Avaliando a Qualidade do Software de um Projeto Distribuído: um Relato de Experiência

Crescencio Rodrigues Lima Neto[1], Emmanuel B. Carvalho[1], Silvio R. L. Meira[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (CIn/UFPE)

{crln,ebc2,srlm}@cin.ufpe.br

Resumo. *Muitas empresas estão implementando o desenvolvimento distribuído de software e agregados aos benefícios existem grandes desafios como a dificuldade de gerenciamento da equipe e problemas na comunicação. Como solução, é preciso adaptar a metodologia tradicional de desenvolvimento de software. Neste contexto, este trabalho relata como a qualidade foi mantida durante um projeto de desenvolvimento distribuído. Por este motivo, um estudo de caso foi realizado com o intuito de investigar como as boas práticas do Scrum foram adaptadas. Ao final do projeto, um conjunto de lições aprendidas foi identificado para manter a qualidade no desenvolvimento distribuído de software de forma prática e eficiente.*

Abstract. *Several companies are implementing distributed software development and with the benefits come great challenges such as the difficulty of team management and communication problems. As a solution, we need to adapt the traditional methodology of software development. In this context, this paper describes how the quality was maintained during a distributed development project. For this reason, a case study was conducted in order to investigate how the Scrum best practices were adapted. At the end of the project, a set of lessons learned was identified to maintain the quality of distributed software development in a practical and efficient way.*

1. Introdução

A globalização e o desenvolvimento tecnológico contribuíram para eliminação da barreira geográfica, permitindo assim que times localizados a quilômetros de distância interajam como se estivessem no mesmo ambiente [Bosnic et al. 2011]. Neste contexto, destaca-se o surgimento do Desenvolvimento Distribuído de Software (DDS).

A grande demanda por DDS está sendo vivenciada por várias organizações nos últimos anos [Binder 2007]. Entretanto, agregados aos benefícios existem grandes desafios. Dentre eles, estão as dificuldades em gerenciar a equipe, problemas na comunicação e diferenças culturais [Olson and Olson 2000].

A distribuição do processo de desenvolvimento demanda a adaptação da metodologia tradicional [Audy and Prikladnicki 2007]. As mudanças implicam em permitir avaliações constantes do planejamento, possibilitar a redução do *time to market* e evitar desperdício de esforço, visando sempre a melhoria da qualidade. Por estes motivos, a adoção de ferramentas de apoio a metodologia ágeis está se tornando cada vez mais constante [Cristal et al. 2008].

Este trabalho relata um estudo de caso que tenta avaliar como a qualidade do software pode ser mantida durante um projeto de desenvolvimento distribuído através da adaptação das boas práticas da metodologia *Scrum* [Schwaber and Beedle 2001].

Este trabalho está organizado da seguinte maneira: a seção 2 apresenta os trabalhos relacionados. A seção 3 aborda a metodologia utilizada no projeto. A Seção 4 exibe o estudo de caso sobre o desenvolvimento e validação do projeto. Na seção 5 as lições aprendidas são apresentadas, e finalmente, a seção 6 apresenta as considerações finais e perspectivas de trabalhos futuros.

2. Trabalhos Relacionados

[Soares et al. 2007] descreve uma aplicação do *Scrum* em um processo de desenvolvimento de uma fábrica de software *open source* que possui características de desenvolvimento distribuído. [Cavalcanti et al. 2009] apresenta uma aplicação *open source*, desenvolvida para facilitar a adoção do *Scrum* por times distribuídos de forma a apoiar o planejamento e controle de projetos.

[Chalegre et al. 2010] e [Júnior et al. 2011] descrevem os resultados do processo utilizado para construção de forma distribuída considerando os principais problemas encontrados e a forma como foram resolvidos, desde a implementação até a comunicação com o usuário final.

Os trabalhos não relatam como a avaliação da qualidade foi feita, nem exploram como o processo de testes foi realizado. Nenhum dos artigos verificou como os testes influenciam na verificação da qualidade do software produzido através do desenvolvimento distribuído. Neste artigo, discute-se como a validação foi executada e gerenciada durante um projeto distribuído através da adaptação de técnicas e boas práticas do *Scrum*.

