CBSOFT

Congresso Brasileiro de Software: **Teoria e Prática** BRASILIA.2013

Congresso Brasileiro de Software: **Teoria e Prática**

29 de setembro a 04 de outubro de 2013

Brasília-DF

Anais

WDDS 2013

VII WORKSHOP DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE

WDDS 2013

VII Workshop de Desenvolvimento Distribuído de Software

29 de setembro de 2013

Brasília-DF, Brasil

ANAIS

Volume 01
ISSN: 2178-6097

COORDENAÇÃO DO WDDS 2013

Alexandre L'Erario (UTFPR)
Elisa Hatsue Moriya Huzita (UEM)

COORDENAÇÃO DO CBSOFT 2013

Genaína Rodrigues – UnB
Rodrigo Bonifácio – UnB
Edna Dias Canedo - UnB

REALIZAÇÃO

Universidade de Brasília (UnB)
Departamento de Ciência da Computação (DIMAp/UFRN)

PROMOÇÃO

Sociedade Brasileira de Computação (SBC)

PATROCÍNIO

CAPES, CNPq, Google, INES, Ministério da Ciência, Tecnologia e Inovação, Ministério do Planejamento, Orçamento e Gestão e RNP

APOIO

Instituto Federal Brasília, Instituto Federal Goiás, Loop Engenharia de Computação, Secretaria de Turismo do GDF, Secretaria de Ciência Tecnologia e Inovação do GDF e Secretaria da Mulher do GDF

WDDS 2013

VII Workshop de Desenvolvimento Distribuído de Software

September 29, 2013

Brasília-DF, Brazil

PROCEEDINGS

Volume 01
ISSN: 2175-9677

WDDS 2013 CHAIRS

Alexandre L'Erario (UTFPR)
Elisa Hatsue Moriya Huzita (UEM)

CBSOFT 2013 GENERAL CHAIRS

Genaína Rodrigues – UnB
Rodrigo Bonifácio – UnB
Edna Dias Canedo - UnB

ORGANIZATION

Universidade de Brasília (UnB)
Departamento de Ciência da Computação (DIMAp/UFRN)

PROMOTION

Brazilian Computing Society (SBC)

SPONSORS

CAPES, CNPq, Google, INES, Ministério da Ciência, Tecnologia e Inovação, Ministério do Planejamento, Orçamento e Gestão e RNP

SUPPORT

Instituto Federal Brasília, Instituto Federal Goiás, Loop Engenharia de Computação, Secretaria de Turismo do GDF, Secretaria de Ciência Tecnologia e Inovação do GDF e Secretaria da Mulher do GDF

APRESENTAÇÃO

É com grande satisfação que apresentamos a seleção de artigos do VII Workshop de Desenvolvimento Distribuído de Software (VII WDDS), que acontece como evento co-alocado ao CBSoft 2013.

A sétima edição do Workshop de Desenvolvimento Distribuído de Software (WDDS 2013) é um fórum para apresentação e discussão de resultados e experiências de pesquisadores e praticantes da área. O objetivo é viabilizar a geração de conhecimentos e a troca de experiências para facilitar a adoção, a avaliação e a execução de projetos DDS de caráter acadêmico e industrial na América Latina.

O comitê de programa do VII WDDS é constituído por 17 membros provenientes de todo o território nacional. O comitê foi responsável em selecionar 12 artigos de um total de 21 submetidos, visando manter a qualidade técnica e, ao mesmo tempo, proporcionar uma representatividade institucional e de locais. O processo de avaliação garantiu que cada submissão fosse avaliada, pelo menos, por 3 revisores.

O programa do evento se destaca pela excelência das sessões técnicas, constituídas pelos trabalhos selecionados de abrangência nacional, identificando-se assim potenciais grupos de pesquisa na área. O VII WDDS apresenta como novidade a parceria estabelecida com a comunidade de ECOSistemas, representando assim mais um passo dado em direção a colaboração entre áreas que se sobrepõem e o fortalecimento de grupos de pesquisas.

Agradecemos imensamente o apoio recebido do CBSoft por abrigar, mais uma vez, este evento. Agradecemos aos autores pelo interesse e qualidade de suas contribuições. Gostaríamos também de agradecer a todos os membros do Comitê de Diretivo pelos seus esforços em promover o VII WDDS e aos membros do Comitê de Programa pelos seus trabalhos na revisão dos artigos e contribuições na discussão final.

Brasília, setembro 2013

Alexandre L'Erario e Elisa Hatsue Moriya Huzita
Coordenadores do WDDS 2013

FOREWORD

It is our great pleasure to welcome you to the VII Workshop on Distributed Software Development (WDDS VII), which is hosted as a co-located event to CBSoft 2013.

The seventh edition of the Workshop on Distributed Software Development (WDDS 2013) provides a forum for presentation and discussion of results and experiences among researchers and practitioners on distributed software development (DSD). The goal is to further knowledge and to share experiences that aim to facilitate the adoption, evaluation and implementation of DSD projects in both academia and industry in Latin America.

The VII WDDS program committee was composed by 17 members, all affiliated to an institution located in Brazil. The committee was responsible for selecting 12 papers out of the 21 submitted. Each paper was reviewed by at least 3 reviewers.

The workshop stands out for its technical paper presentation session's excellence, consisting of research work selected nationwide, thus identifying potential research groups in the area. The WDDS VII presents as a novelty a partnership with the community of Software Ecosystems, representing a further step towards collaboration and research groups.

We greatly appreciate the support received from CBSoft for hosting this event once again. We thank the authors for their interest and quality of their contributions. Finally, we would like to thank all the members of the Steering Committee for their efforts in promoting the workshop and all the members of the Program committee for their work in reviewing the papers and for their valuable contributions in the follow-up discussions.

Alexandre L'Erario and Elisa Hatsue Moriya Huzita
Coordinators - WDDS 2013

BIOGRAFIA DOS COORDENADORES / WDDS 2013
CHAIRS SHORT BIOGRAPHIES

ALEXANDRE L'ERARIO

Alexandre L'Erario é professor da Universidade Tecnológica Federal do Paraná – Campus Cornélio Procópio – desde 2005. Atualmente é vice-coordenador do mestrado profissional do programa de pós-graduação em informática e coordenador do curso de tecnologia em análise e desenvolvimento de sistemas. Seu título de doutorado foi obtido na Escola Politécnica da Universidade de São Paulo em 2009. Desde então seus esforços centram-se em pesquisar problemas relacionados a coordenação e comunicação em ambientes de desenvolvimento distribuído de software.

Alexandre L'Erario is a professor at the Federal Technological University of Paraná - Campus Cornélio Procópio - since 2005. He is currently the vice coordinator of the professional Master's Degree program in Computer Science and the coordinator of the Technologist Degree in System Analysis and Development. He has a PhD in Engineering from the Polytechnic School, University of São Paulo (2009). Since then his efforts are on researching issues related to coordination and communication in distributed software development projects.

ELISA HATSUE MORIYA HUZITA

Elisa Hatsue Moriya Huzita é doutora em Engenharia Elétrica/Computação pela USP-SP. É professora no Departamento de Informática da Universidade estadual de Maringá (DIN/UEM). Atualmente é, também, professora do Programa de Pós-Graduação em Ciência da Computação do DIN/UEM. Pesquisadora na área de Engenharia de Software, tendo como principais interesses: Desenvolvimento Distribuído de Software, Desenvolvimento Baseado em Componentes, Gestão de Conhecimento e de Informações de Contexto em Desenvolvimento Global de Software e Gerenciamento de Projetos. Coordena projeto de pesquisa em Gestão de Conhecimento e de Informações contextuais em Desenvolvimento Distribuído de Software no DIN/UEM. Membro de entidades de área (ACM, IEEE). Autora de diversos artigos em eventos nacionais, internacionais e revistas e, também, membro de comitê de diversos eventos.

Elisa Hatsue Moriya Huzita is a professor in the Department of Informatics at the State University of Maringá (DIN / UEM). Currently, she is also a professor at the Graduate Program in Computer Science from the DIN / UEM. She has a PhD in Electrical Engineering from the University of São Paulo. Her research interests in Software Engineering are Distributed Software Development, Component-Based Development, Information and Knowledge Management in Global Software Development, and Project Management. She coordinates a research project on Contextual Information and Knowledge Management in Distributed Software Development at the DIN / UEM. She is a member of the IEEE and the ACM institutions. She has co-authored several journal articles and served as a committee member of several conferences and workshops.

COMITÊS TÉCNICOS / TECHNICAL COMMITTEES

COMITÊ DE PROGRAMA / PROGRAM COMMITTEE

Arilo Claudio Dias Neto, UFAM
Carina Alves, UFPE
Claudia Werner, COPPE/UFRJ
Fernando Figueira Filho, UFRN
Gledson Elias, UFPB
Heitor Costa, UFLA
Jorge Luiz Audy, PUCRS
José Augusto Fabri, UTFPR
Leonardo Murta, UFF
Marco Aurelio Gerosa, IME/USP
Rafael Prikladnicki, PUCRS
Renata Fortes, ICMC/USP
Sabrina Marczak, PUCRS
Tania Fatima Calvi Tait, UEM
Tayana Uchôa Conte, UFAM
Vander Ramos Alves, UNB

REVISORES DE TRABALHOS / REVIEWERS

George Valença
Josiane Kroll
Priscila Silva Fernandes
Rodrigo Santos

COMITÊ ORGANIZADOR / ORGANIZING COMMITTEE

COORDENAÇÃO GERAL

Genaína Nunes Rodrigues, CIC, UnB
Rodrigo Bonifácio, CIC, UnB
Edna Dias Canedo, CIC, UnB

COMITÊ DIRETIVO / STEERING COMMITTEE

Claudia Werner, COPPE/UFRJ
Cleidson de Souza, IBM Brasil
Elisa Huzita, UEM
Heitor Costa, UFLA
Renata Fortes, ICMC/USP
Sabrina Marczack, PUCRS

PALESTRA CONVIDADA / INVITED KEYNOTE

DESAFIOS DO DDS E A ABORDAGEM DO GRUPO VOLVO

O Desenvolvimento Distribuído de Software(DDS) já não é mais uma novidade para grandes corporações, que mantém operações, simultaneamente, por todo globo terrestre. Essas empresas tiram proveito de tal abordagem para manterem-se competitivas em um mercado de TI que busca por pessoas qualificadas, capacidade de manter aplicações disponíveis 24 horas por dia, 7 dias por semana; qualidade, agilidade e ainda sim, com custos reduzidos.

É possível notar transformações em alguns aspectos do DDS, como a cultura global, criada dentro das organizações, sendo assimilada por seus colaboradores, ou pessoas cada vez mais capazes de se comunicar em uma língua comum, que não a sua nativa, todavia, os desafios do DDS continuam presentes no dia a dia.

Essa palestra objetiva apresentar aspectos do DDS, discutir experiências e trazer algumas das propostas desenvolvidas pelo Grupo Volvo para enfrentar tais desafios.

ALBERTO BARBOSA BIASÃO

Natural de Maringá – PR, mestre e bacharel em Ciência da Computação pela Universidade Estadual de Maringá (UEM), com 9 anos de experiência em desenvolvimento de aplicações, sendo os últimos 4 em ambientes globais.

Mestrado em ciência da computação focou-se, principalmente, na área de engenharia de software. A pesquisa buscou enfrentar desafios do desenvolvimento distribuído de software, por meio de um mecanismo de inferência baseado em percepção de contexto (contextawareness). A arquitetura do mecanismo fez uso de agentes de software, os quais atuavam sobre uma base de conhecimento representada por uma ontologia. A pesquisa resultou em duas publicações internacionais, junto ao grupo de pesquisa.

Possui conhecimento técnico, focado principalmente em tecnologias relacionadas alinguagem Java. Com relação a processos de desenvolvimento, contam experiências emmetodologias tradicionais – cascata – e metodologias ágeis/Lean – Scrum e Kanban – em ambientes globais, e em diferentes áreas de negócios: indústria automotiva, sistemas bancários e sistemas de telecomunicação. Participou de projetos de empresas como Nokia Siemens Network, HSBC, CPqD, Caixa Econômica Federal e Volvo. Tais oportunidades trouxeram experiências próximas ao cliente, alguns deles de diferentes países.

Nos últimos dois anos vem trabalhando como arquiteto de software na Volvo do Brasil em Curitiba, junto a área de Market After Sales and Deliver Customer Loyalty, em sistemas de diagnóstico de falhas em caminhões.

ÍNDICE DE ARTIGOS / TABLE OF CONTENTS

POSTER/SHORT PAPER

A utilização da Programação em Par Distribuída no ensino de programação

Bernardo José da Silva Estácio, Rafael Prikladnicki

Programa de Pós-Graduação em Ciência da Computação – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre - RS – Brasil

{bernardo.estacio@acad.pucrs.br, rafael.prikladnicki@pucrs.br}

Abstract. *Courses taken at a distance are an increasingly present reality in different areas of knowledge. In Computing is not different, programming teaching at a distance requires new challenges and practices for learning to be effective. In the literature, the practice of Pair Programming (PP) has proved to be an effective practice in the learning process. However few studies have examined the PP in the distributed context as a teaching tool and its importance to compose a resume of a computer course. This paper aims to present a conceptual basis of Distributed Pair Programming (DPP) in teaching, presenting the main lessons learned from studies conducted in the area.*

Resumo. *Cursos realizados a distância são uma realidade cada vez mais presente em diversas áreas do conhecimento. Na Computação não é diferente, o ensino de programação a distância requer novos desafios e práticas para que o aprendizado seja efetivo. Na literatura a prática de Programação em Par (PP) mostrou ser uma prática efetiva no processo de aprendizado. Contudo, poucos estudos analisaram a PP no contexto distribuído como ferramenta de ensino e a sua importância de compor um currículo de um curso de Computação. Este artigo tem como objetivo apresentar uma base conceitual de Programação em Par Distribuída (PPD) no ensino, apresentando as principais lições aprendidas dos estudos conduzidos na área.*

1. Introdução

O advento de plataformas virtuais de ensino consolidadas aliada à difusão de cursos semipresenciais e a distância contribui para que atividades pedagógicas sejam cada vez mais realizadas de forma remota, distribuída [Edwards et al. 2010]. A crescente adoção do Desenvolvimento Distribuído de Software (DDS) nas organizações também corrobora na importância do tema e da experiência no currículo dos alunos de um curso de Computação em nível de graduação [L'Erario et al. 2008].

