamou a atenção e nos levou a analisar brevemente alguns dos referenciais teóricos utilizados pelo estudo. Constatou-se, então, que tais referenciais merecem uma investigação mais minuciosa, pois se demonstraram um pouco mais abrangentes do que o referencial teórico adotado neste trabalho.

O Estudo B adotou uma forma diferenciada de avaliação de desempenho das duplas versus esforço individual. Em resumo, nela trabalha-se com o desempenho dos pares reais versus pares nominais. Um par nominal é resultado da combinação artificial (ou aleatória) de dois participantes que desenvolveram o software individualmente. Após essa combinação os membros de cada par nominal são classificados em melhor e segundo melhor integrante da dupla, considerando, por exemplo, a variável qualidade de software. Então, avaliação do desempenho é realizada comparando o desempenho dos pares reais versus os melhores membros das duplas artificiais, ou os segundos melhores membros das duplas artificiais, ou ambos.

Algumas possíveis causas de ameaças à validade de experimentos envolvendo PP:

1. Dificuldade em lembrar a sintaxe da linguagem de programação (pode-se disponibilizar uma documentação online da respectiva linguagem) [Estudo B, Validade interna, Instrumentação];
2. Diferença de desempenho e de produtividade entre desenvolvedores [Estudo A, Validade interna, Seleção];
3. Os participantes não estarem familiarizados com a prática [Estudo B, Validade externa, Participantes];
4. Não há um período de aquecimento para as duplas [Estudo B, Validade externa, Tempo];
5. Desmotivação por parte dos participantes em consequência do uso de uma documentação online da linguagem de programação [Estudo B, Validade de construção, Fatores humanos (ou sociais)];

6. Não implementar a prática de maneira correta [Estudo A, Validade de conclusão, Confiabilidade da implementação dos tratamentos];
7. Vários aspectos relacionados à análise estatística dos dados [Estudo A, Validade de conclusão, Análise e interpretação estatística].

Por fim, o pesquisador deve ficar atento a situações onde um mecanismo de proteção de uma determinada ameaça pode causar uma nova ameaça ou intensificar outra, por exemplo, situações 1 e 5 apresentadas anteriormente. E mais, o eventual questionamento por parte de alguns pesquisadores sobre a não validade do experimento quando são usados estudantes para representar profissionais de software, de acordo com Höst *et al.* (2000), em certas condições, os estudantes podem ser usados sem necessariamente causar ameaça a validade do experimento.

6. Conclusões

A realização de um experimento de qualidade envolvendo a prática ágil Programação em Par é algo custoso e arriscado. Pois, dentre vários elementos a serem definidos e planejados, vimos por meio deste trabalho que a quantidade de ameaças à validade de um experimento desse tipo é de pelo menos 21. Entretanto, considerando o nível de detalhe do Estudo A, a quantidade não se limitaria a esse número. Uma vez que apenas as ameaças relacionadas a Projeto do Experimento nesse estudo são 7 e não apenas uma. E, dependendo a situação, caso o pesquisador desconsidere alguma dessas ameaças o resultado do experimento se torna sem importância ou inválido, havendo a necessidade de realizar um novo experimento com algumas alterações ou até de recomeçar todo o estudo experimental.

Por outro lado, independente da prática ágil investigada, para diminuir o risco associado às ameaças à validade de um experimento o pesquisador deve: (a) fazer a escolha correta do desenho experimental e (b) realizar análise e descrição das ameaças à validade a partir de um "catálogo de ameaças" mais abrangente e detalhado, inclusive especificando para cada ameaça o seu mecanismo de proteção e eventuais relacionamentos de causa desse mecanismo com outras ameaças.

Há uma forma diferenciada (ou mais rigorosa) de avaliação de desempenho das duplas versus esforço individual. Ela foi utilizada no Estudo B.

Como trabalho futuro sugere-se complementar essa lista de ameaças a validade de experimentos envolvendo PP, buscando em outros estudos as ameaças não identificadas neste trabalho. Outro possível trabalho futuro é, usando o método apresentado aqui, realizar o mesmo estudo, mas abordando outra prática ágil que possa vir a ser objeto de investigações experimentais, por exemplo, Desenvolvimento Orientado a Testes.

Referências

Agile Manifesto. (2001), http://agilemanifesto.org.

Bandeira, M. (2012), "Validade interna e externa de uma pesquisa: vieses", http://www.ufsj.edu.br/portal2-repositorio/File/lapsam/Texto%204-VALIDADE.pdf.

Balijepally, V., Mahapatra, R., Nerur, S. e Price, K. H. (2009) "Are two heads better than one for software development? the productivity paradox of pair programming". In: *MIS Quarterly*, v. 33, n.1 (Março), p. 91-118.

Beck, K. e Andres, C. (2004), Extreme Programming Explained: Embrace chance, Boston: Addison-Wesley, 189 p.

Choi, K. S., Deek F. P. e Im I. (2008), "Exploring the underlying aspects of pair programming: The impact of personality". In: *Information and Software Technology*, v. 50, n. 11, p. 1114-1126.

Dybå, T e Dingsøyr, T. (2008) "Empirical Studies Of Agile Software Development: A systematic review". In: *Information and Software Technology*, v. 50, n. 9, p. 833-859.

Dybå, T e Dingsøyr, T. (2009), "What Do We Know about Agile Software Development ?". In: *IEEE Software*, Volume 26, n. 5, p. 6-9.

Fila, N. D. e Loui, M. C. (2010), "Work-in-progress - Who's driving? Structured pairs in an introductory electronics laboratory". In: *IEEE*, p. F3C-1 – F3C-2.

Han, K.-W., Lee, E. e Lee, Y. (2010), "The Impact of a Peer-Learning Agent Based on Pair Programming in a Programming Course". In: *IEEE Transactions on Education*, v. 53, n. 2, p. 318-327.

Hannay, J. E., Arisholm, E., Engvik, H. e Sjøberg, D. I. K. (2010), "Effects of Personality on Pair Programming". In: *IEEE Transactions on Software Engineering*, v. 36, n. 1, p. 61-80.

Höst, M., Regnell, B. e Wohlin, C. (2000), "Using students as subjects - a comparative study of students and professionals in lead-time impact assessment". In: *Empirical Software Engineering*, v. 5, n. 3, p. 201-214.

Mafra, S. N. e Travassos, G. H. (2006), "Estudos Primários e Secundários apoiando a busca por Evidência em Engenharia de Software". COPPE/UFRJ, Rio de Janeiro, Relatório técnico: RT-ES 687/06.

Melo, C., Cruzes, D. S, Kon, F. e Conradi, R. (2011), "Agile Team Perceptions of Productivity Factors". In: *AGILE 2011 Conference*. IEEE, p. 57-66.

Melo, C., Santos, V. A., Corbucci, H., Katayama, E., Goldman, A. e Kon, F. (2012) "Métodos ágeis no Brasil: estado da prática em times e organizações". Departamento de Ciência da Computação do IME/USP, São Paulo, Relatório técnico: RT-MAC-2012-03, http://www.agilcoop.org.br/files/MetodosAgeisBrasil2011.pdf.

Poppendieck, M. e Poppendieck, T. (2011), Implementando o Desenvolvimento Lean de Software: Do Conceito ao Dinheiro, Porto Alegre: Bookman, 280 p.

Salleh, N., Mendes, E., Grundy, J. e Burch, G. (2009), "An empirical study of the effects of personality in pair programming using the five-factor model". In: *3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, p. 214-225.

Salleh, N., Mendes, E. e Grundy, J. (2011a), "Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review". In: *IEEE Transactions on Software Engineering*, v. 37, n. 4, p. 509-525.

Salleh, N., Mendes, E. e Grundy, J. (2011b), "The effects of openness to experience on pair programming in a higher education contexto". In: *24th IEEE-CS Conference on Software Engineering Education and Training*. IEEE, p. 149-158.

Sfetsos, P. e Stamelos, I. (2010), "Empirical Studies on Quality in Agile Practices: A Systematic Literature Review". In: *Seventh International Conference on the Quality of Information and Communications Technology*. IEEE, p. 44-53.

Sfetsos, P., Stamelos, I., Angelis, L. e Deligiannis, I. (2008), "An experimental investigation of personality types impact on pair effectiveness in pair programming". In: *Empirical Software Engineering*, v. 14, p. 187-226.

Shore, J. e Warden, S. (2008), A Arte do Desenvolvimento Ágil, Rio de Janeiro: Alta Books, 420 p.

Teles, V. M. (2004), Extreme Programming: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade, São Paulo: Novatec, 316 p.

Travassos, G. H. (2011), "Experimentação em Engenharia de Software: Fundamentos e Conceitos". In: *X Simpósio Brasileiro de Qualidade de Software*. Curitiba, Minicurso.

Travassos, G. H., Gurov, D. e Amaral, E. A. G. (2002), "Introdução à Engenharia de Software Experimental". COPPE/UFRJ, Rio de Janeiro, Relatório técnico: RT-ES-590/02, http://www.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf.

VersionOne (2011) "State of Agile Development", http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf.

Wainer, J. (2007) "Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação",http://bibliotecadigital.sbc.org.br/?module=Public\&action=PublicationObject\&subject=228\&publicationobjectid=70.

Walle, T. e Hannay, J. E. (2009), "Personality and the nature of collaboration in pair programming". In: *3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, p. 203-213.

Williams, L. A. e Kessler, R. R. (2000) "All I really need to know about pair programming I learned in Kindergarten". In: *Communications of the ACM*, v. 43, n. 5, p. 108-114.

Williams, L., Kessler, R. R., Cunningham, W. e Jeffries, R. (2000) "Strengthening the case for pair programming". In: *IEEE Software*, v. 17, n. 4, p. 19-25.

Desafios de Requisitos em Métodos Ágeis: Uma Revisão Sistemática

Aline Jaqueira[1], Elisa Andreotti[2], Márcia Lucena[1], Eduardo Aranha[1]

[1] Departamento de Informática e Matemática Aplicada
UFRN Natal –RN
[2] Centro de Pós-Graduação e Extensão
FUCAPI Manaus - AM
aline_o@hotmail.com, elisa.andreotti@yahoo.com.br,
{marciaj,eduardoaranha}@dimap.ufrn.br

Abstract. *Agile methods are inserted in dynamic software development environments where requirements are constantly changing and are built from the feedback of stakeholders during the development. Although many changes occur in requirements, little emphasis is given in requirement specification in agile methods. In this context, this paper presents a systematic review of requirements in agile methods in order to present a survey about who is discussing requirements in agile methods, what are the main challenges and what are the proposals to solve them.*

Resumo: *Métodos ágeis estão inseridos em ambientes de desenvolvimento de software dinâmicos onde os requisitos estão em constante modificação e são construídos a partir do feedback dos stakeholders no decorrer do desenvolvimento. Apesar de ocorrer muitas alterações nos requisitos pouca ênfase é dada à atividade de especificação de requisitos no contexto de métodos ágeis. Neste contexto, este trabalho apresenta uma revisão sistemática sobre requisitos em métodos ágeis a fim de apresentar um levantamento de quem está discutindo o assunto sobre requisitos em métodos ágeis, quais os principais desafios apontados e as propostas para solucioná-los.*

1. Introdução

Os métodos ágeis têm ganhado cada vez mais espaço no desenvolvimento de software e interesse entre profissionais e pesquisadores [Cao; Ramesh 2008]. A rápida transformação do ambiente de negócios é um desafio ao desenvolvimento de software, devido às exigências para o software evoluírem tão rapidamente que, em alguns casos, os requisitos se tornam obsoletos antes mesmo do término do projeto. Assim, os métodos ágeis surgiram com o objetivo de minimizar os principais desafios no contexto atual de desenvolvimento.

A engenharia de requisitos, com seu enfoque sistemático, destina-se a elicitar, organizar e documentar os requisitos de um software, com base em um processo que estabeleça e mantenha um acordo entre os *stakeholders* [Engholm 2010]. Pode ser definida como um processo de comunicação onde será especificado o que o software deve fazer, suas funções, suas propriedades essenciais e desejáveis, e também suas restrições [Sommerville 2007].

Apesar da importância da engenharia de requisitos no sucesso do desenvolvimento do software e na minimização dos riscos de projeto, essa atividade é vista nos métodos ágeis como atividade burocrática que torna o processo menos ágil, pois nos métodos ágeis é dada uma maior ênfase na codificação. A principal justificativa ocorre devido ao fato de que os requisitos mudam tão rapidamente que um documento de requisitos fica desatualizado tão logo seja redigido, desperdiçando todo esforço empreendido na documentação. Nesse contexto, a engenharia de requisitos e os métodos ágeis podem ser considerados atividades incompatíveis segundo alguns autores como [Paetsch; Eberlein; Maurer 2003]. Portanto, para melhor entender como os requisitos são tratados nos métodos ágeis este trabalho busca fazer uma revisão sistemática dos principais trabalhos acadêmicos, conferências, workshops que discutem o assunto apresentando os principais desafios encontrados e as propostas existentes para solucioná-los.

Este trabalho está organizado da seguinte forma: a seção 2 apresenta os trabalhos relacionados. A seção 3 descreve a metodologia utilizada para a busca de trabalhos e para a extração de resultados, bem como as questões de pesquisa. A seção 4 apresenta a análise dos resultados encontrados em resposta às questões. A seção 5 apresenta uma discussão sobre os resultados. Por fim, a seção 6 apresenta as conclusões deste trabalho.

2. Trabalhos Relacionados

Inicialmente, a fim de encontrar revisões sistemáticas relacionadas com métodos ágeis, foi realizada uma busca na literatura através do site para pesquisa livre de ciência da computação Free Search (http://dblp.isearch-it-solutions.net/dblp/). Foi encontrado um trabalho de 2008 [Dyba and Dingsøyr 2008] que avalia, sintetiza e apresenta os resultados empíricos do desenvolvimento ágil de software. No Google Acadêmico foi encontrada uma revisão sistemática de 2009 [Hossain, Babar and Paik 2009] cujo objetivo é apresentar os desafios no uso do Scrum para desenvolvimento global de software bem como as estratégias para lidar com eles. Também foi encontrado um estudo empírico de [Cao; Ramesh, L. and Ramesh, B. 2008] que apresenta práticas de requisitos ágeis utilizadas no contexto de projetos reais mostrando seus benefícios e desafios.

O presente estudo apresenta uma perspectiva acadêmica sobre o assunto, destacando os principais desafios de requisitos em métodos ágeis e as principais propostas da literatura que tentam tratar esses desafios. Assim, este trabalho mostra-se relevante para os pesquisadores e profissionais da área no sentido de apoiá-los nas pesquisas e na adoção das metodologias ágeis.

3. Método de Pesquisa

De acordo com [Kitchenham; Charters 2007], a revisão sistemática é uma forma de estudo secundário que usa uma metodologia bem definida para identificar, analisar e interpretar as evidências disponíveis relacionadas a uma questão de pesquisa específica de forma imparcial. Para a elaboração deste trabalho foi adotado o guia de [Kitchenham; Charters 2007] considerado uma referência na área de engenharia de software experimental. A pesquisa foi realizada na literatura publicada em conferências e jornais da área de Ciências da Computação que citaram o assunto ‘requisitos e métodos ágeis’. O período anual delimitado foi de 2008 a 2012 de modo a destacar os resultados mais recentes.

3.1 Questões de pesquisa

Para realizar o levantamento a que se propõe este trabalho, foram elaboradas as seguintes questões de pesquisa:

Q1 – Quais as conferências e workshops publicam sobre métodos ágeis e que podem trazer discussões sobre requisitos ágeis?

Q2 – Existem desafios apontados para requisitos em métodos ágeis? Quais?

Q3 – O que está sendo proposto para tratar os desafios de requisitos em métodos ágeis?

Q3.1 – Existem adaptações e/ou extensões das práticas ágeis para tratar os desafios de requisitos? Quais?

Q3.2 – Existem adaptações e/ou extensões de práticas de métodos tradicionais que são usadas em métodos ágeis? Quais?

A questão Q3 procura por propostas que tratam os desafios endereçados pela questão Q2. As propostas são divididas de acordo com as questões Q3.1 e Q3.2 com o objetivo de fazer um levantamento tanto de propostas provenientes de métodos ágeis como também propostas de adaptações de métodos tradicionais que tratam os desafios de requisitos em métodos ágeis.

3.2 Processo de Busca

As buscas foram realizadas sobre o título e o resumo dos artigos publicados entre 2008 e 2012 combinando buscas automáticas e manuais. As buscas automáticas foram realizadas em *ACM Digital Library, IEEEXplore Digital Library, Science Direct, ISI Web of Science e Scopus*. A *string* utilizada na busca automática foi ("Requirements Engineering" AND "Agile methods") OR ("Agile requirements").

As buscas manuais foram realizadas para as conferências *Agile Conference, Agile Brazil, International Requirements Engineering Conference (RE), Empirical Software Engineering and Measurement (ESEM) e Evaluation and Assessment in Software Engineering (EASE)* e no *Workshop on Requirements Engineering (WER)*. Essas buscas foram feitas em paralelo com as buscas automáticas. Os resultados das buscas automáticas e manuais foram unidos e as duplicatas removidas. A lista final de estudos potencialmente relevantes foi a entrada para a atividade de seleção do estudo.