3. Metodologia Utilizada no Projeto

Uma equipe de 62 alunos de mestrado e doutorado foi dividida em seis grupos compostos por no máximo 10 pessoas. Para cada equipe foi definido um *Scrum Master* [Schwaber and Beedle 2001] que seria o responsável por facilitar a comunicação e coordenar as atividades de cada grupo.

O desenvolvimento do projeto ocorreu de forma distribuída. Não existia um ambiente de trabalho comum. Semanalmente, uma reunião presencial de acompanhamento era realizada com pelo menos um integrante de cada grupo. Nestas reuniões eram abordadas as dificuldades e impedimentos que estavam acontecendo.

O *Product Owner (PO)* [Schwaber and Beedle 2001] foi responsável por escolher os objetivos do desenvolvimento, definir o valor de negócio de cada funcionalidade do produto e decidir junto aos grupos quais requisitos deveriam ser primeiro atendidos. Ele foi considerado parte da equipe e estava sempre em contato com os integrantes do grupo.

Os grupos trabalharam no desenvolvimento da *Firescrum* [Cavalcanti et al. 2009] que é uma ferramenta *open source* desenvolvida para oferecer suporte ao gerenciamento de projetos que utilizem a metodologia *Scrum*. A aplicação foi construída com o intuito de auxiliar pessoas, gerentes e times na concretização das suas metas.

Cada equipe ficou responsável por um módulo da ferramenta. No total seis módulos foram implementados: *Task Board*, *Test Module*, *Bug Tracking*, *Desktop Agent*, *Planning Poker* e o *Core module* (constituído pelas funcionalidades essenciais da ferramenta, todos os demais módulos são conectados a ele).

4. Estudo de Caso

O objetivo do estudo foi investigar de forma exploratória como seria possível garantir a qualidade do software implementado em ambientes de desenvolvimento distribuído. Este estudo de caso relata as experiências vividas pela equipe responsável pelo desenvolvimento do módulo *Core* da aplicação.

A metodologia de pesquisa exploratória foi utilizada para tentar identificar o que estava acontecendo, procurando por novos insights e gerando ideias e hipóteses para nova pesquisa [Runeson and Host 2009]. Como técnica de coleta de dados foi utilizado o método de terceiro grau, ou seja, uma análise independente dos artefatos gerados foi realizada [Lethbridge et al. 2005].

A primeira dificuldade encontrada no início do projeto foi com relação ao funcionamento da metodologia *Scrum*. A maioria dos integrantes dos times consideravam o papel do gerente (ou líder técnico) fundamental para o funcionamento da equipe. Este fator ocasionou em uma falha na interpretação com relação ao papel do *Scrum Master*, que no modelo do *Scrum* nada mais é do que um integrante do grupo que auxilia no processo de comunicação e solução de impedimentos [Schwaber and Beedle 2001].

A oportunidade de tomar decisões e trabalhar de forma distribuída parecia um tanto absurda para os participantes [Olson and Olson 2000], o que levou a hipótese de fracasso do projeto. Porém, a medida que a ferramenta evoluía e os integrantes conheciam melhor o sistema de trabalho essas dificuldades foram superadas.

4.1. Planejamento, Acompanhamento e Implementação

Diversos aspectos do *Scrum* foram utilizados durante o desenvolvimento. Como os grupos não possuíam um local de trabalho em comum, algumas práticas foram adaptadas para permitir que o desenvolvimento distribuído fosse implementado de forma eficiente.

Inicialmente os requisitos do sistema foram definidos com o *Product Owner*. Todos os itens a serem desenvolvidos receberam um *Business Value* (valor de negócio) [Schwaber and Beedle 2001] que era considerado para definição da prioridade de implementação.

O ciclo de desenvolvimento foi dividido em *Sprints* [Schwaber and Beedle 2001] que agendavam a entrega das novas funcionalidades do produto. Eles aconteciam em intervalos de quinze dias entre uma iteração e outra. Este período de encontros quinzenais foi escolhido para facilitar o acompanhamento do projeto. Cada Sprint possuía um objetivo que definia quais funcionalidades deveriam ser entregue.