A Programação em Par (PP) é uma prática do método XP que consiste em dois desenvolvedores trabalhando em um computador sobre o mesmo algoritmo, código ou projeto [Mcdowell et al. 2002]. Esta prática já mostrou muitos benefícios no ensino, tais como: a facilidade de aprendizado [Carver et al. 2007], a motivação dos alunos [Mcdowell et al. 2002] e a melhora na qualidade do código [Ramli et al. 2008]. No contexto distribuído, a Programação em Par Distribuída (PPD) é uma alternativa nos cursos que envolvem o desenvolvimento de software remotamente. Poucos estudos na literatura exploram empiricamente a PPD como ferramenta de ensino. Desta forma, o principal objetivo deste artigo é apresentar os resultados de uma revisão da literatura

sobre os estudos já existentes, apresentando as principais variáveis em torno da prática, bem como um conjunto de lições aprendidas.

Este artigo está estruturado da seguinte maneira: a próxima seção apresenta os principais estudos envolvendo a PP no ensino de programação. Na Seção 3, são apresentados os principais conceitos em torno da PPD. A Seção 4 apresenta uma caracterização de PPD no ensino de programação a partir dos estudos presentes na literatura. A Seção 5 apresenta um conjunto de lições aprendidas obtidas a partir destes estudos. Por fim, na Seção 6 as considerações finais deste trabalho apresentado.

2. O uso da PP no ensino de programação

Alguns estudos da literatura buscaram avaliar PP no contexto educacional. Han et al. (2011) e Ramli et al. (2008) avaliaram a qualidade do código produzido pelos alunos a partir de algumas heurísticas que apresentaram bons resultados, tais como: pouca quantidade de erros gramaticais e lógicos, mais tarefas finalizadas dentro de um espaço de tempo e uso de comentários e variáveis significantes ao longo do código desenvolvido pelos alunos.

Quanto ao desempenho dos alunos, três estudos mostraram que PP ajudou os programadores a desenvolverem melhor as suas atividades. Ramli et al. (2008) relata que os alunos obtiveram um melhor desempenho nas atividades em pares do que individualmente [Ramli et al. 2008]. Mcdowell et al. (2002) apontou como desempenho duas métricas: a qualidade dos programas desenvolvidos e a capacidade dos estudantes aplicarem os conceitos ensinados durante o curso, as notas finais obtidas pelos alunos detalhada foi usada como parâmetro para esses critérios [Mcdowell et al. 2002].

Alguns estudos apresentaram que a PP é uma prática propícia para o aprendizado. Nagappan e Carver afirmam que PP auxilia na não evasão de alunos nos cursos de introdução a programação [Carver et al. 2007], [Nagappan et al. 2003]. Alguns estudos do tipo *survey* mostraram que os alunos ficaram mais motivados e tiveram mais satisfação ao usar a PP [Chigona et al. 2008], [Mcdowell et al. 2002]. Quanto à diferença de personalidade no uso de PP no ensino alguns estudos mostraram que não há um efeito significativo na PP e enfatizam que mais estudos precisam ser realizados [Salleh et al. 2009], [Salleh et al. 2010].

Outra variável importante que tem um efeito positivo no ensino em PP é a produtividade. Nagappan et al. (2003) relatou que no curso de introdução de programação os alunos foram mais produtivos e menos frustrantes [Nagappan et al. 2003]. A medida usada foi a nota dos alunos. Chigona et al. (2008) também ratifica esses resultados por meio de um experimento, onde os alunos possuíram menos dúvidas e informaram por meio de uma *survey* que foram mais produtivos usando PP [Chigona et al. 2008].

A PP também mostrou ser eficiente por meio do aumento das notas dos alunos. Seis estudos apontaram essa métrica positiva [Braught et al. 2008], [Braught *et al.*, 2011], [Brereton et al. 2009], [Mendes et al. 2005]. Outros seis estudos apresentaram que PP aumentou a confiança do programador e todos eles foram aplicados em um contexto educacional em experimentos envolvendo alunos. Para medir a confiança os estudos aplicaram *surveys* após cursos que utilizavam PP [Hanks 2008], [Han et al. 2011], [Mcdowell et al. 2002], [Ramli et al. 2008], [Salleh et al. 2011]. Os resultados

nestes estudos apontam que PP tem um efeito positivo na confiança ao desenvolver as atividades de programação.

3. A Programação em Par Distribuída

A adoção crescente do DDS pelas organizações implica na investigação de como as práticas de desenvolvimento de software são utilizadas com equipes distribuídas. Com a estratégia de adoção de Métodos Ágeis no DDS, a PP se torna alvo desse contexto, sendo uma prática ágil que exige comunicação face a face e uma colaboração com a interação constante entre os programadores. A distribuição geográfica levanta alguns desafios nesses aspectos. Desta forma, a Programação em Par Distribuída (PPD), também conhecida como Programação em Par Virtual ou ainda Programação em Par Remota, é uma prática que possui a necessidade de mais investigações e estudos empiricos.

Baheti (2002) atestou que os benefícios de PP são os mesmos em PPD, tais como a produtividade e qualidade do código [Baheti 2002]. Outro benefício de PPD é que dada as suas características, ela ajuda à promoção do trabalho e da comunicação dentro de equipes distribuídas [Baheti 2002]. Contudo, o foco aos aspectos de infraestrutura deve ser redobrado, e isto se reflete na adoção de um ferramental especifico para a prática. Canfora et al.(2006) em seu estudo relatou que a qualidade do código sofre uma pequena diminuição devido as questões de infraestrutura.

Ho et al. (2004) aponta que o uso de ferramentas específicas para PPD ajuda a tornar a prática mais rápida [Ho et al. 2004]. Canfora corrobora dizendo que uma ferramenta específica evita que o programador fique alternando entre diferentes ferramentas, gastando tempo [Canfora et al. 2006]. Natsu et al. (2003) e Stotts et al. (2004) propuseram diferentes ferramentas de apoio à PPD. Este ferramental tem que prover um bom canal de comunicação por meio de *chat* e divisão da área de trabalho entre os programadores. Natsu et al. (2003) e Stotts et al. (2004) propuseram diferentes ferramentas de apoio à PPD.

Outro ponto a ser abordado é a ideia de time, com a distância geográfica são necessárias estratégias para que todos conheçam o projeto de maneira igual para que não haja empecilhos no desenvolvimento do sistema e não afetar a colaboração entre os pares. A colaboração também pode enfrentar outros obstáculos como o fuso horário, a cultura e o idioma [Canfora et al. 2006]. Quanto ao esforço não há evidências que relatam um aumento [Canfora et al. 2006].

A PPD também mostrou ser uma prática que gera benefícios no ensino de programação. Alguns efeitos positivos foram encontrados em relação ao aprendizado, qualidade do Código e ao desempenho do programador [Hanks 2005]. Em um experimento realizado por Hanks (2005), os alunos que realizaram a PPD tiveram um desempenho tão bom quanto os alunos que estavam com pares co-locados, passando no curso com notas similares. A próxima seção terá um detalhamento maior sobre as variáveis e efeitos em torno dos estudos da prática e do ensino em PPD.

4. O uso de PPD no ensino de programação

Na literatura são encontrados vários estudos sobre ferramentas de apoio e a exploração de PPD como prática, havendo poucos trabalhos que lidam com PPD no ensino de programação. Neste sentido, a tabela 1 apresenta quatro estudos, identificados em uma

revisão sistemática de PPD executada por Estácio (2013), e as suas principais características que analisam PDD sob a perspectiva pedagógica.

Tabela 1. Estudos sobre PPD no ensino de programação

Estudos	Variáveis Independentes	Principais variáveis dependentes	Tipo de coleta das variáveis
[Hanks 2005]	PP X PPD	Desempenho Confiança Satisfação	Nota dos alunos *Survey* aplicada aos alunos
[Zin et al. 2006]	PPD	Confiança Aprendizado	*Survey* aplicada aos alunos
[Edward et al. 2010]	PP X PPD	Satisfação Compromisso Aprendizado	*Survey* aplicada aos alunos
[Zacharis et al. 2011]	PI X PPD	Desempenho Satisfação Qualidade Produtividade de código	Nota dos alunos *Survey* aplicada aos alunos LOC/H

Hanks (2005) realizou um experimento com alunos a respeito de verificar o desempenho do uso de uma ferramenta de apoio a programação em par distribuída em um curso introdutório de programação [Hanks 2005]. O experimento envolveu 112 alunos, sendo que 57 utilizaram PPD e 55 fizeram pareamento co-locado, executando em três sessões. Os resultados mostraram que os alunos que realizaram PPD tiveram um desempenho tão bom quanto os alunos que estavam com pares co-locados, passando no curso com notas similares. *Surveys* foram aplicadas no início e ao final do curso e os resultados indicaram que o nível de confiança se mostrou estatisticamente semelhante.

O experimento realizado por Zin et al. (2006) buscou avaliar a efetividade de PPD. A limitação principal desse estudo é que a PPD foi utilizada de forma assíncrona devido ao sistema utilizado na universidade. O experimento foi realizado em um curso de orientação a objetos, onde 147 participaram da pesquisa respondendo uma *survey* aplicada. Os resultados apontaram que a PPD ajudou na confiança dos alunos e no aprendizado. Os alunos também indicaram que seria mais interessante utilizar a PPD de forma síncrona, por meio de ferramentas de compartilhamentos, isto facilitaria o dialogo entre os pares.

O estudo realizado por Edwards et al.(2010) tinha por objetivo avaliar a efetividade de PPD no currículo de um curso de informática *online*. Um experimento foi realizado com 100 alunos da universidade de Indiana Bloomington de um curso de introdução de Computação. As atividades foram divididas em duas sessões: a primeira utilização PP com pares co-locados e a segunda usando PPD. Ao término, os alunos responderam uma *survey* sobre o compromisso, motivação e o aprendizado deles. Quanto aos resultados, a PP teve um faixa de 80-90% de aprovação pelos alunos, já a PPD teve um índice um pouco menor em uma faixa de 72%- 80%. Edwards aponta que existe uma diferença entre PP e PPD, principalmente em relação à metodologia, mas o uso de uma ferramenta específica é uma solução viável.

Zacharis *et al.* (2011) realizou um estudo com alunos para investigar a efetividade de PPD no desempenho dos alunos e a sua motivação em um curso de introdução de Java [Zacharis et al. 2011]. O experimento foi conduzido com 129 alunos. A comparação foi feita com a programação individual (PI) e os resultados apontaram que os alunos que usaram PPD tiveram 50% menos defeitos, evidenciando a qualidade do código, e foram mais produtivos baseado na métrica de LOC/h [Zacharis et al. 2011]. O desempenho também aumentou com as notas dos alunos. Os alunos também responderam uma *survey* ao final do curso, relatando que PPD deu mais motivação e satisfação.

5. Lições Aprendidas

Os estudos conduzidos por Hanks (2005), Zacharis et al. (2011), Edwards et al. (2010) e Zacharris et al. (2011) apresentam características do uso e aplicação de PPD como ferramenta de ensino. Baseado nestes estudos, algumas lições aprendidas foram identificadas para apoiar a adoção e o uso de PPD no ensino de programação.

Lição 1: *A adoção de PPD necessita de uma ferramenta que possibilite o compartilhamento entre os pares:* o experimento de Hanks (2005) e Zacharis *et al.* (2010) utilizaram uma ferramenta própria para PPD, onde é possível simular outro cursor em forma de "gestos" apontando partes do código, semelhante ao que é feito com os dedos em pareamentos co-locados. Uma ferramenta específica em PPD é recomendada também Canfora et al. (2006) e Ho et al. (2004) que afirmam que tais ferramenta reduzem o tempo de alternar entre diversos programas. Edwards *et al.* (2010) utilizou uma ferramenta proprietária de conferência que exigia apenas acesso a internet, possuindo facilidade na customização. Apesar de Zin et al. (2006) ter implementado a PPD de forma assíncrona, os alunos atestaram a importância de uma ferramenta específica.

Lição 2: *A PPD retorna bons resultados no desempenho, confiança, motivação, satisfação dos alunos:* Hanks (2005) e Zhacharis et al.(2011) obtiveram bons resultados na análise de desempenho dos alunos em função das notas dos alunos, porém Hanks (2005) afirma que cursos introdutórios, em geral, os alunos possuem notas elevadas. Uma limitação neste caso é que todos os quatro estudos utilizaram a PPD em cursos introdutórios. Hanks (2005) e Zin *et al.* obtiveram de retorno dos alunos que com a PPD eles se sentiram mais confiantes nas tarefas de programação. Já em relação em a satisfação e motivação dos alunos em realizar a prática, os resultados obtidos por Hanks (2005), Edward et al. (2010) e Zhacharis et al. (2011) apresentam resultados positivos.

Lição 3: *Existe uma necessidade de investigar mais aspectos de PPD, como a diferença de personalidade e a qualidade e produtividade do código dos alunos:* Nenhum dos quatro trabalhos analisou a colaboração em PPD com diferentes tipos de personalidades. Segundo a pesquisa de Walle (2009), baseado nas evidências de um estudo com empresas, a diferença de traços na personalidade pode aumentar a quantidade de comunicação entre um par em PP [Walle et al. 2009]. Outro ponto importante a ser levantado são variáveis de aspectos técnicos como a qualidade e produtividade do código, apenas o trabalho de Zacharis et al. (2010) analisou este tipo de métrica que em PP possui vários estudos empíricos como Mcdowell et al. (2002) e Chigona et al. (2008).

Lição 4: *É importante que a PPD seja implementada em seções:* Três trabalhos analisados utilizaram a prática de PPD em seções. Edward et al. (2010) utilizou a primeira seção para explicar sobre o conceito de PP e PPD e suas implicações no DDS aos alunos. Hanks (2005) atribui cinco tarefas de programação em cada uma das seções. Zacharis et al. (2010) no seu curso utilizou 4 seções com PPD, mesclando as outras com programação individual e PP. Apesar de Canfora et al. (2006) afirmar que não há aumento de esforço em PPD, seria interessante conduzir mais estudos para analisar o esforço da prática no contexto pedagógico.

Lição 5: *Existe a necessidade de avaliar o papel de um instrutor em PPD:* Segundo Hannay et al. (2010) a presença de um instrutor ajuda a evitar impasses entre os pares e possibilita que as dúvidas referentes a prática de PP possam sem rapidamente respondidas.. Um estudo anterior de Edwards (1997) afirma que cursos a distância que tratem de programação é importante que se tenham encontros presenciais. No estudo realiazo por Zin et al. (2006), os alunos afirmaram que a presença de encontros presenciais com o tutor é importante. Os autores indicam que na cultura asiática é comum os alunos terem encontros face a face com os instrutores.

Lição 6: *Existe a necessidade de se avaliar a importância de PPD no ensino como instrumento de capacitação do currículo do aluno para o DDS:* Nenhum dos quatro estudos buscou avaliar a importância da PPD não apenas como estratégia de ensino de programação de forma remota, mas também como parte de formação e preparo dos alunos a indústria. Muitas empresas tem adotado o DDS e as práticas ágeis tem sido uma estratégia para diminuir os desafios [Paasivara et al. 2009]. Neste caso, a PPD no ensino de programação também auxiliaria na capacitação dos alunos frente a demanda por profissionais por parte das empresas que utilizam DDS.