3.3 Critérios de Inclusão e Exclusão

A inclusão foi dada pela relevância dos trabalhos em relação às questões investigadas. Os critérios usados foram: (i) o artigo menciona explicitamente requisitos em métodos ágeis; (ii) o artigo menciona problemas com requisitos em métodos ágeis; (iii) o artigo menciona contribuições (extensões ou adaptações em metodologias) para tratar requisitos em métodos ágeis; (iv) o estudo usado no artigo é empírico com métodos ágeis. O critério de exclusão adotado foi: (i) estudos incompletos como resumos ou apresentação de slides.

3.4 Seleção do Estudo

O procedimento adotado para seleção desta pesquisa foi dividido em duas etapas. Na primeira etapa, Aline Jaqueira (AJ) e Elisa Andreotti (EA) analisaram o conjunto de estudos obtidos através das buscas automáticas e manuais considerando os critérios de inclusão e exclusão, além de observar os títulos, resumos e palavras-chave. Havendo alguma dúvida ou conflito foi realizada a leitura plena do artigo e discutida sua

relevância pelas pesquisadoras para gerar um consenso. Ao final um conjunto de artigos para a revisão foi selecionado.

Na segunda etapa foram obtidos os artigos completos do conjunto de estudos potencialmente relevantes da primeira etapa. Esse conjunto de artigos foi dividido de modo que cada parte fosse avaliada por uma pesquisadora. Nove artigos foram selecionados e analisados na íntegra: [Cao; Ramesh 2008**],** [Gallardo-Valencia; Sim 2009], [Vlaanderen *et al.* 2009], [Hoda; Noble; Marshall 2010]**,** [Savolainen; Kuusela; Vilavaara 2010], [Sen; Hemachandran 2010], [Mordinyi; Kühn; Schatten 2010], **[**Bjarnason; Wnuk; Regnell 2011], [Espinoza; Garbajosa 2011].

4. Análise dos Resultados

Esta seção apresenta a síntese dos resultados deste trabalho de acordo com as questões de pesquisa organizada em ordem cronológica crescente.

4.1 Questão de pesquisa 1

Q1 – Quais as conferências e workshops publicam sobre métodos ágeis e que podem trazer discussões sobre requisitos ágeis?

A *International Requirements Engineering Conference* é o principal fórum internacional para pesquisadores, educadores, profissionais e estudantes apresentarem e discutirem as mais recentes inovações, tendências e experiências no campo da Engenharia de Requisitos. A *Agile Conference* foi criada por uma equipe de especialistas e profissionais qualificados envolvidos com métodos ágeis para apresentar propostas no contexto de práticas ágeis. Esse fórum é dedicado a descobrir maneiras melhores de desenvolver software inspiradas nos valores e princípios do manifesto ágil. A *International Conference XP* reúne trabalhos relacionados aos métodos ágeis, além de apresentar abordagens para requisitos ágeis.

O primeiro workshop de engenharia de requisitos ágeis, *Agile Requirements Engineering Workshop* (AREW), aconteceu em 2011, em Lancaster, Inglaterra dentro da conferência ECOOP (European Conference on Object-Oriented Programming). O objetivo do workshop foi fazer um balanço para descobrir as necessidades da engenharia de requisitos ágil e se suas práticas fornecem o que a metodologia necessita.

A tabela 1 mostra uma síntese dos principais eventos que tratam de requisitos em métodos ágeis.

Tabela 1: Síntese dos eventos de requisitos em métodos ágeis

Evento	Periodicidade	WebSite do Evento
Workshop on Agile Requirements Engineering (AREW)	Ocorreu em 2011	www.comp.lancs.ac.uk/~gacitur1/AREW11/
International Requirements Engineering Conference	Anual	www.requirements-engineering.org/
Agile Development Conference	Anual	www.agiledevelopmentconference.org
XP / Agile Universe	Anual	http://www.informatik.uni-trier.de/~ley/db/conf/xpu/index.html

4.2 Questão de Pesquisa 2

Q2 – Existem desafios apontados para requisitos em métodos ágeis? Quais?

Ao analisar os trabalhos levantados nesta revisão sistemática podemos destacar os desafios mais citados relacionados a requisitos em métodos ágeis: (i) a disponibilidade do cliente para fornecer *feedback* constante, (ii) a gestão da mudança dos requisitos e da arquitetura do software, (iii) bem como da rastreabilidade já que as mudanças são frequentes, (iv) limitação do cliente que as vezes não possui autonomia ou capacidade para fornecer *feedback* apropriado do projeto como um todo, (v) requisitos não funcionais negligenciados, (vi) consenso na negociação ou priorização dos requisitos, principalmente quando há mais de um cliente, (vii) documentação que é mínima e sua falta pode causar diversos problemas, (viii) equipe de desenvolvimento multifuncional, onde o profissional tem vários perfis, (ix) custo e estimativa do projeto, já que o mesmo tem seu escopo flexível e sofre alterações a todo momento.

Os principais desafios apontados foram encontrados nos artigos selecionados em que os estudos são empíricos baseados na experiência da utilização ou migração para métodos ágeis. Isso contribui para que a avaliação deste trabalho tenha um maior respaldo, já que os estudos apontam os principais desafios dos métodos ágeis de acordo com a prática.

Em [Cao; Ramesh 2008] os desafios de requisitos em métodos ágeis foram apresentados utilizando o método Grounded Theory [Strauss; Corbin 1990]. Aplicando um estudo de caso, foram avaliadas qualitativamente as respostas coletadas em 16 empresas caracterizadas como desenvolvedoras ágeis dos Estados Unidos. Assim, as principais práticas ágeis utilizadas nessas empresas foram apresentadas com seus respectivos desafios:

(i) Comunicação face a face no lugar da especificação escrita: como desafio essa prática apresenta uma dependência de interação entre clientes e desenvolvedores e, consequentemente, da disponibilidade e aptidão do cliente. Quando isso não é possível, há riscos de requisitos desenvolvidos de forma inadequada ou incorreta. A eficácia da comunicação entre o cliente e a equipe depende de vários fatores, incluindo disponibilidade do cliente, consenso entre os grupos de clientes e confiança especialmente durante o início do projeto. A dificuldade de ter o cliente disponível todo o tempo também foi relatada. Outro desafio é a negociação para consenso dos requisitos quando o projeto tem mais de um cliente envolvido.

(ii) Engenharia de requisitos iterativa: para essa prática, o principal desafio é o custo e a estimativa de cronograma já que o escopo do projeto está sujeito a constantes mudanças. Dessa forma, obter apoio da gestão para tais projetos pode ser difícil. Outro desafio é a documentação mínima, pois quando uma falha de comunicação ocorre devido, por exemplo, à rotatividade de pessoal, mudanças rápidas nos requisitos ou indisponibilidade do cliente, a falta de documentação pode causar uma variedade de problemas. Finalmente, um outro desafio é a negligência na descrição dos requisitos não-funcionais, que podem ser mal definidos ou ignorados, pois o cliente muitas vezes está preocupado com funcionalidades essenciais.

(iii) Priorização de requisitos: ao utilizar o valor do negócio como o único critério para priorização de requisitos corre-se o risco de negligenciar requisitos não funcionais, como, por exemplo, não atender requisitos de segurança e eficiência. Além disso, alguns participantes observaram que a redefinição de prioridades contínua, quando não praticada com cautela, leva à instabilidade do software.

(iv) Gestão da mudança de requisitos através de um planejamento constante: algumas arquiteturas de desenvolvimento que a equipe escolhe durante os ciclos iniciais podem tornar-se inadequadas após alteração dos requisitos. O redesenho da arquitetura contribui significativamente para o custo do projeto.

(v) Prototipagem: muitas empresas reconhecem que os riscos de implantação de protótipos em modo de produção podem causar problemas com aspectos de qualidade, tais como escalabilidade, segurança e robustez, ou seja, mais uma vez os requisitos não funcionais podem ser comprometidos. Além disso, a implantação rápida de protótipos nos estágios iniciais cria expectativas irreais aos clientes, que rejeitam ciclos mais longos de desenvolvimento, e que muitas vezes são necessários para desenvolver software mais robusto.

(vi) Desenvolvimento dirigido a testes (TDD): os desenvolvedores criam testes como parte de um requisito antes de escrever novo código especificando o comportamento do código. Um grande desafio para a adoção TDD é que requer equipe multifuncional e os desenvolvedores não estão acostumados a escrever os testes antes da codificação, e a prática exige muita disciplina, compreensão completa dos requisitos e colaboração entre cliente e desenvolvedor.

(vii) Reuniões de revisão e testes de aceitação: a validação dos requisitos não aborda aspectos da verificação formal porque não há modelagem formal de requisitos detalhados. Quanto aos testes de aceitação para verificação dos requisitos, as empresas relatam a dificuldade do cliente escrever esses testes, ou seja, os clientes têm limitações e assim muitas dessas empresas tem um profissional destinado a ajudá-los a criar os testes.

Um outro trabalho [Gallardo-Valencia; Sim 2009] apresenta um estudo no campo de validação de requisitos utilizando práticas ágeis, provendo a validação contínua a todos os *stakeholders*. A análise do trabalho aborda também o desafio da falta de documentação em projetos ágeis, uma vez que as atividades estão centradas em torno da unidade básica da história do usuário, que pode ser em uma notificação por escrito, casos de testes e conversas. As validações ágeis de requisitos, no estudo de campo realizado, foram feitas antes de começar as interações, durante as reuniões de planejamento, abrangendo o projeto como um todo, valorizando as práticas ágeis na validação de requisitos.

A pesquisa desenvolvida por [Hoda; Noble; Marshall 2010] utiliza Grounded Theory [Strauss; Corbin 1990] e apresenta um estudo sobre equipes de desenvolvimento ágil. Nesse trabalho é enfatizado como a falta de envolvimento do cliente causa problemas na coleta e especificação de requisitos contribuindo para a perda de produtividade. São apresentadas as causas e conseqüências da falta de envolvimento do cliente em projetos ágeis. Foram entrevistados 30 profissionais de 16 empresas ágeis de desenvolvimento de software da Nova Zelândia e Índia. A maioria dos participantes afirmou que a falta de envolvimento dos clientes é vista como "a parte mais difícil do desenvolvimento ágil" e "o maior problema", pois o cliente comprometido e envolvido é vital para que o método alcance seu objetivo de ser ágil. Algumas causas levantadas do não envolvimento do cliente são destacadas [Hoda; Noble; Marshall 2010]: (i) ceticismo dos clientes, (ii) distância, (iii) falta de tempo do cliente, (iv) clientes de grandes empresas, (v) cliente ineficiente. E como consequências dessa falta de interesse temos: dificuldade na elicitação e esclarecimento dos requisitos, dificuldades na priorização dos requisitos e confiança do *feedback*, perda de produtividade e prejuízo na empresa.

Em [Savolainen; Kuusela; Vilavaara 2010] são apresentadas as lições aprendidas na transição de dois projetos ágeis e os desafios enfrentados como: o tamanho variado dos requisitos de usuário, o papel dos requisitos no sistema e os requisitos de arquitetura. Também é ressaltado que um dos principais desafios na utilização de métodos ágeis, quando ocorre um elevado número de equipes ágeis, está em dividir o trabalho entre as equipes, alcançar as propriedades de todo o software, e garantir as características simultâneas. Os autores destacam também que, ao ignorar o processo de descrições e especificações dos requisitos detalhados, preenchendo apenas o *backlog* (lista simples de todos os tipos de requisitos), torna-se mais trabalhoso, por exemplo, distinguir o que é requisito de sistema ou de usuário. Destacam ainda que é importante preservar as práticas chaves dos métodos ágeis, e da mesma maneira, é preciso investir na educação de engenheiros de requisitos quanto à importância da elicitação e análise de requisitos em projetos ágeis, removendo práticas desnecessárias e adicionando novas práticas, que auxiliem no sucesso do projeto.

No trabalho de **[**Bjarnason; Wnuk; Regnell 2011] um estudo de caso com nove profissionais de uma grande empresa de desenvolvimento de software é apresentado, destacando os seguintes desafios da engenharia de requisitos ágil: (i) fluxo contínuo de escopo: o sistema não está completo até o final do ciclo de vida, com o risco de não serem descobertas questões de nível de sistema até o final do projeto, (ii) equipes multifuncionais de desenvolvimento, (iii) dificuldade de ter o cliente disponível, (iv) dificuldade em manter o documento de requisitos atualizado, (v) garantir a competência suficiente do cliente e desenvolvedores, incluindo inovadoras ideias na interação entre eles, (vi) equilíbrio entre a agilidade e a estabilidade tanto a nível de projeto como para a equipe de desenvolvimento.

O trabalho de [Espinoza; Garbajosa 2011] aborda a deficiência da rastreabilidade de requisitos quando utilizados em métodos ágeis, devido à ausência de uma documentação adequada. De acordo com os autores, a diferença entre o desenvolvimento de software no modelo ágil e tradicional não se encontra apenas na forma como os processos são realizados, mas também nos artefatos produzidos como saídas de cada processo. Apesar disso, a rastreabilidade deve ser considerada em metodologias ágeis como ponto chave para se desenvolver sistemas de qualidade no intervalo de tempo determinado.

4.3 Questão de Pesquisa 3

Q3 – O que está sendo proposto para tratar os desafios de requisitos em métodos ágeis?

O trabalho de [Mordinyi; Kühn; Schatten 2010] está relacionado com uma proposta para tratar os desafios relativos às questões de projeto e arquitetura no desenvolvimento ágil. É proposto um *framework* de arquitetura chamado *Architecture Framework for Agile Business Requirements* (AFA) que distingui os modelos computacional, organizacional, de coordenação, de distribuição e de comunicação oferecendo flexibilidade sobre as alterações no projeto e arquitetura de acordo com as mudanças nos requisitos. Ao distinguir os modelos, os engenheiros têm mais facilidade para identificar apenas os componentes que são afetados quando surgem novos requisitos. Com isso, os efeitos da implementação desses novos requisitos em outros modelos podem ser minimizados. Uma vantagem dessa estrutura é que um engenheiro com conhecimento limitado opera com conceitos simples em cada categoria.

Já no trabalho de [Sen; Hemachandran 2010] são identificadas questões que levam os *stakeholders* às mudanças de ideias e alterações na fase de análise de requisitos. Ou seja, o trabalho apresenta uma proposta para tratar com gestão de mudança de requisitos. Durante a negociação deve-se chegar a um acordo através de uma linguagem compreendida por todos os envolvidos. Para resolver os problemas de volatilidade dos requisitos, devido à dependência da intuição dos *stakeholders* ou alguns desacordos, foi proposta a técnica *Agile Technique for Agent Based Goal Elicitation* (ATABGE), com base nos mecanismos e princípios da abordagem ágil. A técnica ATABGE assume que o analista trabalha a partir de diagramas existentes ou documentos disponíveis na forma de objetivos da empresa, missão e visão da empresa e política corporativa. Centra-se na identificação inicial e abstração dos objetivos dos *stakeholders* que podem ser as fontes disponíveis de metas, independentemente do âmbito de sua base de conhecimento. Essa abordagem também ajuda na elaboração de metas para representar o sistema desejado. Essa técnica foi aplicada para elicitação de metas da *Assam University Examination Branch* envolvendo cinco *stakeholders*.

O meta-modelo *Traceability metamodel for Methodology definition* (TmM), proposto por [Espinoza; Garbajosa 2011] trata a rastreabilidade dos requisitos em métodos ágeis como tipos de links, papéis e regras de vinculação, que são definidos pela propriedade do usuário, fornecendo apoio ao processo de desenvolvimento e permitindo a definição de um projeto específico que admite aos métodos ágeis utilizar a rastreabilidade, sendo possível alcançar um maior grau de automação em relação ao seu uso.

Q3.1 – Existem adaptações e/ou extensões das metodologias ágeis para tratar requisitos? Quais?

O trabalho apresentado em [Vlaanderen *et al*. 2009] enfatiza a inserção de um método de gerenciamento de produtos de software para a aplicação dos princípios do *framework* Scrum [Schwaber 1997]. No trabalho é apresentado um refinamento de requisitos ágeis baseado no Software Product Management (SPM) a fim de melhorar a capacidade de lidar com grandes escalas de requisitos em um ambiente ágil, dando suporte aos gerentes. Assim este trabalho está relacionado com o desafio de gerenciamento de requisitos. Podemos destacar os seguintes pontos desse trabalho:

(i) Como lições aprendidas, destaca-se a importância dos *sprints* alternados, ou seja, com o uso de SPM metade do desenvolvimento do *sprint* é finalizado garantindo que o *Product Backlog* estará pronto para ser usado quando as equipes de desenvolvimento iniciarem um novo *sprint*.

(ii) Requisitos considerados grandes precisam ser detalhados e estruturados, dividindo-os por temas, conceitos e tipo. O refinamento desses requisitos na estrutura da abordagem ágil, efetivou o gerenciamento de grandes conjuntos de requisitos em granularidade diferente.

(iii) É necessário administrar o *Backlog* com disciplina, para desempenhar com controle o SPM e manter o processo de um *sprint* com qualidade de tempo gasto.

(iv) A comunicação entre a equipe durante o processo de especificação de requisitos é essencial para promover a reutilização e integração dos requisitos, uma vez que, discutir os requisitos antes de serem implementados, auxilia na correção de erros e ambiguidades.