No início de cada iteração era realizada uma reunião de planejamento. Através de uma seção de *Planning Poker* [Schwaber and Beedle 2001], o esforço de desenvolvimento de cada item era debatido. Em seguida era realizada uma negociação com o *PO*, via email ou lista de discussão, visando definir o objetivo da *Sprint*. Por fim, cada membro da equipe escolhia suas atividades.

As tarefas eram registrados através de sistemas de compartilhamento online. Os documentos eram constantemente atualizados com o nome do responsável e o status de cada atividade. O que permitia uma boa visualização do andamento geral da *Sprint*. As falhas encontradas eram registrados em uma ferramenta de *bug tracking*.

Reuniões diárias eram realizadas através de ferramentas de comunicação online. Dessa forma, todos do time sabiam como estava o andamento das atividades e os impedimentos que apareceram até aquele momento. O *Scrum Master* era o responsável por ajudar na resolução dos impedimentos levantados durante a reunião.

Como a maioria dos envolvidos moravam em cidades diferentes, o paradigma do desenvolvimento presencial precisou ser quebrado rapidamente. Porém, o início desta etapa foi marcado por várias reuniões presenciais para nivelar o conhecimento dos integrantes. O *pair-programming* [Schwaber and Beedle 2001] permitiu que os desenvolvedores mais experientes compartilhassem o conhecimento com os demais.

A fim de possibilitar o desenvolvimento distribuído entre as equipes *branches* de desenvolvimento foram criados para cada uma e em marcos específicos era realizada a sincronização com o *branch* principal. Um integrante de cada time era responsável por esta tarefa.

Os desenvolvedores eram responsáveis por verificar os requisitos da tarefa, implementar, executar testes unitários e manter o código sempre sincronizado com o *branch* de desenvolvimento minimizando a possibilidade de perda de código.

4.1. Avaliação da Qualidade

O processo de testes de software não foi apenas voltado para detecção e verificação das correções das falhas, ele foi importante para a garantia da qualidade e otimização do esforço empregado [Collard and Burnstein 2002]. Um dos desafios da equipe foi garantir um código estável para o *Core* da aplicação, garantindo assim, a conectividade junto aos módulos dos outros times do projeto.

Durante o processo de desenvolvimento do projeto, a equipe realizou testes unitários e testes de sistema através de técnicas de caixa preta. Testes exploratórios também foram executados a fim de identificar problemas além dos ciclos de teste [Collard and Burnstein 2002]. Entretanto, outro grande desafio para equipe de testes foi lidar com a distribuição dos seus integrantes o que dificultou a sincronização e monitoramento das atividades.

4.2.1. Análise e Planejamento

A estratégia definida pelo time *Core* consistiu em trabalhar sobre aspectos críticos da ferramenta definindo uma equipe de testes que atuou de forma assíncrona, distribuída e com níveis de conhecimento distintos sobre engenharia de teste.

O documento de casos de teste foi criado juntamente com o documento de requisitos e foi evoluindo assim que as funcionalidades eram desenvolvidas. Testes unitários eram criados em paralelo com a implementação.

Além disso, os outros times que interagiam com as funcionalidades implementadas pelo *Core* também cadastravam falhas. A partir deste momento surgiu a necessidade de manter o monitoramento sobre a duplicidade de falhas.

Para contemplar um canal eficiente de comunicação compartilhado por todas as equipes do projeto, três artefatos foram criados: workflow do ciclo de testes, documento de casos de teste e um modelo para registrar as falhas encontradas.

A Figura 1 apresenta o ciclo de atividades de testes que era composto por: 1) Novos testes eram criados validando as principais funcionalidades. Eles populavam o ciclo de teste; 2) Os testes eram executados; 3) As falhas encontradas eram registradas na ferramentas de *bug tracking* conforme o modelo; 4) As falhas corrigidas eram verificadas e 5) Durante a validação das falhas novos testes eram acrescentados ao ciclo.

Figura 1. Workflow do ciclo de testes

Novos testes eram adicionados no documento de casos de teste assim que novas funcionalidades eram criadas. O ciclo era executado duas vezes no decorrer das *Sprints*. A equipe de testes era composta por testadores responsáveis pela execução e criação dos testes. O registro das falhas era feito por testadores e desenvolvedores.