6. Considerações Finais

A partir dos resultados dos estudos foi possível identificar os benefícios da PPD no processo de ensino. Assim como a PP, a PPD também pode ser utilizada de forma efetiva como ferramenta pedagógica, colaborando em variáveis como o desempenho, confiança e motivação dos alunos. Do tema foram identificadas seis lições aprendidas, as quais caracterizam o estado da arte de PPD no ensino e listam possíveis oportunidades de pesquisa na área.

Em relação a trabalhos futuros, pretende-se avaliar como o ensino de programação por meio da prática de PPD pode colaborar para a formação de profissionais mais bem preparados para o mercado de DDS. Além disso, muitas oportunidades de pesquisa ainda existem como identificadas em algumas lições aprendidas levantadas neste trabalho. Essa pesquisa visa apoiar tanto a educação, dando suporte para PPD como prática de ensino e parte do conteúdo a ser aplicado no DDS, como a indústria, capacitando para que o currículo dos estudantes seja de acordo com a realidade e necessidades das empresas.

Agradecimentos

O desenvolvimento deste trabalho tem apoio financeiro do convênio PUCRS/Thoughtworks.

Referências

Baheti, P. (2002) “Assessing distributed pair programming”. In: OOPSLA, 2002, p. 50–51.

Brereton, P.; Turner, M.; Kaur, R. (2009) “Pair programming as a teaching tool: a student review of empirical studies”. In: Software Engineering Education and Training, 2009, p. 240-247.

Braught, G.; Eby, L.;Wahls, T. (2008) “The effects of pair-programming on individual programming skill”. In: SIGCSE, 2008, p. 200-204.

Braught, G.; Wahls, T.; Eby, L. (2011) “The case for pair programming in the computer science classroom”. *ACM Transactions on Computing Education*, Fev-2011, vol.11-1.

Canfora, G.; Cimitle, A.; Visaggio,C.; Di Lucca,G (2006) “How distribution affects the success of pair programming”. In:International Journal of Software Engineering and Knowledge Engineering, p. 293-313.

Carver, C.; Henderson, L.; Lulu, He.; Hodges, J.; Reese, D. (2007) "Increased Retention of Early Computer Science and Software Engineering Students Using Pair Programming,". In: Software Engineering Education & Training, p.115-122.

Chigona, W.; Pollock, M. (2008) "Pair programming for information systems students new to programming: Students’ experiences and teachers’ challenges," In: Portland International Conference on Management of Engineering & Technology, p.1587-1594.

Edwards, W.K., Elizabeth D. Mynatt, K. P., Mike J. S., Douglas B. T. & Marvin M. T. (1997) “Designing and implementing asynchronous collaborative applications with Bayou”.In: Proceedings of the 10th annual ACM symposium on User interface software and technology.

Edwards, R. L., Stewart, J. K., & Ferati, M. (2010) “Assessing the Effectiveness of Distributed Pair Programming for an Online Informatics Curriculum”. In: ACM Inroads, 1(1), 48-54

Estácio, B. (2013) “Desenvolvimento de um conjunto de boas práticas para a programação em par distribuída”. Dissertação de Mestrado, PPGCC-PUCRS.

Han, L; Wenjuan X. (2011) "An experimental research of the pair programming in java programming course," In: International Conference on e-Education, Entertainment and e-Management (ICEEE), p.257-260.

Hanks, B. (2005)”Student performance in CS1 with distributed pair programming”. In: SIGCSE Bull, p. 316-320.

Hanks, B. (2008) “Problems encountered by novice pair programmers”. *ACM Journal on Educational Resources in Computing*, vol. 7-4.

Hannay, J.; Arisholm, E.; Engvik, H.; Sjoberg, D." (2010) Effects of personality on pair programming". In:IEEE Transactions on Software Engineering, vol. 36-1, pp.61–80.

Han, L; Wenjuan X. (2011) "An experimental research of the pair programming in java programming course," In: International Conference on e-Education, Entertainment and e-Management (ICEEE), p.257-260.

Ho, C.; Raha, C.; Gehringer,E.;Williams, L. (2004)“Sangam: a distributed pair programming plug-in for eclipse”. In: OOPSLA, p. 73–77.

L’Erario, A. Pessôa, M. (2008). “Um método de ensino e práticas de desenvolvimento distribuído de software para cursos de graduação. In: Anais do XXXVI COBENGE - Congresso Brasileiro de Educação em Engenharia, São Paulo.

Mcdowell, C.; Werner, L. ;Bulock,H. ; Fernald, J.(2002) “The effects of pair-programming on performance in an introductory programming course”. In: SIGCSE technical symposium on Computer science education, p. 38–42.

Mendes,E.; Basil, L.; Fakhri,A.; Reilly, A.” (2005)Investigating pair-programming in a 2-year software development and design computer science course”.In: SIGCSE Bull., , p. 296–300.

Natsu, H.; Favela, J.; Moran, A. L.; Decouchant, D.; Martinez-Enriquez, A. M. (2003) “Distributed pair programming on the Web”. In: Proceedings of the Fourth Mexican International Conference, p. 81-88.

Nagappan, N.; Williams, L., Ferzli, M.; Wiebe, E.; Miller, K.; Balik, S. (2003) “Improving the CS1 experience with pair programming”. In: SIGCSE Bull, ,p. 359-362.

Paasivaara, M.; Durasiewicz, S.; Lassenius, C. (2009) In: “Using Scrum in Distributed Agile Development: A Multiple Case Study”. In: ICGSE , p. 195-204

Ramli, N.; Fauzi S. (2008)“The effects of pair programming in programming language subject”. In: International Symposium on Information Technology, p.1-4.

Salleh, N.; Mendes, E.; Grundy, J.; Burch, G. (2009). “An empirical study of the effects of personality in pair programming using the five-factor model”. In: International Symposium on Empirical Software Engineering and Measurement, 12p.

Salleh, N.; Mendes, E.; Grundy, J. (2011) “Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review”. *IEEE Transactions on Software Engineering*, vol.37-4, Jul-Ago.. 509–525.

Stotts, D.; Mc. Smith, J.; Gyllstrom, K. (2004) “Support for Distributed Pair Programming in the Transparent Video Facetop”. Relatório Técnico, Univerdade da Carolina do Norte.

Walle, T.; Hannay, J.E. (2009) “Personality and the nature of collaboration in pair programming”. In: Empirical Software Engineering and Measurement, p.203-213.

Zin, A.M., Idris, S., & Subramaniam, N.K. (2006) “Improving Learning of Programming Through E-Learning by Using Asynchronous Virtual Pair Programming” . In: Turkish Online Journal of Distance Education-TOJDE, vol.7, number 3, article 13.

Zacharis,N. (2011) “Measuring the effects of virtual pair programming in an introductory programming java course”. IEEE Transactions on Education, vol. 54-, Fev-2011, pp. 168–170.

Desenvolvimento Distribuído de Software (DDS) em Processo Iterativo de Desenvolvimento

Laurinex da Silva Souza[1], Márcio Plaheta[2]

[1]Processamento de Dados Amazonas S/A (PRODAM)
69.020-110– Manaus – AM – Brasil

[2]Fundação Centro de Análise Pesquisa e Inovação Tecnológica (FUCAPI)
69.068-970 – Manaus – AM – Brasil

{laurinexsouza, marcio.palheta}@gmail.com

Abstract. *Increasingly companies are distributing their software development process in several possible ways at different locations, pursuing, among other, low risk, higher profits, and competitive advantage. This article reports the experience of developing software in iterative development, overcoming the challenges of communication, management, team collaboration, and other inherent in distributed development environment software, and acquiring several lessons learned that can be applied in a process of continuous improvement of the development process.*

Resumo. *Cada vez mais as empresas estão distribuindo seu processo de desenvolvimento de software de diversas maneiras possíveis em diferentes localidades, visando, entre outros, menores riscos, maiores lucros, e vantagem competitiva. Este artigo relata a experiência de desenvolvimento de software em processo iterativo de desenvolvimento, vencendo os desafios de comunicação, gestão, colaboração da equipe, e outros inerentes de um ambiente de desenvolvimento distribuído de software, e adquirindo várias lições aprendidas que poderão ser aplicadas num processo de melhoria contínua do processo de desenvolvimento.*

1. Introdução

O ambiente de negócios está evoluindo e vários são os fatores identificados como geradores de um ambiente propício ao desenvolvimento distribuído de software – DDS, como a globalização, o fato de os sistemas de informação se tornarem cada vez mais importantes para a vida das empresas, e também o *outsourcing*, que é o processo de terceirização do desenvolvimento de software, em vez de desenvolvê-lo a própria empresa. Muitas empresas têm buscado o DDS, com o objetivo de ganhar maior agilidade no processo, maior qualidade devido à especialização das áreas, vantagens competitivas e benefícios fiscais, por exemplo. [AUDY et al. 2007].

Um número crescente de organizações tem desenvolvido software de forma distribuída motivadas por uma demanda crescente de serviços de software, e pressão para reduzir o *time-to-market* (colocar o produto no mercado), estar presente no mercado global e ficar próximas dos consumidores; e devido a pouca disponibilidade de profissionais, o que aumentou o custo de contratação incentivando-as a contratar por preços mais baixos em outros lugares. [AUDY et al. 2007].

Todos os desafios da engenharia de software a fim de se entregar o produto no prazo, dentro do custo, e com a qualidade esperada, também compõem o DDS, porém com a distância física e ou temporal, diferenças culturais e dependendo do nível de distribuição, muitos destes fatores se agravam e necessitam de atenção especial, como a comunicação, a coordenação e a cooperação da equipe.

A construção de software com qualidade, segurança, confiabilidade, é o que se busca em qualquer processo de desenvolvimento adotado. A qualidade dos produtos e serviços gerados pode ser rastreada através dos processos usados por uma organização, onde ainda se enfrentam diversos problemas como falta de planejamento, que é uma fase importantíssima para o trabalho que será desenvolvido, mudanças ocorrendo constantemente no ambiente de negócios, gerando necessidades de constantes adequações a fim de se manterem competitivas, e a distribuição do desenvolvimento também se configura como um desafio para os processos de desenvolvimento que na geralmente não estão preparados para este tipo de trabalho, necessitando de grande esforço por parte da equipe para atingir as metas. [PRIKLADNICKI, 2002].

O principal objetivo deste trabalho é analisar o desenvolvimento distribuído de software em processo iterativo de desenvolvimento, identificando as principais dificuldades apresentadas, as formas de resolução dos problemas, com registro das lições aprendidas a fim de poder gerar planos de ação para melhorar o processo. Aplicados a um estudo de caso, que se trata do projeto de desenvolvimento de software X.Y da empresa X, onde foram feitas pesquisas de campo para coletar dados sobre o seu processo de desenvolvimento, para obtenção do cenário de trabalho atual.

Desta forma este artigo está organizado da seguinte forma: a seção 2 conceituará o Desenvolvimento Distribuído de Software, mostrando os seus principais conceitos e características; a seção 3 apresenta o estudo de caso, mostrando o processo de desenvolvimento iterativo de software aplicado, bem como a modelagem do mesmo para uma melhor visualização e entendimento, os resultados obtidos, e lições aprendidas que poderão gerar um plano de ação para melhoria do processo em questão. A seção 4 traz a conclusão e trabalhos futuros.

2. Desenvolvimento Distribuído de Software – DDS

A engenharia de software têm obtido grandes evoluções nas últimas décadas, e muitas melhorias nas ferramentas e métodos utilizados tem permitido com que se possa formar equipes de desenvolvimento em grupos distribuídos, trabalhando no mesmo projeto. Bem como o mercado está passando por grandes avanços da economia, os meios de comunicação estão mais sofisticados, além de certa pressão por redução de custos, o que têm incentivado maiores investimentos em DDS, assim muitas organizações começaram a investir em DDS buscando ter maior qualidade no processo de desenvolvimento, reduzir os custos, obter recursos num âmbito global. [AUDY et al. 2007]

Os principais atores identificados no processo distribuído de software são: **cliente**, que se refere àquele que contratou o serviço; **usuários**, aqueles que de fato usarão o software desenvolvido; e os **desenvolvedores**, que se refere a toda a equipe envolvida no processo de desenvolvimento do software.

A distância física dos atores pode ocorrer de quatro formas possíveis, caracterizando os níveis de dispersão entre os atores envolvidos no DDS, [AUDY et al. 2007]:

- Distância Municipal: atores estão localizados no mesmo município, as reuniões da equipe podem ser mais frequentes;
- Distância Regional: atores estão localizados dentro de um mesmo país, podendo haver diferenças de cultura e fusos-horários, a equipe pode reunir-se em intervalos de tempo curtos;
- Distância Continental: atores estão em países diferentes, porém no mesmo continente, fica um pouco mais difícil haver as reuniões da equipe, e as diferenças de fuso-horário podem dificultar a comunicação;
- Distância Global: atores estão em países diferentes, em diferentes continentes, reuniões face a face ocorrem geralmente no início do projeto, diferenças culturais podem ser barreiras para o trabalho, e fuso-horário pode se tornar empecilho de comunicação da equipe.

Um ambiente de desenvolvimento de software distribuído é caracterizado quando pelo menos um dos atores envolvidos no processo estiver fisicamente distante dos demais. A equipe de desenvolvimento pode estar distribuída, seguindo os níveis de dispersão apresentados acima ou centralizadas, numa mesma localidade. [PRIKLADNICKI, 2002].

As empresas que trabalham com DDS apresentam modelos de negócios definidos para o projeto, visando obter vantagens relativas a custos, qualidades, maior produtividade, e diminuição dos riscos. Os principais modelos de negócios adotados pelas empresas para desenvolver serviços ou produtos de software são, [AUDY et al. 2007]:

- *Onshore insourcing*: demanda interna. Existe um departamento ou subsidiária da empresa no mesmo país;
- *Onshore outsourcing* ou *outsourcing*: contratação de uma empresa do mesmo país;
- *Offshore outsourcing* ou *offshoring*: contratação de uma empresa em outro país;
- *Offshore insourcing* ou *captive/internal offshoring*: criação de uma subsidiária da própria empresa em outro país.

Os desafios que compõem o desenvolvimento de software em geral também fazem parte do DDS, porém surgem outros desafios por apresentar em seu cenário características como a distância física entre os atores, diferenças culturais, fusos-horários distintos, portanto a necessidade de um maior suporte tecnológico e uma melhor estrutura para tratar estas questões. Principais desafios do desenvolvimento distribuído de software, [AUDY et al. 2007]:

São fatores críticos para o sucesso de projetos em ambiente de DDS: a comunicação, devendo existir um fluxo de informações eficaz entre os membros da equipe; confiança no trabalho feito pela equipe de modo que transmita segurança; cooperação da equipe em torno de um objetivo comum; a coordenação das atividades de desenvolvimento; e a identificação do nível de diferenças culturais existentes na equipe. Como mostra a figura 1 a seguir.