Para tratar com desafios relacionados com problemas com clientes foram usadas as seguintes estratégias de acordo com [Hoda; Noble; Marshall 2010]:

(i) Prioridade alterada: os requisitos que estavam aguardando maior esclarecimento ou priorização pelos clientes, na falta dos mesmos, tiveram a prioridade rebaixada ou sua implementação foi adiada até a resposta do cliente.

(ii) Avaliação de risco: para medir o nível de envolvimento do cliente no projeto, alguns participantes utilizaram um questionário de avaliação de risco. Tal iniciativa permitiu à equipe descobrir se o nível indicado de envolvimento do cliente é um risco potencial para o projeto. Notou-se que a falta de financiamento e indisponibilidade são as principais causas. A partir da avaliação foi possível negociar melhor com o cliente para convencê-lo da importância do seu envolvimento no projeto.

(iii) Proprietários de histórias: a prática dos proprietários de histórias foi uma adaptação da prática Scrum de alocar um proprietário ao produto [Schwaber 1997]. Proprietários de histórias foram responsáveis por histórias particulares, em vez de todas as histórias no *backlog* do produto, e somente as histórias com um proprietário entravam na priorização. Assim, não era preciso ter somente uma pessoa da organização do cliente para estar continuamente disponível. Isso permitiu à equipe planejar as histórias de acordo com a disponibilidade do proprietário.

(iv) Cliente Proxy: algumas equipes destacaram um membro do desenvolvimento para estar na coordenação junto aos clientes e assegurar deles requisitos e *feedback*. Essa prática foi empregada principalmente onde os clientes estavam fisicamente distantes.

(v) Uso de demos: apesar da relutância ou incapacidade para participar de reuniões, quase todos os clientes foram interessados o suficiente para assistir a demonstrações (demos), que muitas vezes eram o único meio de colaboração regular que as equipes ágeis recebiam dos clientes, usando essa oportunidade para discutir características e receber *feedback*.

(vi) E-colaboração: colaboração eletrônica foi um popular meio de comunicação regular com os clientes usando conferência por vídeo/voz, telefone, email e *chat*. Foi importante porque várias equipes tinham clientes fisicamente distantes e foi convenientemente barato.

(vii) Ágil disfarçado: para os clientes com ceticismo e oposição ao desenvolvimento ágil, algumas equipes escolheram seguir práticas ágeis internamente. Fazendo assim um esforço para evitar consequências da falta de envolvimento do cliente e perda de negócios.

Q3.2 – Existem adaptações e/ou extensões das metodologias tradicionais para serem usadas em métodos ágeis? Quais?

De acordo com os filtros utilizados neste trabalho, não foram encontradas adaptações ou extensões de metodologias tradicionais para tratar desafios dos requisitos nos métodos ágeis.

5. Discussão

A tabela 2 apresenta o resumo das respostas para as questões de pesquisa 2 e 3 identificando os principais desafios levantados neste trabalho, o número de vezes em que foram evidenciados nos artigos analisados e as propostas de tratamento para eles,

quando encontradas no estudo, bem como suas devidas referências. Cada desafio foi associado a uma fase da engenharia de requisitos de modo a considerar os desafios em cada fase específica.

Tabela 2. Desafios e Propostas para requisitos em métodos ágeis

Atividades da ER	Desafios	Referência do Desafio	Número de vezes que aparece nos artigos	Proposta de Tratamento	Referência da Proposta
Elicitação/ Validação	Disponibilidade do Cliente	[Cao; Ramesh 2008]; [Hoda; Noble; Marshall 2010]; [Bjarnason; Wnuk; Regnell 2011]	03	-	
	Limitação do cliente on site	[Cao; Ramesh 2008] ;[Hoda; Noble; Marshall 2010]; [Bjarnason; Wnuk; Regnell 2011]	03	-	
	Equipe multifuncional	[Cao; Ramesh 2008]; [Savolainen; Kuusela; Vilavaara 2010]; [Bjarnason; Wnuk; Regnell 2011]	03	-	
	Negligência de requisitos não funcionais	[Cao; Ramesh 2008]	01	-	
Gerenciamento	Gestão da mudança dos requisitos e da arquitetura do software	[Cao; Ramesh 2008]; [Vlaanderen *et al*. 2009] ; [Mordinyi; Kühn; [Sen; Hemachandran 2010]; Schatten 2010]; [Savolainen; Kuusela; Vilavaara 2010]; [Bjarnason; Wnuk; Regnell 2011]	06	SPM	[Vlaanderen *et al*. 2009]
				AFA	[Mordinyi; Kühn; Schatten 2010]
				ATABGE	[Sen; Hemachandran 2010]
	Documentação	[Cao; Ramesh 2008]; [Gallardo-Valencia; Sim 2009]; [Savolainen; Kuusela; Vilavaara 2010] ; [Espinoza; Garbajosa 2011] ;[Bjarnason; Wnuk; Regnell 2011]	05	-	
	Rastreabilidade	[Espinoza; Garbajosa 2011]	01	TmM	[Espinoza; Garbajosa 2011]
Viabilidade	Consenso na negociação	[Cao; Ramesh 2008]; [Sen; Hemachandran 2010]	02	-	
	Custo e estimativa	[Cao; Ramesh 2008]	01	-	

Durante a execução deste estudo notou-se que existe maior quantidade de trabalhos destacando os benefícios dos métodos ágeis, o que pode ser um indicativo da carência de trabalhos explorando os desafios existentes para as metodologias ágeis assim como as soluções para os mesmos. A partir dos resultados apresentados é possível identificar os principais desafios da engenharia de requisitos ágil e algumas soluções propostas. Vale destacar que a maioria dos resultados é fruto de dados empíricos o que destaca a relevância do presente estudo.

Para os nove principais desafios apontados, somente dois têm pelo menos uma proposta de tratamento, sendo que foram levantadas três propostas diferentes para tratar o desafio de gerenciamento da mudança dos requisitos e arquitetura do software. Os questionamentos que surgem então: (i) a gestão da mudança dos requisitos nas metodologias ágeis tem maior urgência de serem tratadas? Por quê? (ii) Por que os

demais desafios mencionados em mais de um trabalho não tem propostas de tratamento?

Sobre a técnica *Agile Technique for Agent Based Goal Elicitation* (ATABGE) [Sen; Hemachandran 2010], que tem base nos mecanismos e princípios da abordagem ágil, os resultados foram demonstrados com dados obtidos no processo e na aplicação da técnica. No entanto não foi relatado se a mesma atendeu ou não as necessidades da engenharia de requisitos efetivamente.

No S*oftware Product Management* (SPM) proposto por [Vlaanderen *et al*. 2009] o estudo de caso mostrou que, para garantir a SPM ágil e eficaz, vários fatores devem ser levados em conta tais como: tamanho da tarefa, estrutura e o atraso. A principal contribuição do trabalho é a descrição de um processo SPM baseado em princípios ágeis, expondo os diagramas de execução do processo tanto do desenvolvimento de software quanto do SPM, permitindo a reutilização do método descrito nas empresas, que possuem nível comparável de desenvolvimento.

6. Conclusões

Este trabalho apresentou uma revisão sistemática com o objetivo de pesquisar quais são os desafios da engenharia de requisitos nos métodos ágeis, as soluções propostas e as principais fontes de discussão sobre o assunto. O intervalo delimitado foi de 2008 a 2012, e um dos critérios de inclusão foram estudos empíricos de modo a obter respostas de trabalhos práticos aplicados no mercado de trabalho. Nove artigos relevantes foram selecionados como estudo primário gerando os resultados apresentados no trabalho. No entanto, apesar dos métodos ágeis estão sendo cada vez mais utilizados e que a engenharia de requisitos é uma das atividades mais desafiadoras e propensas a erros em um processo de desenvolvimento de software, foi observado que existem poucas pesquisas nesta área.

Com este estudo tem-se uma visão geral atualizada do assunto requisitos e métodos ágeis. É possível visualizar suas principais fontes de discussão, observar os principais desafios, as propostas de tratamento existentes, bem como as lacunas para propostas de tratamento em alguns desafios, conforme apresentado na tabela 2. Como trabalhos futuros pretende-se realizar uma busca manual exclusiva na *International Conference XP* que reúne trabalhos relacionados aos métodos ágeis, mas que, por motivo de falta de espaço, não foi contemplada neste trabalho. Também pretende-se averiguar se as propostas encontradas estão atendendo as lacunas da engenharia de requisitos ágil e propor soluções para os desafios.

Referências

Bjarnason, E., Wnuk, K. and Regnell, B. (2011) A Case Study on Benefits and Side-Effects of Agile Practices in Large-Scale Requirements Engineering. In Proceedings of the 1st Workshop on Agile Requirements Engineering

Cao, L. and Ramesh, B. (2008) "Agile Requirements Engineering Practices: AnEmpirical Study." IEEE Software. 60-67.

Dybå, T. and Dingsøyr, T. (2008) “Empirical Studies of Agile Software Development: A Systematic Review,” IST (50:9-10), pp. 833-859.

Engholm Júnior, H.(2010) Engenharia de Software na Prática, São Paulo: Novatec.

Espinoza, A. and Garbajosa, J. (2011) "Study to Support Agile Methods More Effectively through Traceability," Computer Science Innovations in Systems and Software Engi-neering, Vol. 7, No. 1, 2011, pp. 53-69.

Gallardo-Valencia, R.E. and Sim, S.E. (2009) Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile. In: proceedings of the Second International Workshop on Managing Requirements Knowledge. IEEE Computer.

Hoda, R., Noble, J. and Marshall, S. (2010) Agile Undercover: When Customers Don't Collaborate In: XP2010, Trondheim.

Hossain, E., Babar, M. A. and Paik, H. (2009) Using Scrum in Global Software Development: A Systematic Literatur Review. In: Proc. of the 4th International Conference on Global Software Engineering. IEEE Press, Los Alamitos.

Kitchenham, B. and Charters, S. (2007), Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University.

Mordinyi, R., Kühn, E. and Schatten, A. (2010) Structuring Complexity Issues for Efficient Realization of Agile Business Requirements in Distributed Environments. in: Proc. Agile Processes in Software Engineering and Extreme Programming, Springer Berlin Heidelberg, 48. ISBN: 978-3-642-13054-0; S. 202 - 207.

Paetsch, F., Eberlein, A. and Maurer, F. (2003) Requirements Engineering and Agile Software Development, In: 12th IEEE International WETICE 03, IEEE CS Press. pp. 308–313.

Savolainen, J., Kuusela, J., and Vilavaara, A. (2010) Transition to Agile Development - Rediscovery of Important Requirements Engineering Practices. Paper presented at the 18th IEEE international requirements engineering conference (RE), 2010. IEEE Computer Society (PP. 89-294)

Schwaber, K. (1997) Scrum Development Process, in OOPSLA Business Object Design and Implementation Workshop, J. Sutherland, D. Patel, C. Casanave, J. Miller, and G. Hollowell, Eds. London: Springer.

Sen, A.M. and Hemachandran, K. (2010) Elicitation of Goals in Requirements Engineering using Agile Methods. In: REFS 2010. 19-23 July 2010. Seoul Korea.

Sommerville, I. (2007) Engenharia de Software, 8ª edição, Tradução: Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edilson de Andrade Barbosa; São Paulo: Pearson Addison-Wesley.

Strauss A. and Corbin J. (1990) Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, Sage Publications, 1990.

Vlaanderen, K., Jansen, S., Brinkkemper, S., Jaspers, E. (2009) The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management. In: 3rd International Workshop on Software Product Management.

Incremental Tests: An Approach to Improve Software Tests in Agile Teams

Gabriela de Oliveira Patuci, Regina Lucia de Oliveira Moraes

School of Technology – University of Campinas (UNICAMP)
Limeira – SP – Brazil

opgabi@gmail.com, regina@ft.unicamp.br

Abstract. *Most agile and testing practices primarily make use of user stories for validation. This paper presents how to enhance user stories that normally only register system requirements to also be useful for test cases. This test plan should be as complete as possible so that it could be used in the sprint functional tests. Testers should have an essential role all throughout the process, starting from reviewing the requirements in conjunction with the Product Owner and building the backlog in a test case format. Incremental tests could be done in pairs and combining new functional testing, each day towards the sprint ending. Regression tests complement previous test whenever integration is done.*

1. Introduction

The agile software development methodologies have emerged as alternatives to the traditional methodologies, where the need of large documentation and compliance with standards, requires a longer development cycle. This lengthened time that is necessary to complete all steps of traditional methods, conflicts with the demand to reduce time-to-marketing in today's markets.

Even with the evolution of computers, techniques and tools in recent years, the production of reliable software, correct and delivered within the stipulated time and cost is still a very rare event [Soares, 2004]. For this reason, agile methodologies have been used by several Information Technology (IT) companies that are migrating from their traditional development models to models that allow lighter planning and management, such as Scrum.

Agile methodologies are being used widely, but often, as they seek for reduced cost and time, has caused many companies to despise or get rid of key stages, that later will be configured as low-quality in the developed product. There has been a wide acceptance of Scrum and the growth of the demand for experienced professionals in the field, as can be seen from the number of ScrumAlliance members (Agile Official and Global Community) that represents over 100,000 registered members and officially certified [ScrumAlliance, 2012].

Another way to measure an increase in the use of Scrum in recent years is through the appearance of a large number of conferences, official events and trainings in general. Data taken from the official website [ScrumAlliance, 2012] report the creation of more than 726 events related to Scrum in only 6 months (the last three months of 2011 and the first three of 2012).

Software testing is an activity that is part of the development as a whole and that is performed regardless of the methodology adopted. In Scrum, testing activities are not always completely defined and, by the fact that this activity is not the focus of most researches and case studies in agile environment, a better definition has been left open.

Articles and texts that refer to the agile test reality were researched during the last months. Much of the existing material in the area refers to the work of the same researchers, Crispin and Gregory (2009). Even searching on sites and publications of the agile community, there is still a great difficulty on finding works that focus on the testing activity in agile context, whether in articles or books. Researches in progress that are related to agile development methodologies still reflect the concern of the authors in refining the methodology and resolve gaps in the basic coding process and still didn´t have time enough to get in the validation steps, that are also a very important part of the product development itself.

The main objective of this paper is to show the results of a study created to cover this gap: adapting user stories to be also used as test cases. Also the daily activities are detailed in a way that this expertise can be a differential contribution to the team as a whole. Another objective is to identify how these activities will be carried out thus maintaining quality results and total adaptation to the needs of the agile teams.

The organization of the paper is as follows: after the first section that presents an introduction and motivation, Section 2 presents backgrounds about user stories, acceptance test and acceptance test driven development; the proposal and discussion is presented in Section 3 and Section 4 presents our conclusions.

2. Background

Important concepts such as User Stories, Acceptance Tests and Acceptance Test Driven Development are essential for a better understanding of this paper and its general scope. These will be presented in next subsections.

2.1. User Stories

According to Cohn (2004), “A user story describes functionality that will be valuable to either a user or purchaser of a system or software. Stories are composed of three aspects:

- A written description of the story to be used in planning and also as a reminder;
- Discussion about the story that will serve to raise all the details of the story;
- Tests that convey and document the details and can be used to determine when the story is complete.”

The third factor cited by Cohn is the basis for this proposal since it already makes clear the importance of testing as the development basis and to assist in user stories detailing.

More than just knowing what the user stories are about, what is expected of professionals who will work with that, especially the Product Owners and Test Analysts, is to write good stories. The most efficient way of doing this, according to Cohn (2004) and Crispin and Gregory (2009) would be to work with acceptance tests, which would be considered as the acceptance criteria of this user story.

2.2. Acceptance Tests

Acceptance testing can be considered as any validation that was planned to be taken into consideration to define if a user story is ready or not. The concept of acceptance testing becomes clear when Cohn (2004) considers that "Acceptance Testing is the process of verifying that the stories were developed so each of them works exactly the way the customer expects".

The acceptance test, taking into account the processes of traditional tests, is the last test step to be exercised before its operational use. Normally, the system is tested with data provided by the client and not with the simulated test data. Acceptance testing is an important tool to reveal gaps in the system requirement definition, but also to reveal problems of requirements that make the system does not meet user needs or provide an unacceptable performance [Sommerville, 2010].

According to Atlee and Pfleeger (2009), three types of acceptance test can be used to evaluate the system. The benchmark test evaluates the system based on specified standards. The test cases represent the typical conditions under which the system will operate when installed and are used to evaluate the actual performance for each of the planned test cases. The pilot test allows the user to install the system and enables users to make use of the system as if it was actually installed in the operating environment. It is based on the daily work of the system and often the client prepares a list which incorporates features of the day to day for each user. And last, the parallel test can be used when a new system is replacing an existing one. In this case, the new system operates in parallel to the previous version, and gradually, users will get used to the new system. The gradual transition allows the comparison between the new and the old system and also allows more skeptical users to gain confidence using the new system.

After acceptance testing, the client reports to the development team which requirements were not achieved which should be deleted, revised or included, due to the need for modifications. These changes should be analyzed and the consequences in the design, implementation and testing should be recorded.