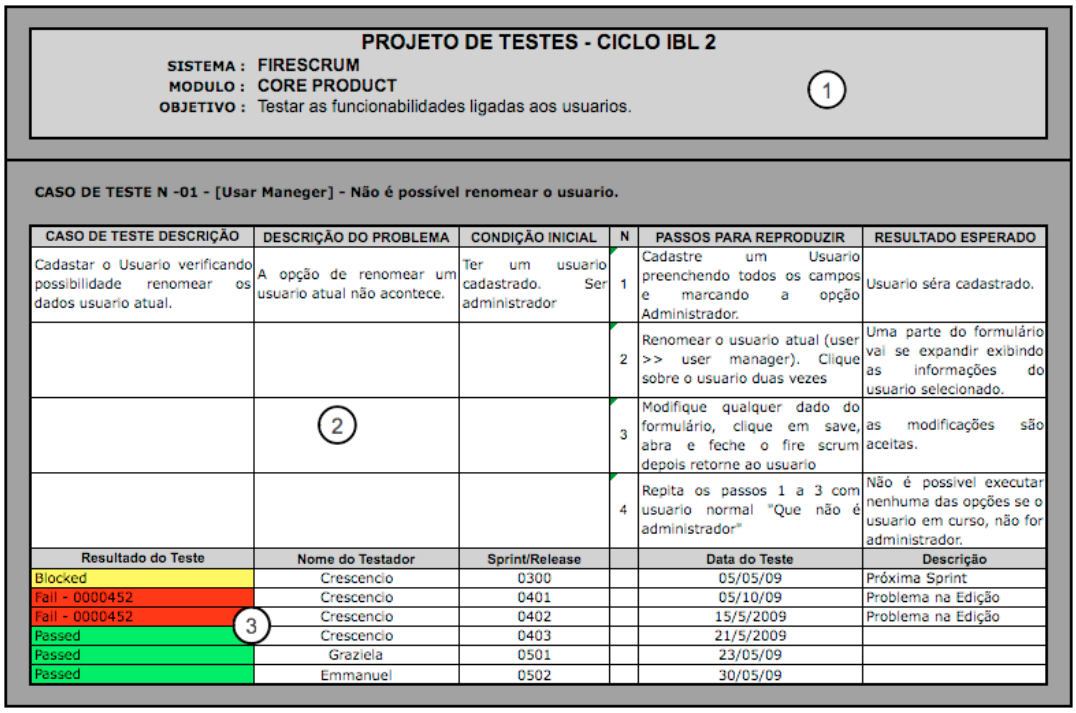
A Figura 2 apresenta o modelo utilizado para criação de casos de teste e registro dos dados da execução do plano de teste. As informações do projeto (modulo que está sendo testado e objetivo do conjunto de testes) são exibidas no topo (1), em seguida os dados necessários a execução do teste (breve descrição do caso de teste e do problema, condições iniciais para execução do caso de teste, os passos que deverão ser seguidos e resultados esperados) são mostrados (2).

Logo abaixo das informações referentes a execução dos casos de teste estão associados os resultados de cada execução (3). Esta conexão de dados foi importante para auxiliar no mapeamento da execução e manutenção dos casos de teste. Como os ciclos foram executados de forma dinâmica, as alterações no documento poderiam ser realizadas a medida que os testes eram executados.

Fatores que impediam a execução dos casos de teste eram considerados como bloqueadores e recebiam a identificação *"Blocked"*. Por exemplo, quando uma determinada funcionalidade prevista não estava implementada durante o ciclo, problemas no ambiente, etc. Quando uma falha era identificada o teste recebia a identificação *"Fail"* e o número da solicitação de mudança era associado para verificação nos próximos ciclos [Keller 2005]. Após a correção era feita uma verificação antes do fechamento da solicitação de mudança. Todos os testes que eram executados com sucesso recebiam a identificação *"Passed"*. Além do resultado dos testes, dados como: nome do testador, número da Sprint, data de execução e descrição de impedimentos também foram armazenados.

4.2.1. Execução do Plano de Testes

A execução do plano de teste era sempre realizada por três atores. Os participantes inexperientes receberam treinamento sobre o processo de execução de testes quanto ao conceito de identificação de falhas, classificações atribuídas ao estado de um caso de teste, utilização dos artefatos e como cadastrar uma solicitação de mudança.