Figura 1. Fatores de Sucesso do DDS. (Fonte: PRIKLADNICKI, 2002, p.21)

A coesão da equipe é uma característica fundamental, pois equipes coesas são mais motivadas, têm maior produtividade, melhor comunicação e se sentem mais satisfeitas com o trabalho realizado. Com as características próprias do DDS esta coesão torna-se mais difícil, por diversas causas como: diferenças culturais, distâncias físicas, problemas de comunicação, diferentes idiomas. O que se torna um grande desafio para o DDS, pois as pessoas são componentes fundamentais no processo de desenvolvimento de software, e alguns dos impactos da dispersão são: o uso excessivo de comunicação, a falta de comprometimento e o desconforto ao utilizar alguns meios, daí surge a necessidade de se investir em desenvolvimento da equipe, a fim de obter os melhores resultados. [LOPES, 2004].

3. Estudo de caso

A empresa X é uma empresa de tecnologia da informação, que trabalha com *outsourcing*, os desenvolvedores fazem parte do quadro funcional da empresa, mas os clientes e usuários são externos, apesar de também trabalhar o desenvolvimento de software para prover soluções internas, e alguns casos que podem envolver empresas parceiras em outros estados.

Neste contexto os clientes estão localizados na cidade de Manaus, e os usuários podem estar localizados em Manaus ou em outros municípios do interior do estado do Amazonas. Caracterizando então o desenvolvimento distribuído de software, em uma distância Municipal e /ou Regional entre os atores do processo, como nas figura 2 e 3.

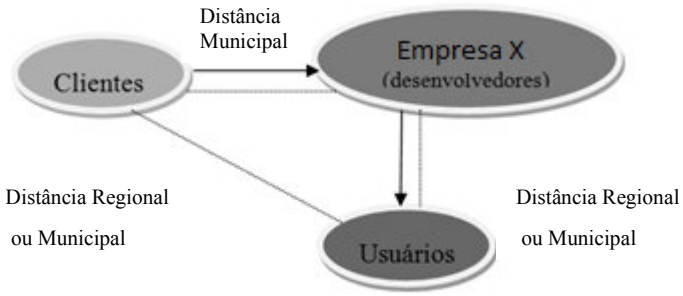


Figura 2. Relacionamento entre os atores do processo.

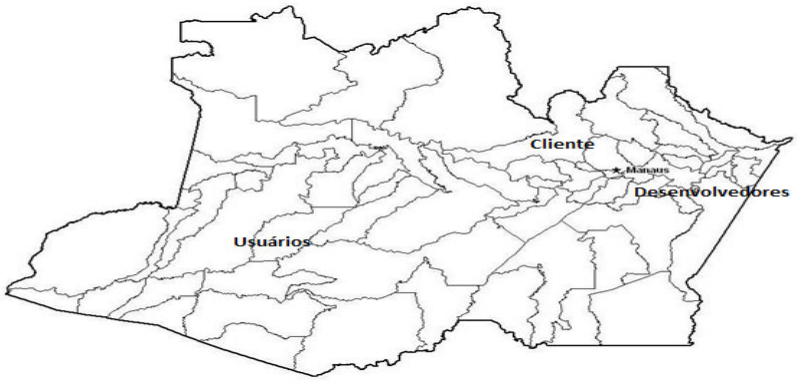


Figura 3. Distância Regional entre os atores do processo.

Quanto à equipe de desenvolvimento, esta trabalha toda na mesma cidade, no mesmo prédio da empresa, porém em diferentes setores, configurando uma descentralização da mesma. Como ilustra a figura 4.

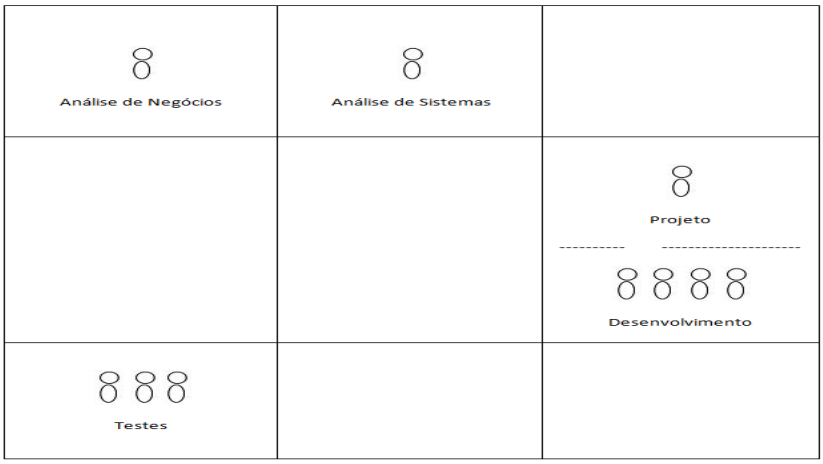


Figura 4. Distribuição da equipe de Desenvolvimento.

3.1. Processo Iterativo de Desenvolvimento

Depois de contratado um serviço pela área comercial através do analista de negócios, o chefe nomeia o analista de sistemas que ficará responsável pelo projeto, e terá a incumbência de fazer toda a análise de requisitos e coordenação do projeto, conforme o ciclo de vida do processo iterativo e incremental de desenvolvimento de software da empresa, modelado na figura 5 abaixo.

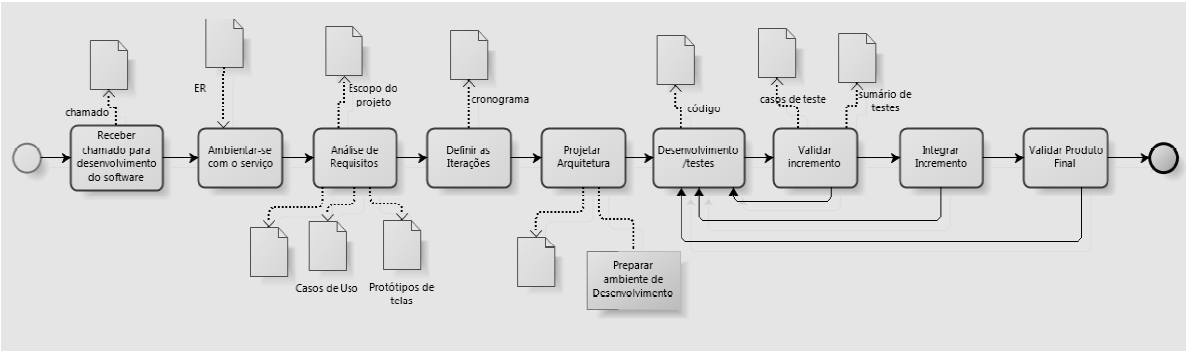


Figura 5. Ciclo de vida do Desenvolvimento de Software

3.1.1. Etapas/Atividades do ciclo de vida de desenvolvimento

As principais etapas realizadas e as atividades dentro de cada etapa são:

- Análise de Requisitos: é feito o levantamento detalhado de informações, através de reuniões e entrevistas com os usuários, a fim de definir os requisitos do software; definição do escopo do projeto; especificação dos casos de uso; realização de reunião de análise crítica com as equipes envolvidas no processo, a fim de obter o consenso da solução a ser desenvolvida e comprometimento de recursos e prazos; criação dos protótipos das telas; geração do modelo Entidade-Relacionamento das tabelas do banco de dados;
- Desenvolvimento/Testes: codificação e geração do programa fonte, compilado e correto; realização de testes dos programas gerados; especificação de casos de testes; realização de testes de integração. Caso sejam encontrados erros, estes são incorporados à próxima iteração (entrega).
- Validar Incremento: os módulos desenvolvidos são repassados à equipe de Testes, que cria e executa os casos de testes, gerando um relatório com os resultados dos testes, caso haja defeitos a serem corrigidos a equipe de desenvolvimento efetua as devidas correções e repete-se o processo até que não haja mais falhas.

Volta-se à fase de desenvolvimento da próxima iteração, incrementando as possíveis correções necessárias da entrega anterior, até que se faça a entrega final do software para que seja disponibilizado em ambiente de produção.

3.2. Resultados Obtidos

O estudo de caso trata do desenvolvimento do Sistema X.Y, cujo cliente está na mesma cidade da empresa através de uma secretaria do governo do estado, e os usuários poderão estar dispersos em diversos municípios do interior do estado nos diversos órgãos vinculados à secretaria em questão, de acordo com o planejamento de implantação do sistema.

Tabela 1. Equipe de Desenvolvimento

Empresa X	Analista de Negócios	Analistas de Sistemas	Desenvolvedores	Testadores
Departamento Comercial	1	-	-	-
Departamento de projetos	-	7	-	-
Departamento de desenvolvimento	-	4	9	-
Departamento de apoio	-	-	-	5

Até o momento neste projeto foram desenvolvidos 205 (duzentos e cinco) casos de uso, agrupados em 9 (nove) módulos, num prazo de 27 (vinte e sete) meses. Possui 28 (vinte e oito) casos de uso em desenvolvimento, dos módulos já implementados; e uma previsão de vários módulos novos a serem especificados e integrados ao sistema.

Dentre as dificuldades enfrentadas, que são próprias do desenvolvimento de software de um modo geral, porém se agravam com o DDS surgindo necessidade de um maior controle e coordenação, pode-se citar:

- Comunicação: por se tratar do envolvimento de pessoas de diferentes setores e fisicamente distantes, enfrentam-se diversos desafios referentes à comunicação, que é essencial para o bom andamento das atividades. Com o amadurecimento dos trabalhos foi-se definindo um processo de comunicação e stakeholders e assuntos a serem comunicados a cada um – percebe-se a grande importância da identificação e correto envolvimento das partes interessadas desde o início do projeto e atualizando-se sempre no seu decorrer;
- Gestão de requisitos: neste cenário o cliente está externo à organização, e para cada módulo do sistema há um grupo de partes interessadas responsáveis pela definição dos requisitos junto aos analistas de sistemas responsáveis por cada módulo, e como trata-se de um sistema que deve ser integrado, acontece muitas vezes conflitos de definições, permuta de stakeholder responsável no meio do processo, impactos nos requisitos de determinado módulo por outro, série de solicitações de mudanças no decorrer do desenvolvimento, onde a equipe de desenvolvimento faz muitas vezes esforços heroicos para fechar o escopo e poder realizar uma entrega – neste sentido a identificação de stakeholder responsável atualizada ajudaria muito, e também uma melhor definição de gerência de requisitos, possibilitando a rastreabilidade dos mesmos com a ajuda de uma ferramenta computacional;
- Burocracias do processo da empresa: neste processo envolvendo diversos setores da empresa, existe grande burocracia, onde cada setor vai ter o chefe responsável, as demandas chegam para este chefe que repassa ao membro da equipe de desenvolvimento, o que faz com que muitas vezes as pessoas envolvidas no projeto não trabalhem de forma coesa, comprometidas verdadeiramente com o sucesso do projeto, o que pode ser causas de retrabalho, desmotivação da equipe, atrasos nas entregas, geração de defeitos, pois não há uma introdução da pessoa ao projeto por esta estar em outro setor – percebe-se aqui a necessidade de se trabalhar o gerenciamento de projetos e quem sabe até propor uma mudança de paradigma da empresa para que trabalhe com uma estrutura matricial forte de desenvolvimento de projetos ou quem sabe mesmo a estrutura projetizada.

3.3. Lições Aprendidas

Algumas lições aprendidas com esta experiência de DDS, que poderão compor um plano de ação para melhorar o processo de desenvolvimento de software da empresa em questão:

- Necessidade de uma comunicação eficaz, onde o comprometimento de cada integrante da equipe é fundamental – definição de um plano de comunicação, identificando todas as partes interessadas, necessidade de comunicação de cada uma e meios de comunicação a serem utilizados;
- O líder precisa desenvolver suas capacidades interpessoais e de liderança a fim de conseguir integrar a equipe e conseguir motivar a todos a uma cooperação mútua – treinamentos;

- A gestão da configuração deve ser definida e tornada do conhecimento de todos no início do projeto, a fim de se evitar problemas de versões, e alterações indevidas – planejamento da gestão da configuração e controle de versões e mudanças;
- As mudanças nos requisitos no decorrer do projeto geram bastante retrabalho e desperdício de tempo e recursos – definição de um processo de gerenciamento e controle de mudanças;
- Realização de reuniões da equipe de desenvolvimento ajuda na compreensão do projeto por parte de todos e no desenvolvimento de soluções de problemas – calendário de reuniões da equipe a fim de tornar coeso o conhecimento do projeto;
- A cooperação entre os membros da equipe é fundamental para o sucesso do projeto, compartilhar conhecimentos e ajuda mútua na resolução de problemas – desenvolver a equipe do projeto.

4. Conclusão e Trabalhos Futuros

Utilizar o desenvolvimento distribuído de software é uma tendência de mercado e vem crescendo competitivamente no Brasil, devido às suas vantagens para as empresas, apesar dos desafios que são críticos para o sucesso do DDS, com destaque para a gestão, gerenciamento de requisitos e comunicação, e os casos de diferentes culturas, idiomas, fusos-horários, necessitando de adaptação, treinamento da equipe, entre outros.

Ao optarem pelo DDS as empresas obtém ganhos em redução de custos e riscos, maior qualidade do software e agilidade do processo, gerando vantagens competitivas.

No cenário da empresa X, constante do estudo de caso, percebe-se muitas oportunidades de melhorias futuras, como um processo mais bem definido de gestão de projetos, e outros pontos que já se percebe um amadurecimento da equipe de acordo com experiências do próprio projeto em estudo, como a definição de um processo de gestão da configuração e de versões.

Para trabalhos futuros pretende-se trabalhar o gerenciamento de projetos nesse ambiente distribuído, principalmente referente à gestão da equipe para que trabalhe de forma coesa, comunicação, requisitos, sistema de controle integrado de mudanças e versões, com objetivo de melhorar o gerenciamento da execução do projeto e das mudanças que surgem no decorrer do desenvolvimento, gerando subsídios para um monitoramento e controle mais eficaz do projeto, reduzindo retrabalho por defeitos ou mudanças de requisitos.

Referências Bibliográficas

Audy, Jorge. Prikladnicki, Rafael. “Desenvolvimento Distribuído de Software – Desenvolvimento de Software com Equipes Distribuídas”, Série Campus. Elsevier Editora Ltda, Rio de Janeiro, 2007.

Prikladnicki, Rafael. “Desenvolvimento Distribuído de Software e Processos de Desenvolvimento de Software”, Curso de Mestrado. PUC- RS, Porto Alegre, 2002.

Lopes, Leandro Teixeira. “Um Modelo de Engenharia de Requisitos para ambientes de Desenvolvimento Distribuído de Software”, Curso de Mestrado. PUC- RS, Porto Alegre, 2004.