2.3. Acceptance Test Driven Development (ATDD)

Acceptance Test Driven Development (ATDD) is increasing its use in software development team in general. The technique consists on using the steps presented in Figure 1 to assure the team build very good user stories and, more than that, a very good final product. The steps to follow are [Hendrickson, 2008]:

- Discuss the requirements;
- Distill tests in a framework-friendly format;
- Develop the code (and hook up the tests);

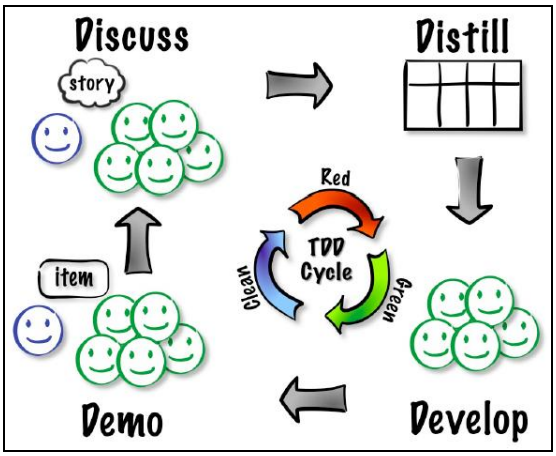


Figure 1. ATDD cycle model [Hendrickson, 2008]

The first step, to Discuss, can be described by the idea of having the right questions on the planning meeting and from them, to have also a better understanding of what has to be done. Once we have this understanding, the acceptance tests can be written in collaboration with the PO. Hendrickson (2008) says that "By asking the right questions we are prompting the business stakeholder to think carefully about his expectations for the feature and gleaning insight into the acceptance criteria for the feature".

After discussing and writing the User Stories, the next step would be to Distill. The idea is that having the tests cases sketched out, the team would be able to capture the tests in an automation framework format such as FIT, Fitnesse and Concordian. The next two steps, team has to Code and Demo the product, so the cycle would be completed.

The two first steps are the main idea of the ATDD model and also, the ones this proposal was most based.

3. The Proposal

When creating the requirements in a new format, it is possible to have a boost on the project quality assurance improving the day by day activities that should be executed, so the tests can be done in the most productive way. The most important concept here is to take advantage of all the sprint days and not having any professional "blocked" waiting for some user story to be ready for functional testing phases. The details of this will be presented in the next two subsections.

3.1. Test Backlog

User Stories construction and how they are applied nowadays were already described on the previous section (refer to Section 2.1). The proposal of this work is to present a new format and also a new function for User Stories. It is a fact that the User Stories have to be well constructed to be the base to a really good software development but since Scrum methodology claims less work on writing documentation in general, to write down good user stories and also to create detailed test plans seems not to be a so agile approach and a rework, since both have to be the expected behavior of the system very

well detailed. Figure 2 presents a User Story as it is used currently in most of agile teams.

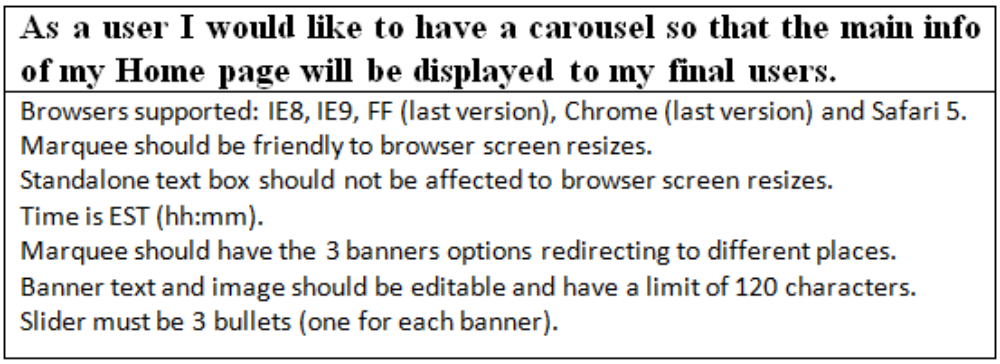


Figure 2. A User Story in the current agile team's format

One role of the tester role is to follow up with the Product Owner on construction of the user stories to help ensure that they best fit into the team necessities, It should be noted that by both working together provided a good match to build really good testable user stories that would benefit both the project requirement and also the test plan.

Another advantage of this approach is the approval of the test plan by the client and its use to a really constructive acceptance tests in every build delivery and validation that would match the value generation expectation. Figure 3 presents the same User Story presented in Figure 2 but now complemented with test cases. The User Story new format can guide tester to validate the system, developers to best understand the requirement and user to make easier to perform acceptance tester Stories construction.

As it can be visualized on Figure 2, the way user stories are written in some of current agile projects is to have all the details that the Product Owner wants the system to have on the User Story cards. It is nice to have these details on the card, but they have to be rewritten in test case format after the planning meeting and, when this time comes, a lot of questions and possible blocks can arise.

The format proposed on Figure 3 uses the idea that is being already started in some projects, but in the simplest way possible to document it. It shows that all the requirements have to be testable so they can be used for both development bases during the whole sprint and test plan for the functional tests execution. All the lines show the possible test cases and the expected results (that would be to pass or to fail). The example shows a very small user story but this template can be followed for all the possible user stories on the sprint planning and for the whole sprint backlog, that in this new format, we are calling Test Backlog.

As a user I would like to have a carousel so that the main info of my Home page will be displayed to my final users.

- Check layout attached on IE8, IE9, FF(last version), Chrome(last version) and Safari 5 (pass)
- Resize the browser screen: Layout should adapt, marquee should keep the same (pass)
- Resize the browser screen: Should not affect the standalone text box (pass)
- Check if the time is displaying in hh:mm format (pass).
- Change the time manually (fail).
- The 3 banners in marquee are redirecting correctly
- Add text in the banner with 120 characters (pass).
- Add text in the banner with 118 characters (pass).
- Add text in the banner with 121 characters (fail).
- Add text in the banner with 121 characters (fail).
- Change the image from the banner for a psd image (fail).
- Change the image from the banner for a jpg image (fail).
- Check if the slider has 3 bullets (pass)
- Check if the sliders are linking to the correct 3 banners in each bullet (pass).

Figure 3. A User Story in the proposed format

It is possible to see on the Figure 4 the same image that is very famous on agile scenarios, but now, with the change to call the Product Backlog with a new nomenclature, "Test Backlog" (and Sprint Test Backlog in consequence). No other changes and no new step were added on the image (on purpose), since the idea is not to add new steps but to take advantage of the time for the backlog creation to do both works: requirement and test plan. This figure was done based on the original one.

Figure 4. Test Backlog in the Scrum Development (adapted from MontainGoat, 2012)

3.2. Testing work – Day by day activities proposal

As previously stated, the reason to name this phase "Incremental Test" is to make it clear that every piece of code can be tested and there is no need to wait until the entire user story is developed to start the test phase. Also, the testers should not be blocked in the beginning of the sprint anymore, waiting for the final code to be delivered for tests. This idea of unblocking the tester is even clearer when proposing a new task of helping the PO on writing good and testable user stories.

These activities are described in the Table 1. The first column presents the data evolution during the Sprint, in the second column the activities planned in current agile projects are shown and the last column presents the proposed activities allowing to compare the changes in the test team activities during the Sprint with the way they are being run currently on the projects.

Date in the Sprint	Current Agile Projects	Activities Proposal
Day 0 (before the planning)	There is no tester participation on this phase.	Like it was described on 3.1, the tester should work together with the PO on the user stories so the backlog will have a test format, what can be called as "Test Backlog".
Day 1 (Planning Meeting)	Tests responsible is always giving the size for the test related tasks and just involved in quality discussions.	There is a gap in the planning meeting for the tester role. Tester can be the more suitable professional to understand what is the customer necessities (customer point of view) and also, can explain it technically to the development team (technical point of view). The tester can filter team questions and raise new blocks to the scrum master.
Day 2 (first sprint development day)	Tester is creating the test plans and not fully involved on the other team members work.	Testing from the very beginning is involved in all tasks. Since the test plan was already created (the requirement will be used for that now), the tester can have his time to work in pairs with developers to make sure the entire team is on the same page and all the bugs that can be avoided will be discussed at this point. Here is the perfect time for the tester to work to help improving the unit test, having a stronger unit test is also the best way to avoid defects and

		issues in the future (and it´s also a very good way to run what we are calling a incremental test).
Day 3 to N (sprint development period)	Functional tests are executed following the test plan created in the Day 2 of the sprint. When there is no user story fully ready to be tested (which means with no other task open but the test execution), the tester(s) is usually blocked.	From the functional tests perspective, anything has to be changed: all the tests will be executed normally by the tester and also by the team, if it is necessary. From the regression tests perspective, this activity should be executed every time a new integration is done, not only on the last day of the sprint (also thinking on an incremental perspective).
Day N + 1 (last sprint development day)	Testers are still running functional tests and have no time to dedicate to the regression tests	Focus on the last Regression Tests execution. All the issues found on this phase should be prioritized and addressed to the next sprint. It´s not recommended fixing any issue during regression (all the work would need to be restarted in this case).

Table 1. Day by day tasks execution.

All these changes can make the day by day more productive and also, should keep the high quality in the deliverables. All the testers and team tasks and their day by day can be visualized on the Figure 5.

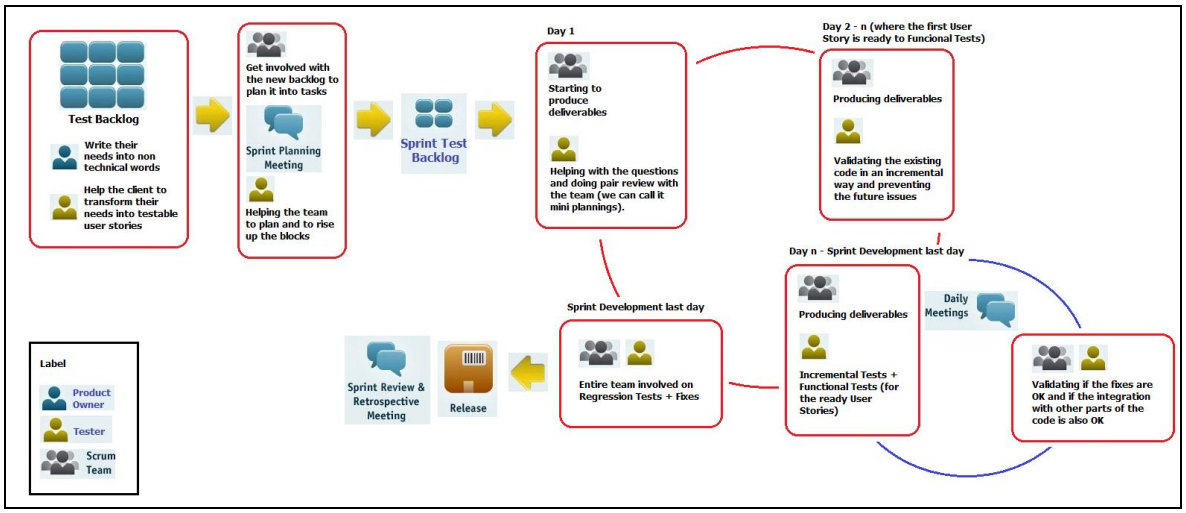


Figure 5. Proposed cycle with tasks and new backlog format.

3.3. Preliminary Results

The approach proposed on this paper for user stories construction was effectively tested in one of the projects that the author has being working as Scrum Master. The format of the team to be select as study case was chosen exactly for being the most similar to what

is found usually in the default agile teams (small team, multifunctional professionals and presence of a Product Owner and a Scrum Master). It is important to point out some of the main details and the structure of this team used as the study case:

- 6 people scrum team (3 developers, 2 web designers and 1 tester);
- Product Owner not fully allocated for the project;
- Scrum Master fully allocated for this team.

Table 2 shows the results that were obtained on this project and the number of opened tickets (during sprint development) was chosen as metric.

Sprint	Methodology	Defects/Issues opened during sprint development
1	1	221
2	2	76
3	3	28
4	3	35
5	3	19

Table 2. Project Results using the proposed approach

During the Sprint 1, a usual methodology was used to write the user stories (and that was called methodology 1). This one was more focused on the specification made in a very similar way to the old Use Cases model (from Rational Unified Process). The relevant factors to be cited are: the creation of the requirements only by the client (without scrum team help), many scope changes requested during the sprint (due to the lack on the prior specification) and user stories very low quality.

The migration from this model to the one it´s being proposed was actually started with an intermediate methodology (called here as 2). It was used during Sprint 2. The documents began to be performed with the team´s help. They´re still shaped like use cases but the long descriptions began to be replaced by a step by step testing.

The last three sprints (3 to 5) were performed using the methodology proposed on this paper (which we call 3). In this methodology, documentation and creation of user stories were made by the Product Owner with the help of the Test Analyst. The format adopted was to describe user stories as executable test cases, with steps and expected results as presented in Figure 3. A clear difference between the total of defects found in these sprints and the relevance of the proposal can be validated by comparing the results of sprints 1 to 5 on Table 2.

Finally, it is important to emphasize the fact that the only variable is the methodology used to create the user stories. Other factors were not changed, even the size of the sprints and the team members were kept the same so that the proposal could be validated with more consistence.

4. Conclusions

This work proposes a new format to prepare the User Stories including functional testing. Also, this new format proposal suggested the collaboration between the PO and the tester. They will help each other during the User Stories creation and both will help the team to better understand the requirements and user needs.

Most part of the issues that could be seen on scrum team work, related to test and product validation could be addressed if they work with less documentation and more alignment (communication) with the customer expectations. Creating the requirement together could be a very good way to address this issue and to have someone inside the development team with a very good vision of what is the real value to be achieved.

Another important issue that was found and could be better treated at this time is that some test teams spend a long period blocked during the sprint development waiting for the user stories to be totally ready for testing. Incremental tests idea and all the activities described on the subsection 3.2 can be a very good way to make the tester responsible to contribute for the project´s quality.

Results showed in Table 2 points to conclude that collaborations of the team during the user stories writing drastically decrease the number of defects/issues, since we observed almost 66% less defects/issues (from 221 down to 76) when only this variant was changed. Another important reduction of defects/issues was observed when the user stories were written using the new format proposed by this work. Again, 64% less defects/issues were observed (from 76 down to 28), culminating in a reduction of 75% (from 76 down to 19) when beyond the format changes the team acquires maturity using the new format.

This proposal can resemble to ATDD (Acceptance Test Driven Development), which makes sense, since some concepts were based on this technique. Developing the code based on a condition that has been created with a test view has much to do with the ATDD and TDD (Test Driven Development) and therefore, it is believed that an adaptation of the technique is very well fit to the Scrum.

The experience acquired as a Scrum Master in agile projects also helped in identifying the need for this adjustment, since in many cases, the client uses this time validating the planned tests as requirements and ends up finding gaps or even asking for changes in the starting . This idea can fit the real needs of the customer in the shortest time possible due to the fact that agile believe to be extremely important to generate the customer values.

If these teams start to use the activities proposed on this paper, most part of this blocks could be resolved and the resources would be more productive for the project as a whole. The key is a whole new mindset for the entire team working together to improve the quality work since the very first day, not only one test responsible can do that, but the scrum team together: improving requirement, unit tests, functional tests and acceptances tests.

References

Cohn, Mike. 2004. User Stories Applied. Addison Wesley, ISBN 0-321-20568-5.

Crispin, L. and Gregory, J. 2009. Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley. ISBN 0-321-53446-8.

Hendrickson, Elisabeth. 2008. Driving Development with Tests: ATDD and TDD, http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf, June.

MountainGoat. 2012. Introduction to Scrum, http://www.mountaingoatsoftware.com, June.

Pfleeger, S. L. and Atlee, J.M. 2009. Software Engineering: Theory and Practice. Prentince Hall. ISBN 978-0136061694.

ScrumAlliance (2012) Official Website, https://http://www.scrumalliance.org, June.

Soares, Michel dos Santos. 2004. Agile Methodologies: Extreme Programming and Scrum for Software Development. System Information Eletronic Magazine, Vol. 3, Number 1.

Sommerville, Ian. 2010. Software Engineering, Pearson Addison-Wesley, 9th Edition.

Estudo sobre dependências e suas consequências no cenário Scrum and Scrum (SandS)

Diego da S. Souza[1], **Rodrigo de Toledo**[1,2], **Jonice Oliveira**[1,2]

[1]Programa de Pos-Graduação em Informática- PPGI
Universidade Federal do Rio de Janeiro (UFRJ)

[2]Departamento de Ciência da Computação
Universidade Federal do Rio de Janeiro (UFRJ)

diessouza@ufrj.br,{rtoledo,jonice}@dcc.ufrj.br

Abstract. *Producing software is an increasingly more difficult activity due to the constantly changing requirements, the greater demand from customers, and the complexity and size of the systems. Due to this, development teams have found difficult to keep their work free of dependencies between tasks. To reduce these problems, Scrum has been one of the most used agile methodologies, because in this method, adjustments can be made to better meet the needs of the team. Thus, this study is a contribution to the Agile community, describing the Scrum and Scrum scenario and addressing the problem of dependency between tasks. A proposed work will be described to mitigate this impact.*

Resumo. *Produzir software é uma atividade cada vez mais difícil devido a constante mudança de requisitos, a maior exigência por parte dos clientes, a complexidade e tamanho dos sistemas. Devido a isso, equipes de desenvolvimento têm encontrado dificuldade em manter seu trabalho livre de dependências entre tarefas. Para amenizar esses problemas, o Scrum vem sendo uma das metodologias ágeis mais utilizadas e positivas, pois adaptações podem ser realizadas nesse método para melhor atender às necessidades da equipe. Diante disso, esta pesquisa vem contribuir para a comunidade ágil descrevendo o cenário de Scrum and Scrum e abordando o problema de dependência entre tarefas. Uma proposta de trabalho será descrita visando amenizar o impacto das mesmas.*

1. Introdução

Desde os anos 2000, tivemos um crescimento das chamadas metodologias ágeis ou metodologias leves [Fowler 2005]. Esses métodos de desenvolvimento surgiram como uma alternativa para os tradicionais modos de produzir um sistema. Tais metodologias tentam buscar meios de contornar ou suavizar os problemas decorrentes do desenvolvimento tradicional de software.