PROJETO DE TESTES - CICLO IBL 2

SISTEMA : FIRESCRUM
MODULO : CORE PRODUCT
OBJETIVO : Testar as funcionabilidades ligadas aos usuarios.

1

CASO DE TESTE N -01 - [Usar Maneger] - Não é possível renomear o usuario.

CASO DE TESTE DESCRIÇÃO	DESCRIÇÃO DO PROBLEMA	CONDIÇÃO INICIAL	N	PASSOS PARA REPRODUZIR	RESULTADO ESPERADO
Cadastar o Usuario verificando possibilidade renomear os dados usuario atual.	A opção de renomear um usuario atual não acontece.	Ter um usuario cadastrado. Ser administrador	1	Cadastre um Usuario preenchendo todos os campos e marcando a opção Administrador.	Usuario séra cadastrado.
			2	Renomear o usuario atual (user >> user manager). Clique sobre o usuario duas vezes	Uma parte do formulário vai se expandir exibindo as informações do usuario selecionado.
	2		3	Modifique qualquer dado do formulário, clique em save, abra e feche o fire scrum depois retorne ao usuario	as modificações são aceitas.
			4	Repita os passos 1 a 3 com usuario normal "Que não é administrador"	Não é possivel executar nenhuma das opções se o usuario em curso, não for administrador.

Resultado do Teste	Nome do Testador	Sprint/Release		Data do Teste	Descrição
Blocked	Crescencio	0300		05/05/09	Próxima Sprint
Fail - 0000452	Crescencio	0401		05/10/09	Problema na Edição
Fail - 0000452	Crescencio	0402		15/5/2009	Problema na Edição
Passed	Crescencio	0403		21/5/2009	
Passed	Graziela	0501		23/05/09	
Passed	Emmanuel	0502		30/05/09	

3

Figura 2. Exemplo de Caso de Teste

Um problema que impactou no processo de teste foi observado quando os demais times do projeto não sincronizavam suas versões com o repositório central nos períodos pré-estabelecidos. Isso acarretava na perda da validação de algumas falhas registradas e eventualmente, novas falhas eram inseridas no código final após a rodada de testes. Para solucionar este problema foi definido um marco para sincronização aonde era aplicado um rótulo no código. Novos ciclos de testes eram executados a partir deste momento.

Um modelo para cadastro de solicitação de mudança foi criado pelo grupo para padronizar todos os registros. Modelo este que depois passou a ser utilizado por todas as equipes do projeto. Para cada solicitação de mudança finalizada pelos desenvolvedores, um engenheiro de teste era indicado para testar e verificar se aquela solicitação foi corretamente solucionada. O sistema de notificação da ferramenta de *bug tracking* era utilizado para alertar a todos quando uma solicitação era finalizada e verificada.

5. Lições Aprendidas

Ao final do projeto as seguintes lições aprendidas foram observadas:

1. Foi possível para equipe de testes realizar o *Daily Scrum Meeting* de forma não presencial. O uso de ferramentas de comunicação online garantiu a realização das reuniões diárias;
2. A solução encontrada para mitigar o problema da não objetividade durante as reuniões diárias foi o uso de comunicação prévia através de email. Possíveis soluções para os problemas eram levantados antes das reuniões. As tomadas de decisão eram realizadas durante o encontro;

3. Os responsáveis pelas atividades de testes que possuíam maior experiência com testes ficaram sobrecarregados no início do projeto. Além das tarefas de testes, eles também coordenaram os treinamentos. Como solução para balancear os papéis, após o treinamento os demais integrantes assumiam atividades gradativamente enquanto adquiriam experiência na prática.
4. As modificações no ciclo de testes eram comunicadas em reuniões presenciais para que todos os envolvidos nas atividades de teste estivessem em sintonia seguindo o mesmo padrão.

6. Conclusões

Com o desenvolvimento distribuído de software, a garantia da qualidade dos sistemas se torna uma difícil tarefa. Um estudo de caso foi realizado com o intuito de analisar a possibilidade de manter a qualidade do software produzido por desenvolvimento distribuído, para isso as boas práticas da metodologia ágil *Scrum* foram adaptadas para o processo de testes.

Durante o projeto uma base de dados foi gerada. Através dela, uma série de análises foram feitas, permitindo assim visualização da evolução do produto. Constatou-se que do total de 292 solicitações de mudança cadastradas somente 18 não foram solucionadas.