Desenvolvimento Distribuído de Software com Scrum

José Augusto Fabri, Alexandre L´Erario, André L. dos Santos Domingues

Programa de Pós Graduação em Informática – Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Cornélio Procópio – Brazil

fabri@utfpr.edu.br, alerario@utfpr.edu.br, anddomingues@utfpr.edu.br

Abstract. *This paper combines Global Development Software (GDS) and Scrum framework (GDScrum). The authors validate this propose by of the controlled experiment. The experiment show that the GDScrum can to be used by software companies whit success.*

Resumo. *O objetivo deste trabalho é aliar as prerrogativas delineadas pela teoria de desenvolvimento distribuído de software ao framework Scrum, gerando assim o DDScrum. Após propor um modelo que una teoria e framework, os autores do trabalho realizaram um experimento controlado do DDScrum. Durante o experimento foi possível verificar que empresas que desejam produzir de forma distribuída podem perfeitamente utilizar o Scrum para isto.*

1. Introdução

A alta carga tributária, a deficiência na formação da mão de obra na área de tecnologia da informação, e a ineficiência do setor produtivo caracterizam como fonte inibidora no processo de expansão externa do Brasil no setor produtivo de software. Universidade, empresa e governo devem desenvolver mecanismos que alterem este cenário.

Com base neste contexto e com o intuito de propor uma alternativa ao processo de produção de software, focando em um aspecto produtivo mais eficiente, este artigo tem como objetivo verificar se é possível aliar as prerrogativas delineadas pelo desenvolvimento distribuído de software aos processos ágeis – basicamente o Scrum.

Para atingir o objetivo proposto este trabalho foi estruturado da seguinte forma: a Seção 2 apresenta alguns conceitos sobre Scrum. A Seção 3 foca a ideia de desenvolvimento distribuído de software. O modelo que une Scrum às prerrogativas do desenvolvimento distribuído de software é mapeado na Seção 4. A Seção 5 apresenta os métodos e os procedimentos utilizados para validar, parcialmente, o modelo. A Seção 6 apresenta a aplicação do modelo por meio de um experimento controlado. Por fim, a Seção 7 apresenta a análise dos resultados, conclusões e perspectivas futuras.

2. Scrum

Segundo Sutherland (2007), o Scrum destaca-se dentre os diferentes métodos ágeis, com uma abordagem enxuta de desenvolvimento. O método surgiu em 1986, quando Takeuchi e Nonaka realizaram um estudo e notaram que pequenos projetos que tinham equipes pequenas e multifuncionais obtinham os melhores resultados. Este estudo serviu como base para que, em

1993, Jeff Sutherland, John Scumniotales e Jeff McKenna criassem o Scrum. Ken Schwaber formalizou a definição de Scrum e ajudou a implantá-lo no desenvolvimento de software em todo o mundo. Uma visão abrangente do SCRUM pode ser visualizada por meio da Figura 1.

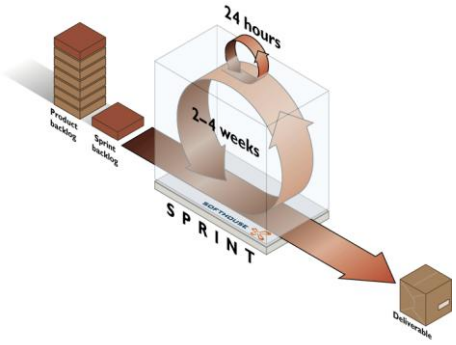


Figura 1 – Scrum *Framework*[1]

O Scrum possui um conjunto de artefatos, entre eles destacam-se: a *Story* (Estória): é uma breve descrição de uma necessidade (ou requisito) do cliente; o *Product Backlog*: são as estórias pendentes. Os requisitos que devem ser implementados para a caracterização do produto; o *Sprint Backlog*: a interpretação técnica de *backlog* do projeto, que resume as tarefas que serão feitas no decorrer do desenvolvimento pela equipe; o *Sprint*: é um período de tempo que pode variar de duas semanas até um mês que resulta em uma parte do software pronto. É importante salientar que após a execução de uma *Sprint* é realizada uma reunião entre os envolvidos com o processo.

Além dos artefatos o Scrum possui alguns papeis: o *Scrum Master*: responsável por cuidar das atualizações diárias do Scrum, papel equivalente ao de gerente de projeto; o *Scrum Team:* uma equipe composta de desenvolvedores, *DBAs* e *quality assurance testers* (responsáveis por desenvolver o produto final); *Product Owner:* voz do cliente na equipe, responsável por manter o foco do projeto nos negócios, ficando em constante contato com os clientes e futuros usuários do sistema.

3. Desenvolvimento Distribuído de Software

Desenvolvimento Distribuído de Software (DDS) é definido pela colaboração e cooperação entre departamentos de organizações e pela criação de grupos de desenvolvedores que trabalham em conjunto, porém distantes fisicamente (Prikladnicki e Audy 2006).

De acordo com Marquardt e Horvath (2001), DDS é composto por equipes globais, grupos de pessoas de diferentes países que trabalham em conjunto no desenvolvimento de um projeto. Karolak (1998) aborda o ambiente distribuído como o resultado da união de organizações virtuais, que se referem a entidades que desenvolvem partes de um projeto em locais dispersos, porém encarando-o como um projeto local.

Prikladnicki e Audy (2006) categorizam o DDS de acordo com a distância geográfica entre os sites (cada unidade de trabalho), classificando-o em níveis de dispersão nacional,

[1] Figura retirada de: http://dojofloripa.wordpress.com/2007/02/07/scrum-em-2-minutos/

continental e global, com equipes localizadas em um mesmo país, em países diferentes situados em um único continente e com alocações em mais de um país, respectivamente. Os dois últimos casos caracterizam desenvolvimento global de software (GSD).

Para considerar um ambiente como distribuído, os autores deste trabalho, apontam alguns requisitos: o ambiente distribuído deve ser constituído por no mínimo dois sites; e deve haver distância física entre eles; há uma granularidade de repasse, que identifica a forma na qual o site entrega seu subproduto para a rede; difusão do processo, aspectos processuais em comum entre os sites da rede; e grau de interação, que revela o quanto a informação é trocada entre os nós, mediante a quantidade de estímulos enviados entre uma instância e outra para realizar uma tarefa.

4. Modelo proposto - DDSCrum

Um modelo caracteriza-se como um artefato mental, destinado a mapear uma determinada realidade, ou alguns de seus aspectos, a fim de torná-los descritíveis qualitativa e quantitativamente e, em alguns casos, passíveis de observação. O surgimento dos modelos busca extrapolar a limitação da percepção humana com relação ao mundo.O modelo proposto neste trabalho reuni as características relevantes sobre a produção distribuída de software, delineadas por L´Erário (2009) e o *framework* Scrum (vide seção 3).

L´Erario (2009) caracteriza a produção distribuída em estados e elementos que indicam a dinâmica produtiva de um produto caracterizado como software (vide Figura 2). Os estados e elementos se relacionam diretamente com o produto e com a unidade de produção de software (fábricas de softwares, *software house*, institutos de pesquisa). Neste texto estas unidades são delineadas como *sites* de Produção.

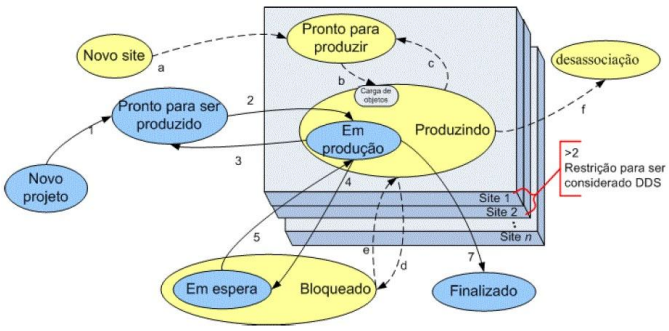


Figura 2 – Modelo de produção distribuída de software (retirado de L´Erario 2009)

Segundo L´Erario (2009), na Figura 2, o estado *Novo site* indica que um novo nó da rede produtiva está pronto para associar-se. Já o estado *Pronto para produzir*, caracteriza que o *site* já se encontra na rede, ou seja, um acordo produtivo entre dois ou mais *sites* foi celebrado.

Assim que uma ordem de serviço (conjunto de requisitos ou componentes de software) for colocada em produção, *site* pode atingir o estado *Produzindo*. Perceba que o *site* também pode transitar do estado de *Produzindo* para *Pronto para Produzir,* este fato ocorre quando existe possibilidade do *site* atender novas demandas de produção. O *site* pode atingir o estado de *Bloqueado*, caso ele não tenha possibilidade de atender mais aos critérios de qualidade (de produção) impostos no acordo celebrado pela rede; não conseguir atender demandas tecnológicas ou contratuais (possibilidade da entrega de um determinado produto (ou subproduto) dentro dos prazos e custos pré-estabelecidos) de um novo produto de software ou

estiver aguardando um artefato de produção para continuar o seu trabalho (relação de dependência de artefatos). É importante salientar que o *site* pode transitar do estado *Bloqueado* para *Produzindo*, caso ele atenda novamente aos critérios de qualidade impostos ou possuir condições de suprir as demandas contratuais ou tecnológicas de um determinado projeto. O *site* pode solicitar a sua *Desassociação* da rede, por meio, do término ou quebra do acordo.

Além de mapear estados e elementos de um *site*, a Figura 2 também caracteriza estados e elementos do projeto (de software). Dentro deste contexto é possível perceber que um *Novo projeto* quando concebido é fatiado em ordens de serviços (OS)[2] e atingi o estado *Pronto para ser produzido*. As ordens de serviços são direcionadas a um determinado *site* e colocadas *Em produção*. Assim que a ordem de serviço é concluída o estado *Finalizado* é caracterizado. É factível que a produção de uma determinada OS (a) dependa de outra (b), que encontra-se em produção em outro *site*, neste momento a OS (a) atinge o estado de *Espera* - o processo não segue o fluxo enquanto a OS não receber os artefatos gerados pela OS (b). Ao receber os artefatos da OS (b), o processo produtivo volta a seguir o seu fluxo – o estado *Em produção* é novamente caracterizado.

Conforme citado anteriormente, a proposta do DDScrum é alicerçada pelo modelo delineado por L´Erario (2009) e pelo *framework* Scrum. O DSScrum é caracterizado por estados e elementos associados à produção de software (elementos de processo e de produto) – Vide Figura 3.

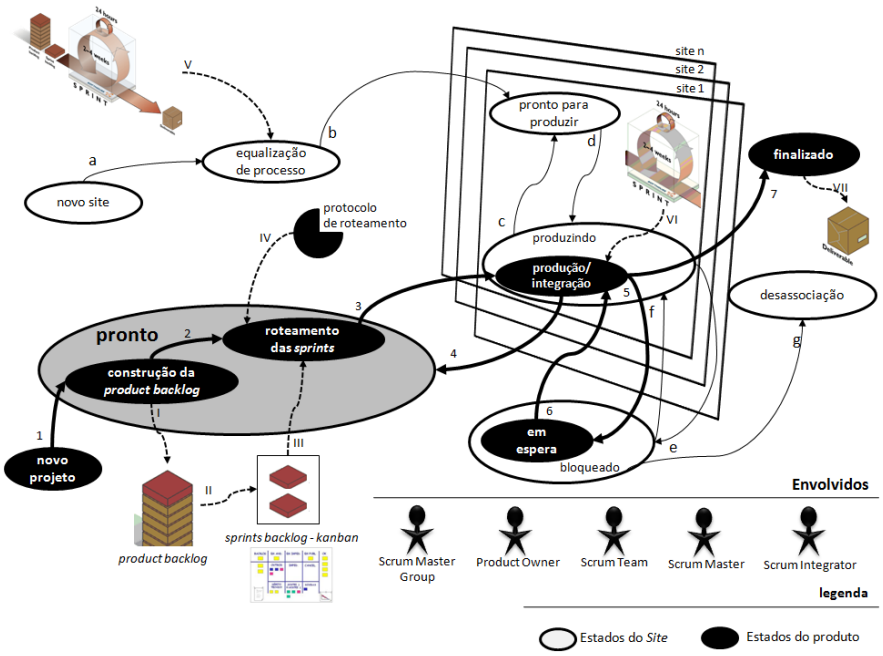


Figura 3 – Proposta do DDScrum

Ao analisar a referida Figura é possível perceber a presença dos estados: *equalizando processo*, *pronto para produzir*, *produzindo*, *bloqueado*, *desassociação*, *pronto* dividido em dois subestados (*construção da product backlog* e *roteamento das sprints*), *produção/integração*, *em*

[2] Uma ordem de serviço reuni um ou mais requisitos de software que serão produzidos por uma determinada entidade.

espera e *finalizado*. Para agilizar a leitura a descrição dos estados, assim como as suas transições foram descritas na Tabela 1.

Tabela 1 – A transição de estados do DDScrum

Estado origem	T[3]	Estado destino	Descrição
novo site	a	equalizando processo	Um novo *site* se candidata a participar da rede de produção.
equalizando processo	b	pronto para produzir	O ***Scrum Master Group*** (***Scrum Master*** de cada *site*) – ***SMG*** - verifica se o *site* candidato a rede de produção possui consistência no processo de produção de software. Celebração de um acordo de produção entre os nós. Após celebração o ***Scrum Master Group*** é acrescido do ***Scrum Master*** do site candidato.
pronto para produzir	d	produzindo	O *site* possui o processo consistente e encontra-se pronto para iniciar a produção das ordens de serviços (OS).
produzindo	c	pronto para produzir	O site recebe uma ordem de serviço e produz artefatos ou subprodutos relacionados ao projeto de software. Finalizada a produção o *site* retorna ao estado anterior.
produzindo	e	bloqueado	O *site* não tem possibilidade de atender mais aos critérios de qualidade (de produção) impostos no acordo celebrado pela rede; não consegue atender demandas tecnológicas ou contratuais (possibilidade da entrega de um determinado produto (ou subproduto) dentro dos prazos e custos pré-estabelecidos) de um novo produto de software ou está aguardando um artefato de produção para continuar o seu trabalho (relação de dependência de artefatos).
bloqueado	f	produzindo	O *site* volta a ter possibilidade de atender aos critérios de qualidade (de produção) impostos no acordo celebrado pela rede; consegue atender demandas tecnológicas ou contratuais (possibilidade da entrega de um determinado produto (ou subproduto) dentro dos prazos e custos pré-estabelecidos) de um novo produto de software ou recebe um artefato de produção para continuar o seu trabalho.
bloqueado	g	desassociação	O *site* permanece por um período longo no estado de bloqueado devido ao não atendimento aos critérios de qualidade impostos pela rede. Este fato leva a *desassociação* do *site* perante a rede de produção. O site pode solicitar a sua dessassociação da rede a qualquer momento.
novo projeto	1	pronto: construção da *product backlog*	Um novo projeto de software é captado pela rede de produção. O ***Scrum Master Group***, constrói o *product backlog.*
pronto: construção da *product backlog*	2	pronto: roteamento das *sprint*	De posse do *product backlog,* o ***Scrum Master Group*** caracteriza as *sprints* – ordens de serviços. Após a caracterização estas *sprints* serão roteadas para os *sites*. Importante: este roteamento segue um protocolo. Esse protocolo deve ser customizado de acordo com as características de cada rede produtiva.
pronto: roteamento das *sprint*	3	produção/integração	As *sprints* são colocadas em produção. A produção e a integração dos componentes de software aglutinados nas *sprints* são de responsabilidade do ***Scrum Team***, ***Scrum Master*** e do ***Scrum Master Group***.
produção/integração	4	pronto	Ao finalizar a produção de uma *sprint* o ***Scrum Team*** e ***Scrum Master*** estão prontos novamente para estabelecer um novo ciclo de produção.
produção/integração	5	em espera	É factível que a produção de uma determinada *sprint* (a) dependa de outra (b), que encontra-se em produção em outro site, neste momento a *sprint* (a) atinge o estado de espera.
em espera	6	produção/integração	Ao receber os artefatos da *sprint* (b), o processo volta a seguir o seu fluxo – o estado em produção é novamente caracterizado – neste momento o ***Scrum Team*** e ***Scrum Integrator*** integram o produto gerado na *sprint* (b) a *sprint* (a).
produção/integração	7	finalizado	Assim que a *sprint* é concluída o estado finalizado é caracterizado. Lembrando que neste momento o ***Product Owner*** recebe um *release* executável do software.