Mudando alguns paradigmas, antes existentes no modelo tradicional de produção de sistemas, as metodologias ágeis inseriram alguns benefícios nesse ambiente. Dentre os muitos existentes, podemos citar a criação do sentido de equipe/time, pois agora os times são multidisciplinares, o trabalho em par (*pair programming*) e reuniões diárias para estimular a colaboração e comunicação entre os membros. Outro benefício que podemos acrescentar é a iniciativa da transparência, que propõe que as informações estejam visíveis

para todos os membros do time. Isso faz com que o desenvolvimento seja acompanhado de perto por todos que participam do seu processo, inclusive clientes.

Porém, empresas de médio e grande porte, com times grandes, precisam adotar outras técnicas para utilizar o Scrum [Teles 2012], pois este framework trabalha com times pequenos, com uma quantidade de 5 a 9 membros. Uma técnica encontrada para contornar esse fato é a adoção de medidas como o Scrum of Scrums (SofS) [Cohn 2012] ou o Scrum and Scrum (SandS).

O cenário de SofS já vem sendo relatado na literatura há alguns anos, apesar de ser posterior a criação do framework Scrum [Sutherland and Schwaber 2011]. Mike Cohn [Cohn 2005] explica o SofS no contexto de um único projeto feito por diversos times. Já o contexto em que múltiplos times trabalham em múltiplos projetos dificilmente aparece na literatura, apesar de ser comum sua adoção em empresas. Este cenário não apresenta uma nomenclatura formal na comunidade ágil, por isso, criamos e adotamos o termo SandS para a descrição do mesmo.

Diante disso, esta pesquisa vem contribuir para a comunidade de métodos ágeis definindo e caracterizando um ambiente de SandS. É feita uma proposta de estudo visando uma solução para o problema de dependência entre as tarefas dos diversos times existentes no SandS

Dessa forma, este presente trabalho está estruturado da seguinte maneira: a seção seguinte a este descreverá o cenário de estudo, o SandS, diferenciando-o do ambiente de SofS. Na Seção 3 apresentaremos os trabalhos relacionados, essa seção trará algumas opiniões e trabalhos realizados por grandes nomes da literatura de métodos ágeis. Na Seção 4 o problema das dependências será abordado, e comparado em 3 diferentes cenários. A proposta de trabalho será apresentada na Seção 5. A Seção 6 trará a conclusão desta pesquisa, assim como o próximo passo que será realizado.

2. Caracterizando o cenário SandS

Sendo um ambiente multitimes, onde cada equipe encontra-se alocada para o desenvolvimento de um projeto, o SandS tem sido uma estratégia de desenvolvimento adotado por empresas que precisam produzir mais de um produto simultaneamente, e desejam utilizar o Scrum como metodologia.

Esta seção caracteriza o SandS e está dividida com a seguinte estrutura: o primeiro subtópico terá a finalidade de apresentar o cenário. No subtópico seguinte será feita uma comparação entre os dois cenários multitimes: SofS e SandS.

2.1. Visão Geral

O cenário para o nosso estudo será o de empresas de desenvolvimento de sistemas de médio e grande porte [SEBRAE 2004], que possuem um elevado número de desenvolvedores. Outra característica desse contexto é a presença de vários projetos sendo executados simultaneamente. Tal cenário descrito anteriormente não permite a utilização de um único time de Scrum, pois o mesmo é limitado a 9 membros.

De acordo com a configuração dessas empresas, dois métodos podem ser aplicados: o SofS ou o SandS. No primeiro cenário temos a utilização de vários times Scrum participando de um mesmo projeto. No outro cenário, SandS, cada time trabalha em um

projeto diferente, compartilhando apenas o mesmo ambiente. Esse modelo vem sendo amplamente utilizado em empresas, apesar da sua pouca literatura.

Nesse cenário, vários times Scrum ficam localizados em um mesmo ambiente de desenvolvimento, onde cada um executa a criação do seu respectivo projeto. Cada time Scrum executa suas atividades de acordo com o framework, realizando os eventos e produzindo os artefatos do Scrum.

Os mesmos papéis encontrados no Scrum perseveram no ambiente de Scrum and Scrum. Além do Time, o Product Owner (PO) e o Scrum Master se fazem presentes nas equipes de desenvolvimento, desempenhando as mesmas funcionalidades descritas pelo framework do Scrum em seu uso tradicional. Porém, por existirem vários projetos nesse ambiente, existirão vários POs e vários Scrum Masters.

Outra característica desse ambiente é viabilizar a disseminação do conhecimento, presente no Scrum, e a colaboração entre os vários membros das muitas equipes existentes. Os POs, por exemplo, podem compartilhar experiências e métodos utilizados na hora de criar, ”fatiar”e priorizar as tarefas, assim como os Scrum Masters podem se auxiliar na hora de manter o time. Membros pertencentes aos vários times podem compartilhar técnicas de desenvolvimento, estimulando o princípio da melhoria contínua.

2.2. Diferenças entre SandS e SofS

Apesar da grande semelhança entre os dois princípios, eles não devem ser confundidos. Enquanto o modelo SofS é caracterizado por diversos times Scrum trabalhando em um mesmo projeto, o modelo SandS é caracterizado por diversos times trabalhando em um mesmo ambiente, porém cada um dedica-se a um projeto distinto.

De uma maneira genérica, o Scrum of Scrums também é apresentado como uma técnica para escalar uma equipe de desenvolvedores em times menores, para que assim, seja possível aplicar o framework do Scrum. Alguns artefatos e/ou eventos do Scrum sofrem pequenas alterações, temos a adição de novos itens no framework para auxiliar o trabalho com multitimes, porém, em sua essência, o seu funcionamento não é afetado.

Cada time menor possui o seu Scrum Master, que assim como no Scrum tradicional, possui os mesmos deveres. O Scrum Master deve proteger o time, livrando-o de situações que possam causar impedimentos na execução da Sprint. Outra atribuição do Scrum Master é garantir que o Scrum seja entendido pela sua equipe e certificar-se que o framework está sendo aplicado de maneira efetiva.

Como esse framework é um conjunto de práticas que auxilia o desenvolvimento e garante uma grande flexibilidade para a sua execução, vários autores e profissionais da área entendem e adaptam o SofS para atender as necessidades de seu ambiente de desenvolvimento. Contudo, alguns eventos sempre aparecem nessas diversas estruturas do SofS. A execução de reuniões entre os Scrum Masters ou de representantes das diversas equipes Scrum é um desses eventos que foram incorporados ao SofS. Essa nova reunião é parecida com a Reunião Diária, evento do Scrum tradicional, porém ela é definida para ocorrer entre os Líderes [1] dos diversos times. Nessa reunião, são abordadas e discutidas as tarefas realizadas pelos times, o que será realizado em novos dias de trabalho e finalmente,

[1]Líderes - Membro(s) do time Scrum responsável(is) por apresentar aos outros times as atividades executas pelo seu, além de impedimentos, boas práticas, etc.

quais os impedimentos que estão afetando as equipes. É um bom momento para que as outras equipes tomem conhecimento das atividades dos times parceiros.

Já em cenários SandS, essas reuniões podem ou não ocorrer, como mencionado anteriormente, cada profissional estabelece o que é necessário para o seu time. Nesse cenário, alguns optam por não ter esse contato entre as diversas equipes, já que trabalham em produtos distintos. Por outro lado, quando esses encontros existem, devem ser controlados para que o foco da reunião não seja perdido. Essa reunião pode abordar os seguintes pontos: a discussão de impedimentos que atrapalham o andamento de um projeto; quais medidas foram adotadas por um time para contornar um determinado problema; ou até mesmo um diálogo verificando a adoção de práticas ou medidas de forma a contribuir com a melhoria contínua do ambiente.

3. Trabalhos relacionados

Conforme citado, há pouco material na literatura referindo-se a esses dois temas. Em relação ao SofS ainda encontramos algumas opiniões de autores renomados em desenvolvimento ágil. Já relacionado ao SandS, não foram encontrados materiais de consultas ou que serviriam como base para este trabalho. Desse modo, essa pesquisa tem objetivo de contribuir para a comunidade ágil, retratando esse cenário que comumente se faz presente em desenvolvimento de sistemas. Os parágrafos seguintes, trazem relatos e comentários de alguns autores sobre SofS e sobre o escalonamento de equipes.

Mike Cohn, um dos principais nomes do movimento ágil e escritor de vários livros como ”User Stories Applied: For Agile Software Development”[Cohn 2004] e ”Agile Estimating and Planning”[Cohn 2005], afirma que as reuniões Scrum of Scrums são um importante mecanismo para escalar projetos grandes. Para Cohn, o Scrum mostra-se funcional quando utilizado com pequenas equipes. Ele ainda ressalta que essas Reuniões Scrum of Scrums permitem que os times discutam os seus trabalhos, mostrando suas dificuldades e soluções, possibilitando uma maior interação entre os times. Não existe um método para a escolha de quem irá participar da reunião, mas para Cohn, essa decisão deverá ficar a cargo do próprio time. O time escolhe qual será o representante (ou os representantes) para a Reunião Scrum of Scrums. Cohn ainda ressalta que, usualmente, a pessoa escolhida é um contribuidor técnico, como desenvolvedor, designer, administrador de banco de dados no lugar do Scrum Master ou do próprio PO.

Já Henrik Kniberg, autor do livro que fala sobre XP e Scrum [Kniberg 2007], que retrata passo a passo o ambiente de desenvolvimento o qual estava inserido, vê a Reunião Scrum of Scrums como uma oportunidade para os Scrum Masters conversarem e discutirem a respeito do trabalho realizado em cada um dos times. Kniberg ainda diz que é aceitável uma equipe com números variando de 3 a 8 pessoas. Ressalta também que a escolha das equipes pode ser uma tarefa delicada, e desse modo, em seu ambiente de trabalho, adotaram um mecanismo de criação do papel de *team lead*, líder de equipe. Esse membro não coordena nenhuma equipe em específico, mas fica a disposição para resolver os problemas que podem ocorrer entre as diversas equipes. Alguns problemas que são de responsabilidade desse líder são a escolha de quem será o Scrum Master de cada time e a divisão dos desenvolvedores entre os times.

Allan Shalloway em seu livro [Shalloway et al. 2009], cita a dificuldade de se trabalhar com equipes grandes. Shalloway diz que vê problemas quando se trabalha com um

grupo de muitos desenvolvedores, 350, por exemplo. Ele diz que, não um produto, mas vários estão sendo construídos com elementos compartilhados entre os múltiplos times. E, dependendo da organização desses vários times, itens podem ser desenvolvidos mais de uma vez por mais de um time. Shalloway ainda explicita alguns problemas que podem ocorrer em cenários como esse. Itens com repetição, citado anteriormente, problemas de comunicação e concordância entre os diversos membros das equipes, também podem ser um grave problema, além de dificuldades em escalonar, de forma eficiente, os vários times.

Ainda existem outras opiniões e comentários a respeito do modelo SofS e da escalabilidade de equipes, eles foram estudados e analisados, porém preferimos não colocá-los nesta seção por serem parecidos com os anteriores.

4. Caracterizando os problemas de dependência

Dependências entre tarefas de um mesmo projeto é um assunto que já acompanha áreas da Engenharia de Software [Cataldo et al. 2008] [Cataldo et al. 2009]. Saber contornar ou minimizar o impacto que elas provocam é uma tarefa difícil e merece atenção da pesquisa acadêmica. Atrasos na entrega do produto, ociosidade por membros da equipe são alguns impactos que elas podem gerar se não forem bem administradas. Em ambientes que fazem uso de metodologias ágeis, essas dependências podem gerar Sprints incompletas, o que pode resultar em um atraso no produto final. Nesta seção abordaremos o problema das dependências entre tarefas em três diferentes cenários, comparando-os.

4.1. Análise de dependência em um ambiente tradicional

Quando falamos em desenvolvimento tradicional, estamos nos referindo ao modelo de produção de sistemas conhecido como cascata [Royce 1970]. Esse tipo de produção é caracterizado pelo desenvolvimento em camadas, além de outras características, como a presença de um sequenciamento das atividades a serem executadas durante a produção do software. A figura abaixo (Figura 1) ilustra o desenvolvimento em cascata.

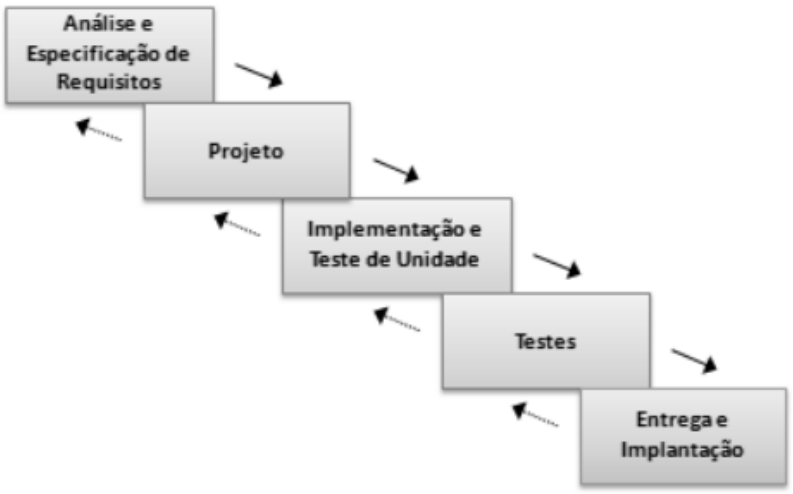


Figura 1. Modelo Cascata.

Nesse contexto, uma etapa se inicia somente com o fim da anterior, pois é ela que irá fornecer os *inputs* necessários para a nova fase. Isso significa que a etapa de codificação, por exemplo, só irá começar após as etapas de coleta de requisitos e de modelagem do projeto estiverem terminadas. Na prática, pode ocorrer uma sobreposição dessas fases. Algumas vezes há a necessidade de retornar a alguma etapa que já foi finalizada, com o objetivo de corrigir algum erro. Essa configuração de etapas sequenciadas pode ser um alerta para a existência de dependências.

Além disso, esse modelo também é caracterizado por uma divisão das equipes em especialistas. Desse modo, cada fase fica sob responsabilidade de uma equipe que possui um conhecimento específico. As pessoas que participam de uma fase geralmente não estão presentes nas outras. Assim, membros que participam da produção do sistema são alocados para executar atividades de uma única fase. A fase de captura de requisitos, por exemplo, ficará sob responsabilidade de analistas de requisitos. Esse acontecimento também promove a existência de dependências. Muitas vezes, membros de outras fases acabam ficando sem nenhuma atividade, pois ainda precisam esperar os responsáveis pelas fases anteriores terminarem os seus trabalhos, liberando recursos.

Outro tipo de dependência que aparece muito nos cenários de produção de software é a dependência entre atividades da mesma fase. Para entender esse acontecimento, devemos entender que um processo de software é composto por várias atividades [Pressman 2006]. Cada atividade necessita de um insumo e gera uma saída. Os insumos necessários para uma determinada atividade provêm de outras atividades, ou seja, de outras tarefas que estão sendo executadas por membros da equipe de desenvolvimento. Desse modo, se as tarefas não forem divididas e atribuídas de forma a contornar ou a minimizar essa dependência, o resultado poderá ser a ociosidade de alguns funcionários e/ou atrasos na entrega do produto final. Esse fato se agrava quando temos atrasos por parte dos desenvolvedores durante a execução das tarefas. Outro fator que contribui para a causa de dependências é a utilização do conceito de especialistas durante a execução de atividades da mesma fase. Isso faz com que, quando aplicado o desenvolvimento em camadas, equipes não especializadas no assunto corrente fiquem inteiramente ociosas. Por exemplo, desenvolvedores de regras de negócio devem esperar tarefas relacionadas a banco de dados serem finalizadas para iniciar seus trabalhos.

Algumas pesquisas buscam tentativas de contornar esse tipo de ocorrência. Áreas como gerência de impacto [de Souza and Redmiles 2008], buscam minimizar os efeitos causados por esse tipo de dependência. Esse estudo foi realizado com duas equipes de desenvolvimento reais, e uma das formas de tentar reduzir a existência das dependências era uma boa comunicação entre os desenvolvedores. Estratégias, como envio de e-mail para os outros membros, eram adotadas para melhorar a comunicação e diminuir o impacto entre as tarefas.