A quantidade de dias necessários para analisar, corrigir e validar uma solicitação de mudança foi em média 3 dias. A Solicitação que passou mais tempo em aberto durou 29 dias para ser finalizada devido a questões de priorização. No projeto como um todo foram gastos em torno de 1000 horas na análise, estudo e correção das solicitações.

Finalizando, foi possível constatar que o comprometimento pessoal dos membros da equipe é importante para o sucesso do desenvolvimento distribuído. Outro ponto fundamental foi a comunicação para evitar o retrabalho e a perda de tempo. A adaptação da metodologia *Scrum* permitiu que o processo de teste acompanhasse o desenvolvimento e a distribuição das atividades de teste entre os membros foi um fator decisivo para garantir a qualidade do produto. Ao final da disciplina o módulo estável, com novas funcionalidades e com falhas irrelevantes foi disponibilizado para o usuário final.

6.1. Trabalhos Futuros

Como trabalho futuro existe a proposta de replicar o estudo de caso na disciplina de desenvolvimento distribuído da Universidade Federal da Bahia (UFBA)[1] em colaboração com a universidade de Jilin da China e a universidade do estado de Iowa dos Estados Unidos.

A proposta tem como objetivo analisar o fator da qualidade durante o desenvolvimento distribuído de times com fuso horário diferentes, como os problemas culturais e de comunicação serão mitigados. Por fim, outra proposta será levar as adaptações das técnicas de validação da qualidade para projetos de desenvolvimento distribuído na indústria.

[1] http://disciplinas.dcc.ufba.br/MATB14/

References

Audy, J. L. N., Prikladnicki, R. (2007). *Desenvolvimento Distribuído de Software: Desenvolvimento de Software com Equipes Distribuídas*. Campus/Elsevier.

Binder, J. (2007). *Global project management: communication, collaboration and management* across borders. Gower.

Bosnic, I., Cavrak, I., Orlic, M., Zagar, M., Crnkovic, I. (2011). Avoiding Scylla and charybdis in distributed software development course. In *Proceedings of the 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development*, pages 26–30, New York, NY, USA. ACM.

Cavalcanti, E., Maciel, T. M., Albuquerque, J. (2009). Ferramenta Open-Source para Apoio ao Uso do Scrum por Equipes Distribuídas. *In III Workshop de Desenvolvimento Distribuído de Software*.

Chalegre, V. C., Santos, W. B., Souza, L. D., Muñoz, H. J., Meira, S. R. L. (2010). Estudo de Caso da Utilização de Scrum no Desenvolvimento Distribuído de Software. In Workshop Brasileiro de Métodos Ágeis.

Collard, J. F., Burnstein, I. (2002). *Practical Software Testing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Cristal, M., Wildt, D., Prikladnicki, R. (2008). Usage of scrum practices within a global company. In *Proceedings of the 2008 IEEE International Conference on Global Software Engineering*, pages 222–226, Washington, DC, USA. IEEE Computer Society.

Júnior, A. B., Kenji, F., Alves, P., Rocha, R., de Azevedo, R. R., Meira, S. (2011). A Utilização de Práticas Scrum no Desenvolvimento de Software com Equipes Grandes e Distribuídas: um Relato de Experiência. In *V Workshop de Desenvolvimento Distribuído de Software*.

Keller, A. (2005). Automating the Change Management Process with Electronic Contracts. In *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology Workshops*, pages 99–108, Washington, DC, USA. IEEE Computer Society.

Lethbridge, T. C., Sim, S. E., Singer, J. (2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering*, 10:311–341.

Olson, G. M., Olson, J. S. (2000). Distance matters. *Human-Computer Interaction*, 15(2):139–178.

Runeson, P., Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164.

Schwaber, K., Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall.

Soares, F., Mariz, L., Cavalcanti, Y. C., Rodrigues, J. P., Neto, M. G., Bastos, P., Almeida, A. C., Pereira, D., Seabra, R., Albuqueruqe, J. (2007). Adoção de SCRUM em uma Fábrica de Desenvolvimento Distribuído de Software. In *I Workshop de Desenvolvimento Distribuído de Software*.