5. Métodos e Procedimentos Aplicados para Validar o Modelo

Para validar o modelo foi aplicado o método de pesquisa experimental. O referido método realiza teste das hipóteses por meio de um experimento controlado, projetado de forma a produzir dados necessários, podendo ser realizado em laboratório ou no próprio campo.

A utilização do método prevê a execução das seguintes atividades: a) Definição da hipótese. b) Concepção do protocolo experimental. Conjunto de regras ambientais e comportamentais na qual se enquadra o experimento. c) Execução do experimento. d) Análise dos resultados (mapeado em uma seção específica devido a sua importância).

a. Definição da hipótese: Para verificar a aplicabilidade do modelo, os autores deste trabalho definiram a seguinte hipótese: **H1**: É possível criar um modelo que alinhe a idéia de desenvolvimento distribuído de software e o *framework* Scrum.

[3] T significa Transição dos estados.

b. Protocolo experimental

Para a concepção do protocolo experimental é necessário:

- Definir o ambiente do experimento. Nesta etapa os pesquisadores devem responder a seguinte questão: O experimento será realizado no laboratório ou no próprio campo do conhecimento?
- Configurar o ambiente: A execução deste passo prevê: 1) Definição das entidades envolvidas no experimento (pessoas, software ou componentes). 2) caracterização das entidades (idade, formação, local de trabalho...); 3) definição da amostra (quantidade de entidades envolvidas no experimento). 4) definição da forma de coleta das informações (aplicação de questionário, observação direta das entidades, avaliação dos resultados gerados segundo um conjunto de critérios.). 5) Avaliação das informações (as informações geradas possuem consistência, são passíveis de generalização?).

As informações inerentes ao protocolo experimental podem ser verificadas no Quadro 1.

Apresentado os métodos e procedimento utilizados neste trabalho, a próxima seção irá descrever a aplicação do DDScrum.

0. Ambiente: Laboratório de Engenharia de Software da Universidade Tecnológica Federal do Paraná – Campus Cornélio Procópio.
1. Entidades: Gerentes de projetos de empresas de desenvolvimento de software que desejam formar uma rede de produção. Estes gerentes foram treinados pelos autores deste trabalho no primeiro semestre de 2013.
2. Caracterização das entidades: Gerentes com formação em Análise e Desenvolvimento de Sistemas, Ciência da Computação e Engenharia da Computação.
3. Definição da amostra: 4 gerentes de 4 empresas – todos com mais de 3 anos de experiência no cargo.
4. Forma de coleta das informações: Observação direta das entidades (durante o treinamento) e depoimento das entidades após a participação do DDScrum. O desenvolvimento do experimento foi realizado em salas separadas. As *sprints* no Scrum variam entre 2 a 4 semanas, no experimento este tempo foi customizado entre 2 a 16 horas.
5. Avaliação: A avaliação será delineada a partir da produção de um software utilizando o DDScrum

Quadro 1 – Protocolo experimental

6. A aplicação do DDScrum

O DDScrum foi aplicado no treinamento, efetuado no mês de maio de 2013, de 4 gerentes de 4 empresas de produção de software. É importante ressaltar que o objetivo destas empresas é compor uma rede de produção.

Durante o treinamento os autores deste trabalho se posicionaram como o *Scrum Master Group – SMG,* apresentaram o DDScrum para os gerentes - estado *equalização do processo* – configuraram dois *sites* de produção, cada um deles com 2 gerentes de projetos – estado *pronto para produzir*. É importante salientar os *sites* foram alocados em salas distintas e comunicação só poderia ser feita por: e-mail, *call-conference* (utilização de um chat) ou telefone celular. Somente os membros do *SMG* tinham acesso direto aos *sites*.

Além de percorrer os dois estados citados no parágrafo anterior, os autores deste trabalho definiram, a partir de um projeto de software (estado *novo projeto*[4]), uma *product backlog*[5] (estado *construção da product backlog*). Além deste artefato os autores do trabalho também definiram o diagrama de entidade e relacionamento (DER[6]) e o modelo de interface[7]. Todos estes artefatos foram publicados nos endereços listados no rodapé desta página.

[4] Vide características em https://dl.dropboxusercontent.com/u/3525445/ddscrum/objetivos.pdf
[5] Vide *product backlog em* https://dl.dropboxusercontent.com/u/3525445/ddscrum/pb.pdf

[6] Vide DER em https://dl.dropboxusercontent.com/u/3525445/ddscrum/der.png

[7] Vide modelo de interface em https://dl.dropboxusercontent.com/u/3525445/ddscrum/interface.pdf

Após a construção dos artefatos, os membros do *Scrum Master Group* se reuniram e iniciaram a configuração das *sprints*, fato este que proporcionou o *roteamento das mesmas*. O *SMG* também configurou a gestão global do projeto utilizando o Kanban. Neste momento os *sites* iniciaram a sua *produção*. Enquanto permaneceram neste estado, os sites mapearam – via kanban: a) o nome da funcionalidade a ser desenvolvida; b) a data início da implementação; c) a data de término; d) a quantidade de horas utilizadas na produção. Ao atingir o estado de espera – fato este ocorrido pelo *site* 2 ao implementar o item 4 da *product backlog* – o *site* apontava quantidade de horas em estado de espera. É possível visualizar o kanban e as informações geradas pelos *sites* na animação apresentada neste *link*: https://dl.dropboxusercontent.com/u/3525445/ddscrum/sprint.ppsx. É importante salientar ainda que os *sites* durante a *produção* do código aplicaram a técnica de desenvolvimento guiado por testes – técnica esta apresentada durante o treinamento.

Ao terminar a implementação os sites notificavam o *SMG* e estes ficaram responsáveis por integrar as funcionalidades geradas – Informações sobre o processo de integração podem ser visualizadas no log de integração (vide: https://dl.dropboxusercontent.com/u/3525445/ddscrum/log.pdf). Os problemas reportados no log eram apresentados aos sites e as manutenções corretivas eram efetuadas, fato este ocorrido com o *site* 2 que ficou responsável pelo desenvolvimento do 4º item da *product backlog* – o site não respeitou as especificações espelhadas no diagrama de entidade e relacionamento[6].

Por fim, ao integrar as funcionalidades agrupadas nas *sprints* o produto (ou parte dele) atinge o estado finalizado e o site encontra-se novamente pronto para produzir.

7. Análise dos Resultados, Conclusões e Perspectivas Futuras

Os resultados obtidos com a aplicação do DDScrum, levando em consideração o objetivo inicial traçado (possibilidade de unir as prerrogativas delineadas pelo desenvolvimento distribuído de software ao *framework* Scrum), pode ser constatado por qualitativamente nos itens abaixo:

- A *equalização do processo* foi feita durante o treinamento, perceba que o DDScrum foi apresentado aos gerentes logo na fase inicial da execução do experimento. Em um ambiente real de produção esta *equalização* pode necessitar de um esforço maior do *SMG*, visto que o *site* candidato a rede já possui um processo de produção, e este deve ser avaliado e posteriormente equalizado. A mudança cultural no processo não é trivial.
- Para efetuar o *roteamento das sprints* o *SMG*, além da *product backlog*, também tiveram que especificar o modelo de dados[6] e o documento de interface[7].
- A tabela de roteamento não foi caracterizada.
- A proposta de uma ferramenta de gestão de projetos alinhada ao kanban, foi gerada, pelos autores deste trabalho, durante o desenvolvimento do experimento – esta ferramenta será apresentada em um trabalho futuro.
- Perceba que o DDScrum foi aplicado a construção e teste do código fonte, as tarefas relacionadas a atividade de projetos de software foram caracterizadas de forma centralizada pelo *SMG*.
- O estado de desassociação não foi atingido durante o experimento.

- Perceba que *site* 2 ficou duas horas no estado de *espera* – valor este capturado pelo *kanban* - https://dl.dropboxusercontent.com/u/3525445/ddscrum/sprint.ppsx.
- Perceba que as *sprints* foram mapeadas na coluna *to do* do Kanban.
- Foi possível verificar que o DDScrum mapeia todos os requisitos de um ambiente de desenvolvimento distribuído: o ambiente distribuído foi constituído por no mínimo dois sites; o experimento proporcionou a distância física entre eles – cada *site* foi alocado em uma sala; uma granularidade de repasse, que identifica a forma na qual o site entrega seu subproduto para a rede foi mapeado; o processo foi difundido; e grau de interação, que revela o quanto a informação é trocada entre os nós, mediante a quantidade de estímulos enviados entre uma instância e outra foi caracterizada durante a integração do produto - participaram da integração *SMG* e o *Scrum Team* de cada *site.*

Dado o contexto delineado para avaliação do modelo, conclui-se que **H1** (é possível criar um modelo que alinhe a idéia de desenvolvimento distribuído de software e o *framework* Scrum) pode ser caracterizada como verdadeira.

Por fim, cabe aos autores deste trabalho proporcionar a implantação do DDScrum nas 4 empresas e comparar os aspectos quantitativos e qualitativos da produção centralizada e distribuída.

8. Referências Bibliográficas

Karolak, D. W. (1998) "Global Software Development – Managing Virtual Teams and Environments". Los Alamitos, EUA. IEEE Computer Society, 159 p.

L'Erario, A. (2009) "M3DS: um modelo de dinâmica de desenvolvimento distribuído de software". 175 p. Tese (Doutorado) – Escola Politécnica da USP. Universidade de São Paulo, São Paulo.

Marquardt, M. J. e Horvath, L. (2001) "Global Teams: how top multinationals span boundaries and cultures with high-speed teamwork". Davies-Black Publishing. Palo Alto, EUA. 246 p.

Prikladnicki, R. e Audy, J. L. N. (2006) "Uma análise comparativa de práticas de Desenvolvimento Distribuído de Software no Brasil e no exterior". In: XX SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. Florianópolis: SBES. p. 255 – 270

Sutherland, A. *et. al*, (2007) "Distributed SCRUM: Agile Project Management with Outsourced Development Teams" in HICSS'40, Hawaii International Conference on Software Systems, Big Island, Hawaii, 2007.

Analisando as Contribuições da Comunidade Open Source Brasileira em Projetos Distribuídos de Software Um Estudo Inicial

Gustavo Pinto[1], **Fernando K. Kamei**[1,2]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50.740-560 – Recife – PE – Brasil

[2]Instituto Federal do Sertão Pernambucano (IF Sertão-PE - Campus Ouricuri)

{ghlp, fkk}@cin.ufpe.br

Abstract. *In the last years, the software development has demanded improvements on techniques and tools in order to facilitate the communication of distributed teams. In this context, social coding environments have gained attention of researchers and mainly of open source communities, due to the fact that its development process is necessarily distributed. This study aimed to understand how the contributions are made by Brazilians users in a distributed software development open source environment. We analyzed the activities of more than 4.000 developers and more the 15.000 projects during one year. The results show that the Brazilian community has weak activity, although of performing various types of contributions.*

Resumo. *Nos últimos anos, o desenvolvimento de software tem exigido a melhoria de técnicas e ferramentas que facilitem a comunicação de equipes distribuídas. Nesse contexto, ambientes de social coding vêm ganhando atenção de pesquisadores e principalmente das comunidades open source, devido ao fato do seu processo de desenvolvimento ser necessariamente distribuído. Este estudo objetivou entender como são realizadas as contribuições de brasileiros em ambientes distribuídos de software open source. Foram analisadas as atividades de mais de 4 mil desenvolvedores, e mais de 15 mil projetos no período de um ano. Os resultados mostram que a comunidade brasileira é pouco ativa, apesar de realizar vários tipos de contribuições.*

1. Introdução

Como reflexo da globalização dos negócios e do crescimento da economia, ocorreu uma migração do mercado local para o mercado global, criando novas formas de competição e colaboração. Esses fatores também atingiram o mercado de software. Neste contexto, surge o Desenvolvimento Distribuído de Software (DDS), onde as equipes de desenvolvimento estão distribuídas em diferentes localizações [Carmel 1999]. Segundo [Prikladnicki 2012], a distribuição do processo de desenvolvimento de software faz ampliar os problemas inerentes ao desenvolvimento tradicional e gera novos desafios ao adicionar distância física, dispersão temporal e diferenças culturais. Algumas maneiras de enfrentar esses desafios é a adoção de processos, técnicas e ferramentas de suporte ao DDS, que objetivem a efetiva comunicação e coordenação de equipes distribuídas.

Neste cenário de DDS destacam-se as comunidades open source, que emergem através do compartilhamento do código fonte dos projetos, desenvolvidos de maneira colaborativa. Ademais, desenvolvedores tem testemunhado o crescimento de plataformas para contribuição de projetos open source, como Github[1] e BitBucket[2]. Estes ambientes chamados de *social coding* oferecem inúmeras funcionalidades, possibilitando que grupos distribuídos possam trabalhar em equipe em um mesmo projeto, facilitando a colaboração. Dessa forma, o desenvolvimento de projetos open source, através do uso dessas ferramentas, tornou-se intrinsecamente distribuído.