4.2. Análise de dependência em um ambiente SofS

Quando se adota o modelo Scrum of Scrums de desenvolvimento de sistemas, esse cenário de dependências diminui um pouco. Essa redução ocorre principalmente pelo SofS ser um modelo ágil de desenvolvimento, e com isso, possuir uma divisão da equipe em times Scrum multidisciplinares e um desenvolvimento em fatias. Esses times Scrum recebem esse adjetivo, pois um mesmo time pode realizar todas as funções necessárias para entregar um pedaço funcional do sistema final.

Para ilustrar a forma que a multidisciplinaridade auxilia na redução das dependências, podemos pensar em duas equipes de desenvolvimento, uma delas utilizando métodos tradicionais para produzir o software (Equipe A) e outra utilizando o framework Scrum, aplicado em um cenário SofS (Equipe B). Quando uma atividade de cadastro de aluguel, por exemplo, é pedida a Equipe A, esta deverá analisar quais são as entidades envolvidas no relacionamento, e deixar pessoas responsáveis pelo banco de dados executarem as criações das entidades, relacionamentos, etc., para que depois os desenvolvedores codifiquem as regras de negócio do aluguel. Se essa mesma atividade for atribuída para a Equipe B, esta segunda equipe também irá analisá-la, planejar qual a melhor forma de execução da mesma e dividi-la em tarefas. Em uma equipe multidisciplinar ideal, todos do time têm aptidão para executar todas as tarefas, ou seja, os membros do time possuem habilidades parecidas (devido a troca de experiências durante a programação em pares). Dessa maneira, vários membros do time podem executar as tarefas ao mesmo tempo, de modo a finalizar a atividade, diminuindo as dependências. Com o decorrer do tempo e a evolução do time devido às práticas já citadas, a equipe B tende a se tornar uma equipe multidisciplinar ideal.

No SofS, diversos times executam tarefas relacionadas a um único produto, portanto, deve-se tomar cuidado com as tarefas que podem ser dependentes. Um time poderá ficar dependente de um segundo time e ter que esperá-lo terminar sua tarefa para liberar o *input* necessário para o primeiro iniciar a execução da sua. O papel do PO (ou POs) é de extrema importância nesse cenário, pois saber ”fatiar”e priorizar uma atividade com o objetivo de minimizar as consequências da mesma, se torna difícil devido ao intrínseco relacionamento entre os diversos artefatos e atividades de um sistema [Grinter 1998].

O cenário SofS tem uma redução das dependências entre atividades, porém elas ainda existem e podem afetar o ambiente de produção, fazendo com que Sprints possam ser entregues incompletas. Porém, deve-se tomar cuidado com a divisão das atividades entre os times, pois se mal divididas, muitas dependências podem ser geradas.

4.3. Análise de dependência em um ambiente SandS

O modelo de desenvolvimento baseado no Scrum and Scrum vem sendo muito utilizado por várias empresas [Bardusco 2008]. Ele se mostra como uma alternativa de aplicação do framework do Scrum quando uma empresa possui muitos desenvolvedores e ainda há uma necessidade de produzir vários projetos em paralelo. As dependências entre as atividades também estão presentes neste contexto, porém com uma menor intensidade que em relação ao SofS.

Ao contrário do que ocorria no modelo de SofS, no SandS cada time se dedica exclusivamente a desenvolver um projeto, ou seja, cada time está responsável por um único produto final. Essa situação já proporciona uma diminuição ainda maior na ocorrência de dependências, pois um time Scrum, do modelo SandS, não precisa depender do outro time desenvolver alguma tarefa relacionada ao seu projeto. Mas elas ainda existem e merecem atenção.

Dependências neste cenário podem ser classificadas como indiretas. Por exemplo: Um time (time A) está alocado para trabalhar no desenvolvimento de um determinado projeto para o qual necessitou desenvolver três bibliotecas. Eventualmente, uma dessas bibliotecas veio a ser usada por outros times em outros projetos, porém não atendia esses

times por completo, alguns ajustes eram necessários. Como essa biblioteca foi criada pelo time A, um bom cenário é esse mesmo time ficar responsável pelas alterações que eram necessárias, a fim de evitar, neste contexto, retrabalho e perda de tempo por parte dos times. Desse modo, algumas tarefas de alteração na biblioteca foram acrescentadas ao Backlog desse time. Os outros times tornam-se dependentes do time A realizar as alterações na biblioteca para continuar o seu trabalho, gerando dependências.

4.4. Scrum e dependências em linhas gerais

A gerência e as práticas do Scrum se tornam mais fáceis de serem aplicadas quando existe um time apenas alocado por projeto, proporcionando uma redução das dependências. Eventos como as Reuniões Diárias permitem uma maior comunicação entre os membros do time, fazendo com que todos de um mesmo time saibam o que os outros membros estão fazendo.

5. Proposta de Trabalho

O objetivo principal deste trabalho é amenizar o problema das dependências entre tarefas que existem em ambientes de desenvolvimentos multitimes. Para isso, foi escolhido um cenário que faz uso do modelo de SandS, para observação, análise dos dados e aplicação da solução encontrada. Assim, podemos listar os seguintes objetivos específicos:

- Analisar um cenário real que faça uso do modelo SandS para observação, análise dos dados e aplicação da solução encontrada.
- Propor uma abordagem de visualização das dependências.
- Coleta e análise das dependências (identificação, mensuração e etc.).
- Verificar a existência de padrões, práticas e realizar análises sobre os times.
- Criação de possíveis soluções, por exemplo redimensionamento de tarefas e equipe.

Esta proposta poderá ser dividida em várias etapas para que seu objetivo seja atingido, como ilustrado na figura abaixo (Figura 2).

1. Fomentar visibilidade das dependências A primeira etapa da pesquisa terá o objetivo de permitir ou aumentar a visualização das dependências entre tarefas. Muitas vezes essas dependências existem, mas não são identificadas. Para esta etapa, serão necessárias visitas e negociações com empresas que tenham seus ambientes de desenvolvimento semelhantes ao cenário escolhido para a pesquisa. Algumas empresas serão escolhidas para a avaliação de acordo com alguns critérios. Dentre os critérios já estabelecidos, podemos citar o uso de metodologias ágeis e ambiente de produção de sistemas voltado para multitimes. Um mecanismo deverá ser inserido nas empresas escolhidas para tornar transparente para todas as equipes as dependências entre atividades. Aproveitando a simplicidade das metodologias ágeis ao utilizar quadros em seus frameworks, uma tabela fixada (Figura 3) em um quadro foi pensada como sendo o mecanismo principal para a visualização. Essa simples ferramenta permite que a informação fique no próprio ambiente de desenvolvimento, visível a todos, sem necessidade de outros recursos de maior complexidade. Os times estarão dispostos nas linhas e colunas da tabela, e a relação será *linha* depende de *coluna*, constituindo uma DSM (*Dependency Structure Matrix*)

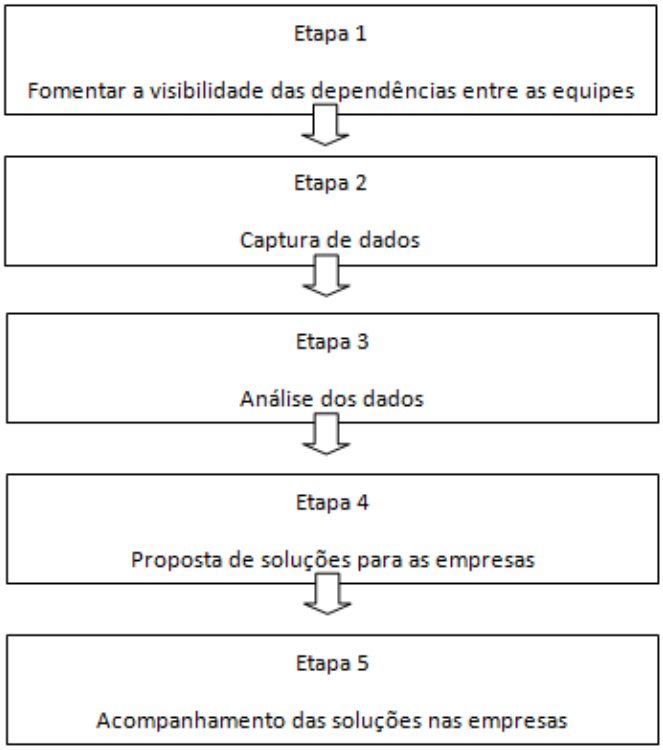


Figura 2. Etapas da pesquisa.

[Browning 1998]. Um post-it (Figura 4) contendo alguns campos para preenchimento será utilizado para marcar a dependência ocorrida entre times. Esses campos serão usados para identificar motivos, duração e tipo das dependências. Os campos para preenchimento serão pequenos e em pouca quantidade, de modo que o membro do time não gaste seu tempo preenchendo o post-it. Para diferenciar o nível das dependências e facilitar a visualização, três post-its de diferentes cores serão utilizados. O de cor verde será usado quando ocorrer uma dependência de menor nível crítico, enquanto o rosa (mais próximo da cor vermelha) irá representar uma de maior nível crítico, a cor amarela representará um nível intermediário entre as outras.

Durante essa etapa, será necessária a utilização da etnografia [Easterbrook et al. 2007] [Robinson et al. 2007]. Esta abordagem terá por objetivo certificar-se de que o experimento está sendo realizado como o esperado, ou seja, se os membros do time estão manipulando o quadro proposto e os post-its corretamente. A etnografia foi escolhida por ser uma técnica muito utilizada em áreas relacionadas a computação, como por exemplo, Trabalho Cooperativo com Auxílio de Computadores (*Computer Supported Cooperative Work - CSCW*) [Shah and Harrold 2010].

2. Captura Na segunda etapa, captura, o quadro irá fornecer as informações desejadas. Para isso, em intervalos regulares de tempo, sincronizados com as Sprints, uma fotografia do quadro será tirada. Esse procedimento se repetirá em cada empresa alvo e poderá ser realizado pelo próprio time. O uso de uma abordagem etnográfica nesta etapa também será útil, pois através dela teremos uma maior

	Time A	Time B	Time C	Time D	Time E	Time F
Time A	X					
Time B		X				
Time C			X			
Time D				X		
Time E					X	
Time F						X

Figura 3. Mecanismo que será utilizado para visualização das dependências.

Figura 4. Diferentes post-its para identificar as dependências.

quantidade de informações a respeito das dependências ocorridas. Informações oriundas da observação do comportamento dos membros dos times e de entrevistas com os mesmos, serão úteis para o melhor entendimento da ocorrência das dependências. Ao final dessa etapa, teremos várias fotografias de diferentes times e de diferentes dependências entre tarefas, assim como as informações provenientes das atividades de observação e entrevista.

3. Análise A terceira etapa da pesquisa será a análise dos dados. Após cada coleta, uma análise desses dados será realizada, verificando quais equipes são mais dependentes e aquelas que são menos dependentes. Os dados coletados serão analisados sob as seguintes formas: quantitativamente e qualitativamente. De maneira quantitativa foram pensadas medidas como **frequência** e **duração** das dependências. Já sob aspecto qualitativo, medidas como **criticidade** (o quão crítico foi uma dependência) e **tipo** serão verificadas. Essa interpretação dos dados terá como objetivo analisar e verificar os motivos que geram as dependências. A comparação entre times de diferentes empresas também será realizada com o objetivo de interpretar e buscar práticas utilizadas em uma empresa que diminua ou aumente a dependência entre tarefas.

4. Idealização de Soluções Após a análise dos dados, inicia-se a quarta etapa da pesquisa: a descoberta e o fornecimento de soluções para as empresas. A princípio, três tipos de possíveis soluções foram idealizadas. São elas: mudanças nas práticas e medidas adotadas pelos times ou adoção de novas, proposta de adoção

de algumas ferramentas que venham melhorar o ambiente de desenvolvimento, ou proposta de reorganização dos membros do time. Elas foram baseadas em experiências dos autores no cenário de metodologias ágeis. Essas soluções serão estudadas para verificar qual se encaixará no perfil de cada ambiente de trabalho, porém, não nos limitaremos a elas. A partir da análise feita, novas soluções poderão ser propostas.

5. Proposta de Soluções Na quinta etapa, será realizado um acompanhamento do ambiente das empresas após o fornecimento das possíveis soluções. Existe um risco nessa etapa, pois dependemos das empresas adotarem nossas soluções. Com esse acompanhamento, conseguiremos verificar se as soluções propostas realmente auxiliam, em ambientes reais, a diminuição das dependências entre tarefas.

6. Conclusão

Essa pesquisa vem contribuir para a comunidade ágil, pois se dedica a pesquisa e descrição de um cenário que vem ocorrendo em muitas empresas. O cenário de SandS foi identificado como um novo contexto envolvendo a produção de software. Como uma forma de escalar equipes grandes em times Scrum, esse modelo de desenvolvimento é uma alternativa para empresas que desejam adotar a metodologia do Scrum, produzindo vários projetos simultaneamente. Vale reforçar que não foram encontrados trabalhos relacionados a esse tema, assim, o cenário SandS está sendo definido por nossa pesquisa. O nome Scrum and Scrum foi criado baseado no modelo Scrum of Scrums, pois ambos os contextos são parecidos na forma de escalonar equipes grandes.

Pesquisas sobre a aparição de dependências e suas consequências também foram alvo deste trabalho. Uma proposta de trabalho com o objetivo de amenizar o impacto da mesma foi apresentado. Um mecanismo capaz de proporcionar maior transparência da ocorrência de dependências foi proposto, assim como mecanismo para a captura de dados. Práticas ou medidas adotadas pelos times serão pesquisadas para a identificação de padrões de desenvolvimento. Ao final da análise dos dados, algumas soluções serão idealizadas como forma de amenizar a ocorrência das dependências entre os times.

O próximo passo é iniciar a captura das informações em cenários reais de desenvolvimento, ou seja, fazer um estudo em empresas sobre como e por que as dependências ocorrem.

Referências Bibliográficas

Bardusco, D. (2008). Scrum na globo.com: Derrubando mitos. http://blog.bardusco.com/2008/10/25/falando-em-agile-2008/ - junho.

Browning, T. (1998). Use of dependency structure matrices for product development cycle time reduction. In *ISPE '98: Proceedings of the Fifth ISPE International Conference on Concurrent Engineering: Research and Applications*.

Cataldo, M., Herbsleb, J. D., and Carley, K. M. (2008). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11, New York, NY, USA. ACM.

Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D. (2009). Software dependencies, work dependencies, and their impact on failures. *IEEE Trans. Software Eng.*, 35(6):864–878.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Cohn, M. (2005). *Agile Estimating and Planning*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Cohn, M. (2012). Scrums meeting. http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting/ - junho.

de Souza, C. R. B. and Redmiles, D. F. (2008). An empirical study of software developers' management of dependencies and changes. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 241–250, New York, NY, USA. ACM.

Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2007). *Selecting Empirical Methods for Software Engineering Research*. Springer.

Fowler, M. (2005). The new methodology. White Paper - http://martinfowler.com/articles/newMethodology.html - junho.

Grinter, R. E. (1998). Recomposition: Putting it all back together again. In *CSCW*, pages 393–402.

Kniberg, H. (2007). *Scrum and XP From the Trenches*. C4Media, 1st edition.

Pressman, R. (2006). *Engenharia de Software*. McGraw-Hill; 6th edition.

Robinson, H., Segal, J., and Sharp, H. (2007). Ethnographically-informed empirical studies of software practice. *Information and Software Technology*, 49(6):540–551.

Royce, W. (1970). Managing the development of large software systems. *Proc. IEEE WESTCON, Los Angeles*, pages 1–9.

SEBRAE (2004). Fatores condicionantes e taxa de mortalidade de empresas no brasil. http://www.biblioteca.sebrae.com.br/bds/bds.nsf/9A2916A2D7D88-C4D03256EEE00489AB1/$File/NT0008E4CA.pdf - junho.

Shah, H. and Harrold, M. J. (2010). Studying human and social aspects of testing in a service-based software company: case study. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '10, pages 102–108, New York, NY, USA. ACM.

Shalloway, A., Beaver, G., and ames R. Trott (2009). *Lean-Agile Software Development: Achieving Enterprise Agility*. Addison Wesley; 1st edition.

Sutherland, J. and Schwaber, K. (2011). The scrum guide. http://www.scrum.org/scrumguides/ - junho.

Teles, V. M. (2012). Scrum team. http://improveit.com.br/scrum/scrum_team - junho.

Proposta de um plano de métricas para o monitoramento de projetos Scrum

Eduardo H. Spies, Duncan Dubugras A. Ruiz

Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 1429 – 90619-900 – Porto Alegre – RS – Brazil

eduardo.spies@acad.pucrs.br, duncan.ruiz@pucrs.br

Abstract. *A lot of critics have been made to agile development methods due the simplified process definition and lack of control. Sometimes these methods are characterized as without rigor or control. However, several studies have been presenting effective ways to gain control while maintaining the agility of agile development projects. This paper presents a metrics plan focused on monitoring and tracking of Scrum projects, aiming to reach a satisfactory level of control without interfering with the main principles or structure of this framework.*

Resumo. *Muitas críticas a métodos ágeis são feitas devido a simplicidade das definições de processo e de como o controle do projeto deve ser realizado. Por vezes, eles vêm sendo caracterizados como métodos sem rigidez ou controle. Contudo, diversas pesquisas vêm apresentando maneiras eficazes para o controle deles, mantendo a agilidade de projetos de desenvolvimento. Este artigo apresenta um plano de métricas focado no monitoramento e acompanhamento de projetos Scrum, almejando alcançar um nível satisfatório de controle sem interferir nos príncipios ágeis ou na estrutura deste framework.*

1. Introdução

Nos últimos anos métodos ágeis vêm se tornando um tema bastante recorrente, tanto na indústria [Williams, Brown, Meltzer, & Nagappan, 2011] quanto em trabalhos acadêmicos [Torgeir Dingsøyr, 2012] Estes métodos utilizam conhecimentos empíricos para melhor gerenciar projetos complexos, de requisitos instáveis, e para melhor responder às mudanças. Em geral estes métodos seguem princípios definidos no manifesto ágil e compartilham algumas características como o desenvolvimento através de pequenas iterações, entrega frequente de software ao cliente e aumento do poder do time de desenvolvimento de um projeto. No âmbito das publicações, a grande maioria fala sobre *Extreme Programming* (XP) apesar do Scrum ser mais dominante na indústria [Dingsøyr, 2012]. Contudo, nos últimos anos, o Scrum vem ganhando cada vez mais destaque já ultrapassando outros métodos em publicações anuais [Dingsøyr, 2012].

Métodos ágeis em geral são recomendados para projetos de pequeno porte em virtude de adotarem um processo de desenvolvimento simples e intuitivo, buscando não onerar o time de desenvolvimento com atividades burocráticas que não agreguem valor [Sommerville, 2011]. Contudo, existem críticas quanto à falta de controle em projetos ágeis justamente devido a estas características de não definição clara de processos, para

permitir avaliar e monitorar o desempenho do projeto [Longstreet, 2011]. Outros autores discutem os benefícios de utilizar práticas de modelos como o CMMI para complementar os métodos ágeis, especialmente quando o porte dos projetos cresce. É esperado conseguir, desta forma, aproveitar o controle e a disciplina destes modelos em conjunto com a agilidade e a rapidez de resposta a mudanças dos métodos ágeis [Jakobsen & Johnson, 2008], [Mahnic & Zabkar, 2008], [Jakobsen & Sutherland, 2009].

Esse artigo apresenta a proposta de um plano de métricas para o monitoramento de projetos Scrum, que seja compatível com os princípios e critérios para criação de métricas ágeis já definidos [Hartmann & Dymond, 2006]. Este plano foi criado com a intenção de ser utilizado no ambiente de gestão de métricas de projeto SPDW+, o qual é baseado em Data warehouse (DW) [Silveira, Becker, & Ruiz, 2010] e tem o intuito de automatizar o processo de coleta de métricas. O plano de métricas apresentado é o passo inicial na adaptação e evolução do ambiente SPDW+, desenvolvido inicialmente para uma abordagem tradicional de desenvolvimento, em cascata, para que o mesmo possa ser utilizado em métodos ágeis. Espera-se prover, a este tipo de projeto, uma estrutura de gerência de métricas de baixa intrusão (pouco envolvimento dos colaboradores), baseado em um processo de ETC (extração, transformação e carga) automatizado.

2. O problema

O Scrum possui características que aumentam a adaptabilidade e produtividade em projetos de desenvolvimento de software. Todavia, em projetos de maior porte, o arcabouço Scrum se beneficia da utilização de práticas de engenharia de software que tem por objetivo aumentar a disciplina e controle destes projetos. Existem diversos relatos de sucesso de aplicação do Scrum em grandes companhias [Williams *et al*., 2011] e em projetos de grande porte [Jakobsen & Sutherland, 2009]. Nestes casos de sucesso foi realizada a aplicação de outras práticas de engenharia de software em conjunto com o arcabouço Scrum, evitando que aconteça o que é chamado de um *"Flaccid Scrum"* (termo forjado por Martin Fowler [Fowler, 2009] para descrever uma precária implantação do Scrum onde não há aplicação de nenhuma outra prática de engenharia de *software* para dar suporte ao arcabouço). Fowler também enfatiza que o Scrum não cita nenhuma técnica de engenharia, mas que não se deve ignorar a importância ou, muitas vezes, a necessidade destas técnicas. A ausência de práticas de engenharia no Scrum é proposital, e tem o objetivo dar poder de decisão ao time de desenvolvimento sobre quais técnicas utilizar, de acordo com as características específicas do projeto [scrummethodology, 2009]. Desta forma, o uso de práticas de engenharia para o controle e utilização do arcabouço com disciplina, fornecem aos projetos uma combinação de adaptabilidade e previsibilidade [Jakobsen & Sutherland, 2009] aumentando a possibilidade de sucesso desses projetos.

Uma das maneiras de se obter um nível desejável de controle é através de utilização de métricas e mensuração do projeto de desenvolvimento. Buscamos suprir esta necessidade através da apresentação de um plano de métricas compatível com os princípios ágeis de modo a obter controle e agilidade no decorrer destes projetos.

3. Metricas e métodos ágeis

Segundo Kan [Kan, 2002] as métricas de software podem ser classificadas em três categorias: processo, produto, e projeto. As métricas de processo têm por objetivo obter controle sobre o processo de desenvolvimento e monitorar como o trabalho vem sendo

feito como, por exemplo, a eficiência de remoção de defeitos, o tempo de resposta a defeitos, e a aderência ao processo. As métricas de produto têm relação com as características inerentes do produto desenvolvido. Exemplos englobam o tamanho do código, a complexidade, a densidade de defeitos entre outras. Já as métricas de projeto visam monitorar o projeto considerando atributos como o tamanho do time, produtividade, cronograma e esforço, de modo a manter o projeto controlado e dentro das expectativas.

Métricas de software em métodos ágeis vêm sendo estudadas recentemente [Kunz, Dumke & Schmietendorf, 2007] e sua importância é cada vez mais reconhecida. A utilização de métricas possibilita a análise concisa da situação atual do projeto, a tomada de decisão baseada em atributos concretos e a criação de um repositório de métricas para comparações com projetos futuros.

A qualidade do produto é um princípio considerado essencial ao Scrum e, devido a este fator, métricas de qualidade do produto começam a se tornar necessárias. Métricas de código para métodos ágeis foram estudadas [Sato, Goldman, & Kon, 2007], [Ambu *et al*., 2006] e consideradas como um bom indicador da qualidade do código**.** Já Kunz, Dumke, & Zenker [Kunz, Dumke & Zenker, 2008] propuseram uma ferramenta para a coleta de métricas de produto integrada à IDE de desenvolvimento Eclipse. Alshayeb & Li [Alshayeb & Li, 2005] propôs a métrica *System design instability* e Knoernschild [Knoernschild, 2006] propôs métricas de qualidade e design de código.

Quanto a métricas de processo, Jeffries propôs a *running tested features* [Jeffries, 2004], Jakobsen [Jakobsen & Sutherland, 2009] propôs o acompanhamento do tempo de resposta a uma *build* falha e Mahnic [Mahnic & Zabkar, 2008] propôs o número de impedimentos levantados durante as reuniões diárias.

Já no âmbito das métricas de projeto, Fuqua [Fuqua, 2003] realizou a avaliação do esforço através de pontos por função em projetos XP e considerou a técnica inadequada aos princípios ágeis de desenvolvimento. Por definição, o Scrum mede o esforço através da quantidade de horas de esforço restantes às tarefas e itens de backlog [Schwaber & Beedle, 2001]. Mike Cohn propôs o planejamento de esforço através de pontos de histórias de usuário e velocidade, e o monitoramento através de gráficos de *Burndown* [Cohn, 2005], práticas estas, que vêm se tornando comuns ao Scrum e a outros métodos ágeis. Outro aspecto bastante explorado no contexto de métricas de monitoração de projeto foram as métricas de valor agregado, que foram abordadas em [Sulaiman, Barton, & Blackburn, 2006], [Cabri & Griffiths, 2006], [Erdogmus, 2010] e [Mahnic & Zabkar, 2008].

Além das pesquisas em métricas de software, também existem pesquisas nas métricas de caráter econômico, como o *Real Option Thinking* apresentado por Racheva [Racheva & Daneva, 2008], o *Earned Business Value* [Rawsthorne, 2006], em complementação as técnicas *Return on Investment* e *Internal Rate of Return*, comumente utilizadas em métodos ágeis [Cohn, 2005].

Desenvolvimento de programas de métricas para o Scrum já foram feitos por [Oualid & Lévesque, 2010], [Jakobsen & Sutherland, 2009] e [Mahnic & Zabkar, 2008]. Oualid e Lévesque [Oualid & Lévesque, 2010] realizaram uma *survey* com integrantes de times de desenvolvedores Scrum e levantaram métricas que os desenvolvedores consideravam importantes. Neste trabalho se aprofundaram na resolução do *Technical Debt* (Custo que relacionado à revisão de aspectos técnicos do

sistema implementados de maneira não ideal - quanto mais complexa a solução, maior o custo de revisitação e, consequentemente, maiores os custos do projeto) do time de desenvolvimento e ressaltaram as dificuldades de definir e implementar um programa de métricas apropriado em um ambiente de métodos ágeis que pode, por vezes, ser bastante hostil em relação a práticas não ágeis.

Já [Jakobsen & Sutherland, 2009] focaram na melhoria do processo de desenvolvimento através da sugestão de métricas para mensurar características do processo, melhorando o comportamento organizacional e atingindo um maior nível de desempenho na execução do arcabouço Scrum na empresa Systematic. O trabalho de [Mahnic & Zabkar, 2008], por sua vez, foca na apresentação de um plano de métricas com o objetivo de monitoração de métricas de processo, produto e projeto. Baseado neste plano de métricas, eles ainda vão além e apresentam um modelo de um repositório de dados para o armazenamento e consulta destas métricas, o que é considerada uma boa prática ao se trabalhar com métricas de desenvolvimento [CMMI, 2010].O foco de trabalho da nossa proposta é a definição de um plano de métricas visando o monitoramento e controle de projetos de desenvolvimento, dando destaque às métricas de projeto e produtividade. Métricas de produto e processo também foram incluídas de modo a tornar o plano de métricas mais completo, mas não refletem o escopo deste trabalho.

A Tabela 1 separa os trabalhos de acordo com a classificação das métricas sobre as quais focam. Sendo estas divididas em métricas de processo, produto, projeto e econômicas.

Tabela 1 – Métricas em Projetos Ágeis

Métricas	Processo	Produto	Projeto	Econômicas
Sato, Goldman, & Kon, 2007	-	X	-	-
Ambu, Concas, Marchesi, & Pinna, 2006	-	X	-	-
Kunz, Dumke, & Zenker, 2008	-	X	-	-
Alshayeb & Li, 2005	-	X	-	-
Knoernschild, 2006	-	X	-	-
Jeffries, 2004	X	-	-	-
Jakobsen & Sutherland, 2009	X	-	-	-
Mahnic & Zabkar, 2008	X	-	X	-
Fuqua, 2003	-	-	X	-
Sulaiman, Barton, & Blackburn, 2006	-	-	X	-
Cabri & Griffiths, 2006	-	-	X	-
Erdogmus, 2010	-	-	X	-
Racheva & Daneva, 2008	-	-	-	X
Rawsthorne, 2006	-	-	-	X
Oualid & Lévesque, 2010	X	-	X	-
Scrum SPDW+	X	-	X	-

De acordo com a tabela, o trabalho mais similar ao nosso é a proposta de Mahnic e Zabkar [Mahnic & Zabkar, 2008], que também apresenta um plano de métricas para a monitoração de projetos que utilizam Scrum. Contudo, nos diferenciamos em alguns aspectos, como na monitoração através de valor agregado de Sulaiman [Sulaiman, Barton, & Blackburn, 2006], a não utilização de métricas de produto e a utilização das

métricas de processo [Jakobsen & Sutherland, 2009]. A necessidade da realização destas alterações será melhor explicada na seção 5 onde descrevemos mais detalhadamente a escolha das métricas pertencentes ao plano.

4. Solução

Para o desenvolvimento deste plano de métricas buscamos na bibliografia básica do arcabouço Scrum as métricas já utilizadas. Também buscamos seus princípios e valores para compreender as necessidades de tal tipo de projeto. Como passo seguinte, passamos a procurar em diferentes trabalhos científicos propostas de métricas para este tipo de projeto, selecionando métricas compatíveis com o nosso foco em monitoramento e acompanhamento de projeto (essencialmente métricas de projeto) e compatíveis com o arcabouço Scrum e heurísticas de métricas ágeis [Hartmann & Dymond, 2006]. Desta forma, agrupamos um conjunto de métricas que consideramos adequadas para o objetivo de acompanhamento de projetos Scrum, as quais são apresentadas nas tabelas 2, 3 e 4. A primeira demonstrando o conjunto de métricas primitivas juntamente com seu ponto de coleta, e as duas seguintes contendo as métricas derivadas de projeto e de processo respectivamente, e suas formulas de obtenção.

Todas as métricas devem ser apresentadas e preenchidas nas reuniões já estabelecidas pelo arcabouço Scrum de modo que a sua coleta seja realizada da maneira mais natural possível, buscando não impactar nas atividades do time de desenvolvimento.

Por padrão, a única métrica definida no Scrum para a realização do monitoramento do projeto é a quantidade de trabalho restante [Schwaber & Beedle, 2001]. Para a definição do plano de métricas, nos baseamos na descrição do arcabouço Scrum e também nas técnicas de estimativas de pontos de histórias e de velocidade, apresentadas por Cohn [Cohn, 2005]. A inclusão das métricas de histórias de usuário e de velocidade ocorreu devido às mesmas já serem práticas comuns ao Scrum utilizado no mercado. Para o monitoramento e análise de tempo e custo, utilizamos a abordagem *AgileEVM* proposta por Sulaiman [Sulaiman, Barton, & Blackburn, 2006] devido à análise de valor agregado ser uma técnica de grande poder informativo, tanto que, muitas vezes, acaba sendo uma exigência contratual, ou até mesmo a sua utilização é uma política organizacional. O *AgileEVM* apresenta a técnica de análise de valor agregado de maneira simples, sem ferir os princípios ou prejudicar a agilidade de projetos ágeis. Desta forma, as métricas necessárias para a utilização desta técnica foram adicionadas ao plano de métricas proposto (Tabela 2), de modo a viabilizar um monitoramento mais eficaz do projeto de desenvolvimento.

Tabela 2 - Plano de Métricas Scrum

Áreas de Qualidade	Métricas Derivadas	Métricas Diretas	Ponto Coleta
Esforço	**V** – Velocidade **TRT** – Trabalho restante para todas as atividades da Sprint **PRC** – Pontos de release completados até o momento	**TR** – Trabalho restante em cada tarefa do backlog de Sprint [Schwaber & Beedle, 2001]	Reuniões diárias
		TRA – Trabalho restante de atividades adicionado [Schwaber & Beedle, 2001]	Reuniões diárias
		TRR – Trabalho restante de atividades removido [Schwaber & Beedle, 2001]	Reuniões diárias
		PHPR – Pontos de história de usuário planejados para release	Planejamento de Release
		PHC – Pontos de história completados na sprint [Cohn, 2005]	Planejamento de Sprint
		PHA – Pontos de história adicionados [Cohn, 2005]	Planejamento de Sprint
		PHR – Pontos de história removidos [Cohn, 2005]	Planejamento de Sprint
Processo	**MTRI** – Média de tempo para resolução de impedimentos	**NI** – Nro impedimentos por Sprint/Release [Mahnic & Zabkar, 2008]	Reuniões diárias
		DAI – Data de abertura do impedimento [Mahnic & Zabkar, 2008]	Reuniões diárias
		DFI – Data de fechamento do impedimento [Mahnic & Zabkar, 2008]	Reuniões diárias
		NBS – Nro de Builds realizadas com sucesso [Jakobsen & Sutherland, 2009]	Reuniões diárias
Tempo	**DER** – Data da entrega da Release	**TS** – Tamanho da Sprint [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Release
		NP– Nro de Sprints planejadas [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Release
		DIR – Data de início da Release [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Release
		NS – Nro da Sprint atual [Sulaiman, Barton, & Blackburn, 2006]	PS
Time		**NIT** – Nro. de integrantes do time [Mahnic & Zabkar, 2008]	Planejamento de Release
		DT – Percentual de dedicação do time ao projeto [Mahnic & Zabkar, 2008]	Planejamento de Release
Custo	**TCR** – Percentual de trabalho completado real **TCP** – Percentual de trabalho completado planejado **VPA** – Variação de prazo agregada **VCA** – Variação de custo agregada **VA** – Valor agregado **VP** – Valor Planejado **PC** – Percentual planejado completado **IDC** – Indice de desempenho de custo **IDP** – Índice de desempenho prazo	**CPR –** Custo planejado para a Release [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Release
		CR – Custo real da Release [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Release
		CS – Custo real da Sprint anterior [Sulaiman, Barton, & Blackburn, 2006]	Planejamento de Sprint

Tabela 2 - Plano de Métricas Scrum (continuação)