No entanto, enquanto que por um lado o acesso ao código é extremamente facilitado, aspectos como a compreensão das comunidades e a análise das contribuições são tópicos que ainda precisam ser melhores explorados, uma vez que grande parte do sucesso da filosofia open source advém das diversas formas de contribuições das comunidades. Dessa forma, as comunidades e suas contribuições desempenham papel importante no desevolvimento distribuído de software open source.

O presente trabalho tem como objetivo investigar as principais formas de contribuições, bem como o perfil das comunidades open source brasileiras que interagem no *social coding* GitHub. Para tanto, este trabalho responderá as seguintes perguntas de pesquisa:

(i) A comunidade brasileira de desenvolvimento distribuído open source é ativa? Este estudo classificou a atuação da comunidade brasileira como pouco ativa. Os dados mostraram que apenas 30% dos usuários brasileiros registrados no GitHub tiveram ao menos uma contribuição durante o período de um ano. No entanto, deste grupo de contribuidores, cerca de 40% realizaram 20 ou mais contribuições ao longo de um ano. Supreendentemente, este estudo identificou que a maioria dos usuários ativos no GitHub são da Região Nordeste (46% do total), além de participarem, em sua maioria, da iniciativa privada (83,65%).

(ii) Quais são as formas de contribuição mais comumente realizadas? Este estudo analisou as principais contribuições públicas que os desenvolvedores podem fazer no GitHub. Destas, como esperado, as mais utilizadas por brasileiros são os commits (95% do total), seguidos de pull requests (4% do total), e abrir e fechar issues (1% do total), respectivamente. Ademais, foi observado que boa parte dos projetos contribuídos são ativos, ou seja, continuam recebendo contribuições ao longo dos meses.

O restante do trabalho está estruturado da seguinte maneira: na Seção II são apresentados informações preliminares sobre o GitHub; na Seção III é apresentado as questões de pesquisa, e como a pesquisa foi conduzida para realizar a coleta, e análise dos dados; na Seção IV são apresentadas os resultados e discussões para as questões de pesquisa; na Seção V é apresentada uma discussão sobre trabalhos relacionados; por fim, na Seção VI apresentamos as conclusões e os trabalhos futuros.

2. Background

Ambientes de *social coding* têm fornecido oportunidades para que pesquisadores possam responder diversas questões ligadas ao desenvolvimento de software. Durante os últimos

[1]http://www.github.com
[2]http://www.bitbucket.org

anos, o GitHub se tornou uma das mais populares plataformas de colaboração em projetos de desenvolvimento distribuído de software [Gousios and Spinellis 2012], contendo mais de 4 milhões de repositórios de software, e mais de 10 milhões de desenvolvedores.

Criado em cima do sistema de versionamento de código `Git`, adicionou funcionalidades que não estão presentes no software base. No GitHub, um dado desenvolvedor pode participar de vários projetos e cada projeto pode ter mais de um desenvolvedor. Ainda é possível utilizar de recursos como wiki, issues e review de código. Dentro do GitHub existem páginas para desenvolvedores e páginas para usuários[3]. A página de usuário inclui informações sobre as contribuições realizadas, bem como a quantidade seguidores, a quantidade de desenvolvedores que ele segue, o número de projetos forks (projeto que é uma cópia independente de um outro projeto), o número de projetos que ele está acompanhando (watching), dentre outros. Demais tipos de contribuições podem ser consultados através de uma API pública[4]. Esta transparência é um dos pontos fortes de ambientes de *social coding*.

O termo "contribuição" pode ser utilizado para expressar inúmeras atividades do processo de desenvolvimento de software. Dentro da plataforma de *social coding* GitHub, os desenvolvedores podem realizar outros tipos de contribuições, como comentar dentro de uma issue ou em uma específica linha de código. Para este trabalho, porém, apenas as contribuições públicas[5] foram analisadas: (i) Commits, (ii) pull requests, e (iii) abrir e fechar issues. A seguir, é apresentado o conceito de cada contribuição:

- **Commit**: é uma modificação realizada em um artefato do projeto, usualmente o código fonte. Em um commit pode haver inúmeras modificações, mas é recomendado que apenas as modificações relacionadas a uma determinada atividade sejam agrupadas em um mesmo commit.
- **Pull Request**: é a modificação realizada por um usuário (commit) que é enviada para os mantenedores do projeto, para que estes avaliem e, por ventura, aprovem ou não a modificação. Se aprovada, a modificação será incorporada ao repositório oficial do projeto. Um pull request pode agrupar um ou mais commits.
- **Abrir/Fechar Issue**: é o controle de atividades (correções de bug, melhorias, novas funcionaldiades, etc.) que devem ser adicionadas ao projeto. Quando novas, quando uma nova issue é adicionada ao projeto, esta é classificada como *open*, e quando resolvida, e quando resolvida, seu status muda para *closed*.

3. Metodologia

Este trabalho está interessado em descobrir como os desenvolvedores brasileiros contribuem em projetos distribuídos de software open source. Para atingir este objetivo, as seguintes questões e sub-questões de pesquisa serão respondidas:

1. A comunidade brasileira de desenvolvimento distribuído de software open source é ativa?
   (a) Quais são as características dos projetos contribuídos?
   (b) Quais são as comunidades mais ativas?
2. Quais são as formas de contribuição mais comumentes realizadas?

[3]Um exemplo de página de desenvolvedor pode ser em visualizada: https://github.com/gustavopinto
[4]http://api.github.com
[5]https://help.github.com/articles/why-are-my-contributions-not-showing-up-on-my-profile

3.1. Coletando Dados do Github

Este trabalho realizou a coleta dos dados em duas etapas. A primeira, responsável por identificar os usuários brasileiros, e a segunda, para coletar as informações destes usuários. Para identificar os usuários brasileiros, foi realizada uma busca através do campo de usuário *Location*, que pode pode ser preenchido com a sua localidade. Exemplos de localidades brasileiras incluem "Brasil", "Brazil", e "São Paulo".

Uma limitação do GitHub é que por mais que a busca retorne um total de, por exemplo, 20.000 desenvolvedores para uma dada localidade, apenas os primeiros 1.000 registros podem ser coletados. Dessa forma, não é possível coletar informações de todos os usuários de uma localidade. Uma forma encontrada para minimizar este problema foi modificando o valor inserido no campo *Location*. Por exemplo, todos os nomes dos Estados e das capitais brasileiras foram utilizados como entrada para busca. Ao fim do processo, os usuários duplicados foram removidos, e o total de 4.481 perfis de usuários foram coletados.

Posteriormente, as contribuições destes desenvolvedores foram coletadas utilizando os seguintes passos: para cada desenvolvedor, é feito um request para a página de perfil deste usuário, então é feito o download do conteúdo da página web e são extraídas informações como o número de seguidores, projetos e contribuições. Os dados deste estudo estão baseados nas contribuições realizadas entre 16/05/2012 e 15/05/2013.

4. Resultados e Discussões

Nesta seção são apresentados os resultados que permitem responder as questões de pesquisa.

4.1. Questão 1: A comunidade brasileira de desenvolvimento distribuído de software open source é ativa?

Foram identificados 4.481 usuários brasileiros, destes, 3.121 (cerca de 70% do total) usuários não possuíam nenhuma contribuição no período analisado, sendo classificados como inativos. Os demais usuários, os ativos, foram então sub-divididos em três novos grupos: pouco ativos (até 19 contribuições em um ano), razoavelmente ativos (até 49 contribuições em um ano) e muito ativos (acima de 49 contribuições em um ano). A escolha destes valores teve como referência o resultado do terceiro quartil (41 contribuições) do total de contribuições. Portanto, o percentual dos grupos de usuários pouco ativos, razoavelmente ativos e muito ativos são, respectivamente, 57,8%, 20,9%, 21,3%, mostrando que a comunidade brasileira de desenvolvimento distribuído de software open source é pouca ativa. As demais análises deste trabalho levam em consideração apenas o grupo de usuários ativos.

4.1.1. Questão 1.1: Quais são as características dos projetos contribuídos?

Foram identificados 15.595 projetos, dos quais 6.290 (40,33%) são forks e 9.305 (59,67%) não são forks. A Figura 1 fornece uma visão geral sobre esses projetos.

Na Figura 1-(a) é possível perceber que, cerca de 75% dos projetos forks analisados foram criados há um pouco mais de 2 anos. Considerando o período de um ano,

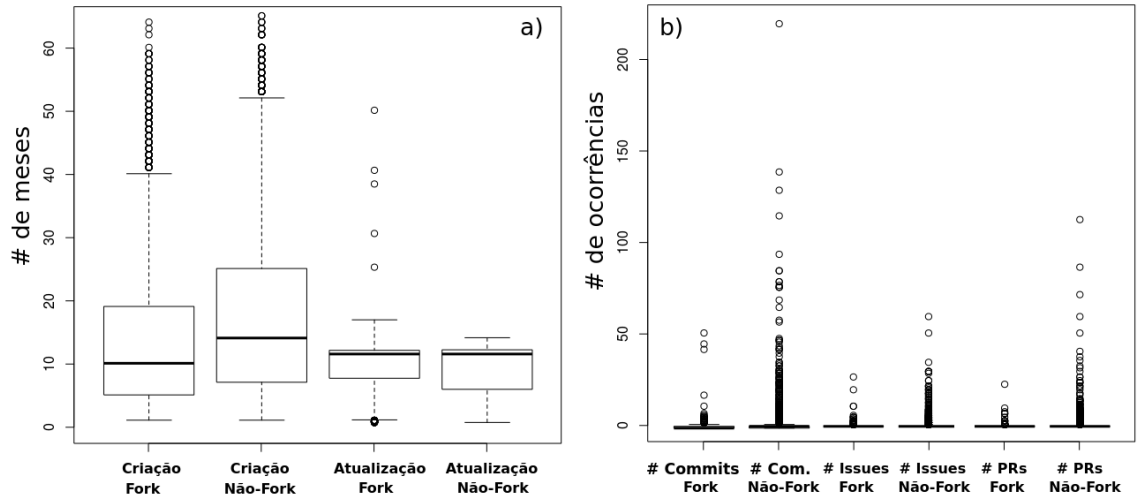


Figura 1. Características gerais sobre os projetos contribuídos ao longo de um ano. A Figura a) tem informações sobre as datas de criação e última atualização dos projetos em meses. A Figura b) contém as informações sobre a distribuição dos commits, issues e pull requests (PRs).

63,78% dos projetos forks já haviam sido criados, enquanto que para este mesmo período, os não-forks somam 51,59%. Porém, um dado interessante está relacionado a atualização desses projetos. É possível perceber que 75% dos projetos tiveram a sua última atualização há um pouco mais de 2 meses, indicando que, a maioria dos projetos criados receberam ao menos uma atualização recentemente.

Na Figura 1-(b) percebe-se novamente um comportamento bem similar entre projetos forks e não-forks. No entanto, observa-se que os projetos originais (não-forks) recebem mais contribuições, principalmente commits e pull requests, do que de forks, o que era esperado. Em resumo, os projetos originais que tiveram atualização há menos 3 meses representam 29,48%, e 70,52% de 3 a 6 meses. Em contrapartida, 24,15% dos projetos forks tiveram atualizações há menos de 3 meses e 75,74% de 3 à 6 meses. Esses dados reforçam o fato de que os projetos, ao contrário dos usuários, são ativos, tanto os originais quanto forks.

Também foi possível analisar a popularidade das linguagens de programação, uma vez que o GitHub infere qual a linguagem mais utilizada em um projeto. Levando em consideração os projetos originais e forks, as 10 linguagens mais utilizadas são: JavaScript (22,77%), Ruby (18,78%), Java (14,19%), PHP (13,95%), Python (9,86%), C (3,36%), C++ (2,94%), Shell (2,55%), Objective-C (1,95%) e C# (1,88%).

A partir desses resultados, é possível observar que, apesar dos desenvolvedores contribuirem principalmente com linguagens consolidadas como JavaScript e Java, novas comunidades, como Ruby, vem ganhado espaço. Um dos motivos pelo qual JavaScript apareceu como a mais utilizada é devido ao fato de que muitos projetos são web, e utilizam-a para manipulação de página. No entanto, é comum que estes projetos também utilizem de outras linguagens na camada de negócio do código. Dessa forma, JavaScript pode ser facilmente utilizada com uma sub-linguagem em um projeto.

Finalmente, uma análise de popularidade dos projetos foi realizada. Este trabalho

considerou que, para um projeto ser considerado popular, o projeto deveria possuir ao menos uma issue (aberta ou fechada), um download, mais que 3 watchers e pelo menos um fork. Com isto, 836 projetos foram classificados como populares. Destes, apenas 65 (7,78%) projetos são forks, o que mostra que alguns forks foram capazes de evoluir a ponto de se tornar reconhecido. Ao total, o número de projetos populares é de 8,28% (projetos originais) e 1,03% (forks).

4.1.2. Questão 1.2: Quais são as comunidades mais ativas?

Ao ser analisada a localidade da comunidade brasileira open source, 46% desses estão localizados na Região Nordeste, 25% na Região Sul, 12% no Sudeste, 12% no Centro-Oeste e apenas 5% no Norte. Em uma análise por Estado, o maior número de usuários ativos encontra-se no Paraná (13,65%), seguido do Ceará (12,37%) e Pernambuco (11,25%). A Figura 2 apresenta os dados das contribuições por Estado.

É importante ressaltar que 92% dos usuários ativos preencheram o campo localidade da sua página de perfil, o que torna estes resultados confiáveis. Levando em consideração o total de contribuições por Estado, Pernambuco é o que mais tem contribuído, com 16,46% das contribuições realizadas no período analisado. Seguido pelos Estados do Paraná (10,63%), Bahia (10,43%), Ceará (10,27%) e do Distrito Federal (10,13%). Os demais Estados juntos somaram 42,04% do total.

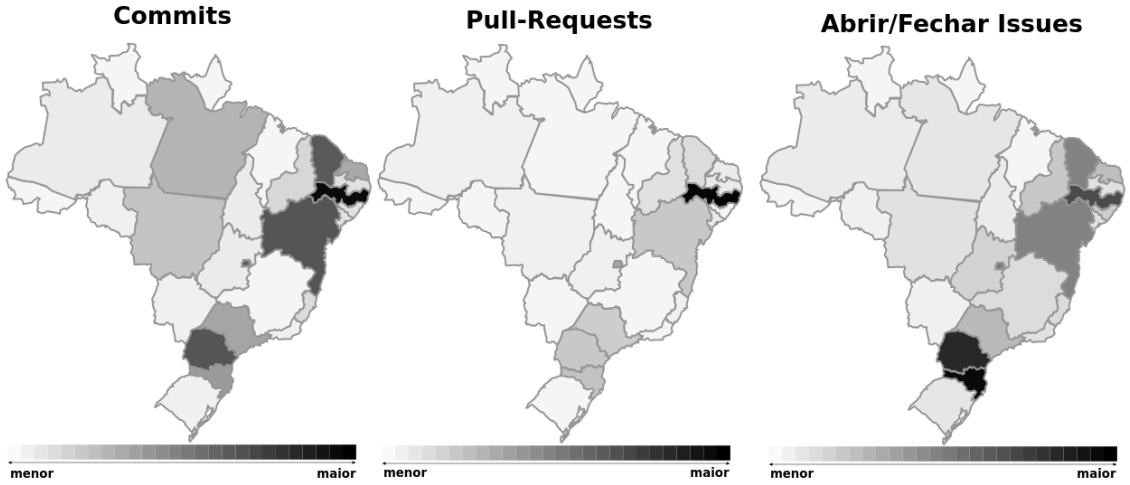


Figura 2. Dados demográficos das contribuições por Estado.