Áreas de Qualidade	Métricas Derivadas	Métricas Diretas	Ponto Coleta
Qualidade	**MRDI** – Média de tempo para resolução de defeitos internos **MRDE** – Média de tempo para resolução de defeitos externos **ERD** – Eficiência de remoção de defeitos	**NDI** – Nro defeitos internos [Mahnic & Zabkar, 2008]	Reuniões diárias
		NDE – Nro defeitos externos [Mahnic & Zabkar, 2008]	Reuniões diárias
		DADI – Data de abertura do defeito [Mahnic & Zabkar, 2008]interno	Reuniões diárias
		DFDI – Data de fechamento do Defeito interno [Mahnic & Zabkar, 2008]	Reuniões diárias
		DADE – Data de abertura do defeito externo [Mahnic & Zabkar, 2008]	Reuniões diárias
		DFDE – Data de fechamento do defeito externo [Mahnic & Zabkar, 2008]	Reuniões diárias

As métricas são divididas em diretas ou derivadas. As primeiras sendo atributos coletados em diferentes pontos do processo: planejamento de *release*, planejamento de *Sprint* e reuniões diárias. As derivadas, por sua vez, são calculadas a partir das métricas diretas com o intuito de obter um diferente nível de informação. Dentro da tabela 2 as métricas estão divididas em diferentes áreas de qualidade:

- Tempo – Área fundamental para o monitoramento do projeto através da análise de valor agregado. Em métodos ágeis ao invés de considerar datas de início e fim de atividades do cronograma, o cálculo é realizado através da avaliação da quantidade de *Sprint*s planejadas e a duração fixada para cada *Sprint*.
- Esforço – Nesta área o programa de métricas é dividido em dois diferentes níveis. No primeiro o esforço é monitorado através da avaliação do trabalho restante nas tarefas da *Sprint*, realizado através da mensuração da quantidade de itens de *Backlog* de *Sprint* e a atualização diária do trabalho restante para cada um dos itens. Já em nível de Release o esforço é monitorado a partir da noção de pontos de histórias de usuário planejados para a *release* e a velocidade do time de desenvolvimento.
- Custo – Métricas de custo são essenciais à monitoração de valor agregado do projeto. Esta monitoração será realizada em nível de Release e envolve toda a análise das métricas de valor agregado, variação de custo, cronograma e os índices de desempenho clássicos a esta técnica.
- Processo – Métricas relacionadas ao monitoramento das práticas processuais. Estas métricas têm por objetivo avaliar se o processo de desenvolvimento adotado está sendo devidamente utilizado.
- Qualidade – Indicadores da qualidade são coletados por motivos de registro e referência histórica. Estes atributos refletem a qualidade do produto que, apesar de importantes, não desempenham um papel fundamental no monitoramento e controle do andamento do projeto.
- Time – Métricas indicadoras do time de desenvolvimento. Usualmente times de desenvolvimento Scrum são pequenos (7-10) e se busca alcançar dedicação exclusiva ao projeto por parte dos integrantes [Cohn, 2005]. Estas métricas têm por objetivo avaliar estas características do arcabouço Scrum, porém não desempenham um papel no acompanhamento do projeto.

Na **Error! Reference source not found.** são descritas as métricas derivadas com foco no monitoramento do projeto.

Tabela 3 - Métricas Derivadas (Projeto)

Métricas Derivadas	Objetivos	Equação
Percentual de trabalho completado real (**TCR**)	Medir o percentual do trabalho completado até o momento baseado nas histórias planejadas e completadas.	TCR = PHC/PHPR
Percentual de trabalho completado planejado **(TCP)**	Medir o progresso estimado para o projeto até o momento	TCP = NS/NP
Valor Agregado **(VA)**	Medir o valor agregado do projeto até o momento.	VA = TCR*CPR
Valor Planejado **(VP)**	Medir o valor que o projeto tinha planejado entregar até o momento.	VP = TCP*CPR
Variação de prazo agregada **(VPA)**	Fornecer a relação entre a estimativa de prazo original e o percentual atual de trabalho completado até a data de status.	VPA=VA-VP
Variação de custo agregada **(VCA)**	Fornecer a relação entre o custo original planejado e o percentual de trabalho completado até a data de status.	VCA = VA - CR
Índice de desempenho de custo **(IDC)**	Fornecer a relação entre os custos consumidos pelo projeto e o valor agregado.	IDC = VA/CR
Índice de desempenho de prazo **(IDP)**	Fornecer a taxa de conversão do valor estimado em valor agregado	IDP = VA/VP
Trabalho restante para todas as atividades da Sprint **(TRT)**	Mede o total de trabalho restante para a *Sprint* atual	$\mathrm{TRT} = \sum_{k=1}^{n} \mathrm{TR_k}$ Onde n > 0
Velocidade **(V)**	Mede a capacidade de implementação de histórias do time, utilizada para a realização dos cálculos de data de entrega.	$\mathrm{V} = \frac{1}{n}\sum_{k=1}^{n} \mathrm{V_k}$ Onde n > 0
Pontos de *Release* completados **(PRC)**	Mede o total de pontos de histórias restantes para a *Release* atual	$\mathrm{PRC} = \sum_{k=1}^{n} \mathrm{PHC_k}$ Onde n > 0
Média de tempo para a resolução de impedimentos **(MTRI)**	Tem o objetivo de avaliar o tempo médio que se leva para resolver os impedimentos relatados pelo time de desenvolvimento	$MTRI = \frac{\sum(DAI - DFI)k}{NI}$

Já na Tabela 4 estão as métricas derivadas de processo e qualidade do produto. Para cada uma delas, estão descritas as métricas primitivas das quais estas métricas se originam, seu propósito e a equação correspondente.

Tabela 4 - Métricas Derivadas (Processo e Produto)

Métricas Derivadas	Objetivos	Equação
Média de tempo para a resolução de Defeitos Externos **(MRDE)**	Tem o objetivo de avaliar o tempo médio que se leva para resolver os defeitos internos relatados pelo time de desenvolvimento	$MRDE = \frac{\sum(DFDE - DADE)k}{NDE}$
Média de tempo para a resolução de Defeitos Internos **(MRDI)**	Tem o objetivo de avaliar o tempo médio que se leva para resolver os defeitos externos relatados pelo time de desenvolvimento	$MRDI = \frac{\sum(DFDI - DADI)k}{NDI}$
Eficiência para a remoção de defeitos **(ERD)**	Tem o objetivo de avaliar a capacidade do time de desenvolvimento de encontrar os defeitos antes de o produto chegar ao cliente	$ERD = \frac{\text{NDI}}{\text{NDE} + \text{NDI}}$
Data de entrega da *Release* **(DER)**	Dado o progresso do time até o momento tem o objetivo de proporcionar uma possível data de entrega da release baseado no progresso do time até o momento	$DER = DIR + TS \cdot \left(\frac{NS}{TCR}\right)$

O monitoramento do projeto ocorre em dois diferentes estágios, o monitoramento da *sprint* e o monitoramento da *release*. No decorrer da *Sprint* o esforço despendido nas tarefas é monitorado através da quantidade de trabalho restante. Este valor já é atualizado diariamente, por padrão, nos *Daily Scrum*, e visualizado através de gráficos de *Burndown*. Já em nível de planejamento de *Release* é aplicada a técnica *AgileEVM*, avaliando não mais a granularidade de esforço relacionado às atividades, e sim a quantidade de histórias de usuários completadas, número de *Sprint*s realizadas e a velocidade média de implementação do time. Baseado nestes dois níveis de monitoramento, a análise de progresso pode ser tanto em nível de *Sprint*, através de gráficos de *burndown* de atividades, quanto em nível de release através de gráficos de *burndown* de histórias em conjunto com os índices de desempenho de custo e cronograma calculados através do valor agregado.

Quando comparando este plano de métricas ao de Mahnic e Zabkar [Mahnic & Zabkar, 2008], ressaltamos as seguintes diferenças. Métricas de tamanho não foram utilizadas em virtude que as mesmas não serem definidas, por padrão, no Scrum. Usualmente, métodos ágeis utilizam a noção de pontos de história como a medida de tamanho e esforço que, apesar de compartilhar características com técnicas como Pontos de função e pontos de caso de uso, é uma métrica de tamanho relativo e não pode ser utilizada para a comparação com outros projetos. Uma métrica física que poderia ser utilizada é a contagem de linhas de código. Entretanto, esta não é uma prática recomendada [Kan, 2002] em virtude da grande variação entre plataformas e por esta não levar em consideração a complexidade da funcionalidade sendo desenvolvida. Desta forma, optamos por não utilizar o tamanho físico do código como uma métrica a ser analisada por padrão e sim como uma opção, caso se julgue necessário, devido a políticas organizacionais ou mensuração de atributos relacionados ao produto, como a densidade de defeitos.

As métricas qualitativas baseadas em *surveys*, apesar de úteis para mensurar atributos como a satisfação do time e cliente, não têm foco na monitoração e controle do projeto. Além disso, elas apresentam um nível de envolvimento do time com as atividades de formulação, aplicação e avaliação da *survey*. Este formato de métrica não

é adequado para a utilização no ambiente *SPDW+* baseado em um processo de ETC automatizado.

Mahnic e Zabkar [Mahnic & Zabkar, 2008] também propõem realizar o monitoramento do projeto através da análise de valor agregado. Contudo, optamos pela utilização da técnica proposta por Sulaiman [Sulaiman, Barton, & Blackburn, 2006] em virtude de a mesma medir um nível acima [Hartmann & Dymond, 2006], monitorando o projeto em nível de *release*, e utilizando técnicas bastante comuns no cenário ágil de desenvolvimento, como a velocidade e as histórias de usuários. Além disso, a abordagem de Mahnic e Zabkar para avaliação do valor agregado implica na mensuração do esforço realizado em cada uma das tarefas, o que é uma prática considerada inadequada para o Scrum, como descrito por Schwaber [Schwaber & Beedle, 2001]. :

> *"Relatórios de trabalho restante atulizam o número estimado de horas necessárias para completar uma tarefa. Isso não deve ser confundido com relatórios de tempo, que não são parte do Scrum. Não existem mecânismos no Scrum para monitorar a quantidade de tempo que um time trabalha. Times são mensurados através do cumprimento de objetivos, e não através da quantidade de horas que eles levam até cumprir o objetivo. Scrum é orientado a resultados, não à processos."*

5. Considerações finais

Considerando a importância do monitoramento e controle de projetos e usual utilização de métricas com este objetivo, métodos ágeis de desenvolvimento também podem começar a avaliar a utilização destas técnicas para dar suporte ao seu processo. Neste trabalho foi definido um plano de métricas para o arcabouço Scrum, de modo a obter um nível de controle satisfatório, e buscando um menor envolvimento do time através da coleta de métricas simples em reuniões já especificadas no Scrum.

Contudo, o principal objetivo desta pesquisa é a utilização deste plano de métricas para a definição de um ambiente de DW e com um processo de ETC automatizado [Silveira, Becker, & Ruiz, 2010]. Este ambiente de gestão de métricas de projeto visa reduzir ao mínimo indispensável o envolvimento do time na manipulação e análise das métricas. Espera-se torná-lo, desta forma, ideal para utilização em conjunto com métodos ágeis de desenvolvimento, obtendo toda a agilidade necessária para a execução do processo de monitoramento e controle do projeto. Como passo seguinte, realizaremos a adaptação deste ambiente de DW em conjunto com o plano de métricas definido neste artigo e buscaremos a sua utilização em uma empresa desenvolvedora de modo que uma avaliação sobre o ambiente possa ser realizada, buscando a comprovação dos benefícios da utilização desta abordagem em um ambiente real de desenvolvimento.

6. Referencias

Alshayeb, M., & Li, W. (2005). An Empirical Study of System Design Instability Metric and Design Evolution in an Agile Software Process. *Journal of Systems and Software* , 269 - 274.

Ambu, W., Concas, G., Marchesi, M., & Pinna, S. (2006). Studying the Evolution of Quality Metrics in an Agile/Distributed Project. In: W. Ambu, *Extreme*

Programming and Agile Processes in Software Engineering (pp. 85-93). Springer Berlin / Heidelberg.

Cabri, A., & Griffiths, M. (2006). Earned Value and Agile Reporting. *AGILE '06 Proceedings of the conference on AGILE 2006* (pp. 17-22). IEEE Computer Society Washington, DC, USA ©2006 .

CMMI. (2006). *CMMI for Development, Version 1.2.* Software Engineering Institute, Carnegie Mellon University.

Cohn, M. (2005). *Agile Estimating and Planning.* Prentice Hall PTR Upper Saddle River, NJ, USA ©2005 .

Erdogmus, H. (2010). Tracking progress through earned value. *IEEE Software*, 2-7.

Fowler, M. (30 de January de 2009). *Flaccid Scrum*. Fonte: martinfowler.com: http://martinfowler.com/bliki/FlaccidScrum.html

Fuqua, A. M. (2003). Using function points in XP – considerations. *XP'03 Proceedings of the 4th international conference on Extreme programming and agile processes in software engineering* (pp. 340-342). Springer-Verlag Berlin, Heidelberg ©2003 .

Hartmann, D., & Dymond, R. (2006). Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value. *AGILE '06 Proceedings of the conference on AGILE 2006* (pp. 126 - 134). IEEE Computer Society Washington, DC, USA ©2006 .

Jakobsen, C. R., & Johnson, K. A. (2008). Mature Agile with a Twist of CMMI. *AGILE '08 Proceedings of the Agile 2008* (pp. 212-217). IEEE Computer Society Washington, DC, USA ©2008.

Jakobsen, C. R., & Sutherland, J. (2009). Scrum and CMMI – Going from Good to Great. *Agile Conference, 2009. AGILE '09.* (pp. 333 - 337). IEEE Computer Society Washington, DC, USA ©2009.

Jeffries, R. (14 de June de 2004). *A Metric Leading to Agility*. Fonte: XP Programming: http://xprogramming.com/articles/jatrtsmetric/

Kan, S. H. (2002). *Metrics And Models In Software Quality Engineering.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2002 .

Knoernschild, K. (07 de 06 de 2006). *Using Metrics To Help Drive Agile Software*. Fonte: Agile Journal: http://www.agilejournal.com/articles/columns/the-agile-developer/56-using-metrics-to-help-drive-agile-software

Kunz, M., Dumke, R. R., & Zenker, N. (2008). Software Metrics for Agile Software Development. *ASWEC '08 Proceedings of the 19th Australian Conference on Software Engineering* (pp. 673-678). IEEE Computer Society Washington, DC, USA ©2008.

Kunz, M., Dumke, R., & Schmietendorf, A. (2007). How to Measure Agile Software Development. In: *Software Process and Product Measurement* (pp. 95 - 101). Springer-Verlag Berlin, Heidelberg ©2008.

Mahnic, V., & Zabkar, N. (2008). Measurement repository for Scrum-based software development process. *CEA'08 Proceedings of the 2nd WSEAS International*

Conference on Computer Engineering and Applications (pp. 23-28). Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS).

Oualid, K., & Lévesque, G. (2010). Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want? *C3S2E '10 Proceedings of the Third C* Conference on Computer Science and Software Engineering* (pp. 101-107). ACM New York, NY, USA ©2010 .

Racheva, Z., & Daneva, M. (2008). Using measurements to support real-option thinking in agile software development. *APOS '08 Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral* (pp. 15-18). ACM New York, NY, USA ©2008 .

Rawsthorne, D. (07 de June de 2006). *Calculating Earned Business Value For An Agile Project.* Acesso em 14 de June de 2012, disponível em The Agile Journal: http://www.agilejournal.com/index2.php?option=com_content&do_pdf=1&id=54

Sato, D., Goldman, A., & Kon, F. (2007). Tracking the evolution of object-oriented quality metrics on agile projects. *XP'07 Proceedings of the 8th international conference on Agile processes in software engineering and extreme programming* (pp. 84-92). Springer-Verlag Berlin, Heidelberg ©2007 .

Schwaber, K. (2004). *Agile Project Management With Scrum.* Microsoft Press Redmond, WA, USA ©2004.

Schwaber, K., & Beedle, M. (2001). *Agile Software Development with Scrum.* Prentice Hall PTR Upper Saddle River, NJ, USA ©2001.

scrummethodology. (04 de June de 2009). *Flaccid Scrum?* Acesso em 14 de June de 2012, disponível em Scrum Methodology: http://scrummethodology.com/flaccid-scrum/

Silveira, P. S., Becker, K., & Ruiz, D. D. (2010). SPDW+: a seamless approach for capturing quality metrics in software development environments. *Software Quality Journal*, 227-268.

Sommerville, I. (2011). *Software Engineering (9th edition).* Pearson Addison Wesley.

Sulaiman, T., Barton, B., & Blackburn, T. (2006). AgileEVM - Earned Value Management in Scrum Projects. *AGILE '06 Proceedings of the conference on AGILE 2006* (pp. 7 - 16). IEEE Computer Society Washington, DC, USA ©2006 .

Torgeir Dingsøyr, S. N. (2012). A decade of agile methodologies: Towards explaining agile software development. *J. Syst. Softw. 85, 6* , 1213-1221.

Williams, L., Brown, G., Meltzer, A., & Nagappan, N. (2011). Scrum + Engineering Practices: Experiences of three Microsoft Teams. *ESEM '11 Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement* (pp. 463-471). IEEE Computer Society Washington, DC, USA ©2011 .