Nesse estudo também foi possível mapear as comunidades com base nas linguagens de programação mais utilizadas pelos usuários, uma vez que o GitHub infere qual a linguagem mais utilizada por um usuário. As 10 linguagens inferidas pelo GitHub como mais utilizadas foram: JavaScript (18,97%), Java (17,42%), Ruby (15,88%), PHP (13,23%), Python (7,72%), C++ (3,01%), C (2,94%), C# (1,98%), Shell (1,83%) e Objective-C (1,32%).

Este resultado é similar ao apresentado para os projetos, com algumas ressalvas. Apesar da linguagem Ruby apresentar maior número de projetos do que a linguagem Java, o número de desenvolvedores é menor, sugerindo que a comunidade Ruby é mais

ativa com relação a comunidade Java. Do total, 10,51% dos usuários não tiveram sua linguagem inferida. Finalmente, foi feita uma análise para identificar o vínculo educacional/empregatício do usuário. Foi observado que, apenas 716 informaram algum tipo de vínculo. Destes, 83,65% estavam vinculados a uma empresa privada, e os demais 16,34% estavam relacionados a algum órgão governamental, destes a maioria ligados a universidades públicas.

4.2. Questão 2: Quais são as formas de contribuição mais comumentes realizadas?

No período analisado, foram identificadas um total de 58.890 contribuições em mais de 15.500 projetos, destas, a sua maioria foram realizadas a partir de commits, representanto 95,50%. As solicitações de pull requests aparecem em seguida, com 3,52% e, por fim, as aberturas e fechamento das issues com 0,98%. A Figura 3 sumariza o total de contribuições por categoria no período.

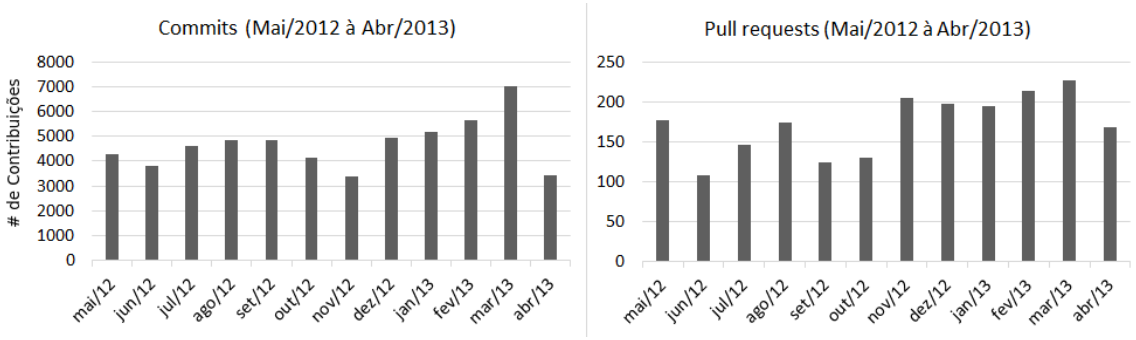


Figura 3. Contribuições por categoria ao longo de um ano.

Como pode ser visto na Figura 3, o número de commits de um dado mês chega a ser 40 vezes maior quando comparado ao número de solicitações de pull request do mesmo período. Uma das principais causas para tal, é o fato de que um único pull request pode conter vários commits. Ademais, durante o período em que os mantenedores do projeto analisam o pull request, outros commits podem ser requisitados para que a feature esteja de acordo com o padrão de codificação adotado.

Um outro ponto importante está relacionado a frequência com a qual as contribuições são realizadas. Diferentemente do esperado, o número de contribuições apresenta um crescimento ao longo do ano, independentemente do tipo da contribuição. Por outro lado, meses como Dezembro e Janeiro aparentam ser atípicos de contribuições, quando observado apenas commits. O gráfico de issues foi omitido pela escala ser cerca de 100 vezes menor que a de commits. De acordo com os resultados apresentados, as maiores contribuições estão relacionadas à commits.

5. Trabalhos Relacionados

A análise de sistemas distribuídos open source é um tópico bastante explorado na engenharia de software nos últimos anos [Gousios and Spinellis 2012]. Muitos destes trabalhos estão interessados em avaliar custos de desenvolvimento [Amor et al. 2006], produtividade e performance do desenvolvedor [Koch and Schneider 2000], e até técnicas de desenvolvimento ágil [Warsta and Abrahamsson 2003]. No entanto, nenhum trabalho que

se interesse sobre as principais formas de colaboração de comunidades open-source em ambientes distribuídos foi encontrado.

Na literatura podem ser encontrados outros trabalhos que analisam ambientes de DDS como uma estrutura de redes. O trabalho mais próximo é o de [Surian et al. 2010], que extraí padrões de colaboração no SourceForge em formato de gráfos com alto nível de detalhamento, a partir da análise de contribuições de atributos não relacionados ao código. No entanto, este trabalho não é capaz de identificar as principais comunidades, seja baseada em locais ou por linguagem de programação.

6. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado um estudo inicial para entender como são realizadas as contribuições de brasileiros em projetos distribuídos de software open source, a partir da análise dos três principais tipos de contribuições públicas do GitHub. Com base nos resultados obtidos é possível afirmar que a comunidade brasileira é pouco ativa: 30% dos usuários são responsáveis por 100% das contribuições. Dentre as comunidades de software, as que mais se destacam ainda estão relacionadas as linguagens mais tradicionais, como JavaScript e Java, além da sua maioria participar da iniciativa privada (84%).

Como trabalho futuro, espera-se analisar outros repositórios de forma a criar relacionamento entre estes dados, a fim de responder perguntas como: (i) Por que determinadas regiões podem estar contribuindo menos?; (ii) Por que determinadas linguagens estão predominando?; (iii) As comunidades também são atuantes em outras plataformas de *social coding*?

Referências

Amor, J. J., Robles, G., and Gonzalez-Barahona, J. M. (2006). Effort estimation by characterizing developer activity. In *Proceedings of the 2006 international workshop on Economics driven software engineering research*, EDSER, pages 3–6.

Carmel, E. (1999). *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Gousios, G. and Spinellis, D. (2012). Ghtorrent: Github's data from a firehose. In *Proceedings on Mining Software Repositories*, MSR, pages 12–21. IEEE.

Koch, S. and Schneider, G. (2000). Results from software engineering research into open source development projects using public data. In *Informationswirtschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversitaet*, page 22.

Prikladnicki, R. (2012). Propinquity in global software engineering: examining perceived distance in globally distributed project teams. *Journal of Software Maintenance*, 24(2):119–137.

Surian, D., Lo, D., and Lim, E.-P. (2010). Mining collaboration patterns from a large developer network. In *Reverse Engineering (WCRE)*, pages 269–273.

Warsta, J. and Abrahamsson, P. (2003). Is open source software development essentially an agile method? In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 143–147.

Analysing Requirements Negotiation in Software Ecosystems with Multi-Agent Systems Techniques

George Valença[1], Carina Alves[2], Patrícia Tedesco[2], Lucas Moreno[2]

[1]Departamento de Informática – Universidade Federal Rural de Pernambuco (UFRPE)
Recife, Pernambuco – Brazil

[2]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife, Pernambuco – Brazil

georgevalenca@deinfo.ufrpe.br, {cfa,pcart,lma5}@cin.ufpe.br

Abstract. *A Software Ecosystem (SECO) can be seen as a distributed network of software companies interacting either in a cooperative or competitive manner, being connected by a shared platform. They participate in complex interrelations and play different roles. This perspective brings several social, managerial and technical challenges. The goal of this paper is to investigate the challenges involved in requirements negotiation within a SECO. In particular, requirements must be negotiated by multiple and distributed actors, who have different and sometimes conflicting expectations. This paper starts a discussion on requirements negotiation strategies for a SECO by drawing concepts from Multi-Agent Systems field.*

1. Introduction

Software Ecosystem (SECO) sheds light on the co-evolutionary relationships of organisations in the software industry. Software companies become connected to a wider business strategy and share a technological platform. According to Jarke and Lyytinen (2010), this perspective shifts from projects where systems are developed from scratch to a broader landscape of integration of existing applications and interdependent networks of developers and users.

Santos et al. (2012), and Campbell and Ahmed (2010) stress that SECO approach impacts on the traditional Software Engineering (SE) models and requires novel processes.Accordingly, it brings new challenges to Requirements Engineering (RE). By putting together multiple and dispersed actors, SECO approach hampers requirements elicitation, communication and negotiation, state Manikas and Hansen (2013). In particular, Bosch and Sijtsema (2010), Fricker (2009), Kazman and Chen (2010) highlight that negotiation activities face constraints such as:

- Actors in an ecosystem are diverse and globally distributed, which causes the inefficiency of negotiation activities that rely on face-to-face communication or that assume interaction between teams;
- The majority of requirements are defined by participants of the ecosystem, rather than elicited from users. Requirements emerge through communications amongst the actors of the ecosystem, whose expectations and goals must be understood to align needs with solutions;

- Since players must create value while providing innovation within the ecosystem, requirements definition cannot be disconnected from business processes governing the ecosystem (e.g. product management). Hence, business strategy of SECOs must guide the negotiation and prioritisation of requirements that will turn into product functionalities.

Recent research in SECO has examined the relationships among stakeholders and communication aspects during RE (Fricker (2009, 2010), Knauss et al. (2012)). However, there is no proper guidance on how organisations should conduct RE activities in a SECO, states Huang et al. (2013). This paper explores this issue by focusing on requirements negotiation along ecosystem's evolution from a social dimension. We start a discussion on how to conceive requirements negotiation strategies for SECO inspired by Multi-Agent Systems (MAS) field. Briscoe (2010) claims that any ecosystem can be analysed making use of MAS concepts. These establish a basis to define the key properties of an ecosystem and allow one to reason about its functioning.

The outline of this paper is as follows. Section 2 provides an overview of SECO field and reports on related work. Section 3 outlines our proposal for requirements negotiation within SECO. Section 4 discusses the use of concepts from MAS to develop requirements negotiation strategies for SECO. Finally, Section 5 concludes the paper and presents future research.

2. Background and Related Work

The emergent notion of SECO is described by Bosch (2009) as *"a set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions"*. This concept is gaining popularity amongst large organisations and relies on the adoption of a common technological architecture for multiple product development; state Campbell and Ahmed (2010). Examples of SECOs are the closed commercial oriented Apple and Microsoft, and the open social oriented Eclipse and Drupal.

In a SECO, organisations participate in complex interrelations and play different roles. Hanssen (2012) describes three fundamental role types in a SECO. The keystone is one organisation or a small group that leads the development of the central software technology. Niche players are third party organisations that use the central technology as a platform to develop related solutions or services. Finally, end-users of the central technology need it to carry out their business.

According to Campbell and Ahmed (2010), a SECO can be examined from three perspectives. In its business dimension, we can analyse factors such as vision, innovation and strategic planning. It also encompasses profit and financial revenues of the companies. Its architectural dimension is concerned with Domain Engineering, Product Line Architecture, and Commonality & Variability Management. Finally, SECO social dimension is related to the actors in the cooperative development environment. It addresses the relationships among companies in the ecosystem.

The decentralised and collaborative nature of SECOs has its roots in Distributed Software Development. In this field, Daniela Damian's evolved the understanding of distributed negotiations by exploring the use of mixed media for requirements selection.

Damian et al. (2008) discusses the combination of synchronous and asynchronous media during negotiations. It evidences that negotiations are more effective when asynchronous discussions of requirement issues are undertaken prior to synchronous negotiation meetings. In its turn, Seyff et al. (2005) proposes an approach based on the EasyWinWin technique for distributed contexts. It promotes active negotiation among success-critical actors to improve cooperation in decision-making, with the support of the ARENA (Anytime, Anyplace REquirements Negotiation Aids) application.

Kukreja (2012) considers the increasing popularity of Facebook to conceive an innovative way for collaborative requirements elicitation and management. The tool is named Winbook and was essentially designed to follow the WinWin negotiation process. It is based on the social networking paradigm and email organization using labels. Alimazighi and Boumahdi (2011) investigates the complexity of RE in a scenario of networked organisations. The study adopts MAP process models to comprehend the multiple goals that arise from interconnected companies. A collaboration model between common and shared goals is established, supporting requirements analysis.

In SECO field, Samuel Fricker contributed to the analysis of RE practices. Fricker (2009) proposes a modelling notation based on negotiation and network theory to describe a requirements communication network in a SECO. The approach focuses on the relationships among interdependent players that need to collaborate and agree with each other to bring new products and systems to success. Fricker (2010) introduces the notion of "requirement value chain". It describes connections among several actors and requirements information diffusion from end-users or customers to the ecosystem.

The former proposals seek to define mechanisms that foster interaction of stakeholders to align goals and priorities. However, there is a need to examine actors and relationships properties in a SECO to evaluate their influence on a shared decision-making. We believe that exploring this gap we may pave the way for SECO success.

3. Towards a Requirements Negotiation Model for SECO

Requirements negotiation involves a decision-making process conducted by several different actors along SECO lifecycle. Based on Moore (1993), such evolution is composed by four stages. At birth, the ecosystem focuses on defining customer requirements. Players join the SECO to participate in the conception of products and services that meet market needs. During expansion, the SECO experiences the growing of the software platform and customer base, with disputes among rival ecosystems. At leadership, the SECO proves to be profitable and internal disputes emerge among participants to get more power. Lastly, the self-renewal stage aims to keep the ecosystem strong by increasing its capacity to innovate and adapt to changes.

In this networked scenario organisations become part of a value chain and the decisions taken should be aligned with the Software Platform Management, states Peeters (2012). This framework consists of four macro processes: Portfolio Management, Roadmap Definition, Release Planning, and Requirements Management. The later includes requirements gathering and prioritisation by relevant stakeholders.

Given this context, our premise is that an appropriate requirements selection directly contributes to ecosystem success. Our intention is to deeply understand the

particularities of RE in SECO to develop a Requirements Negotiation Model (Figure 1). This model aims to provide a group of strategies that support the decision-making involved in negotiation activities. Its initial structure is established over three dimensions, which are following presented:

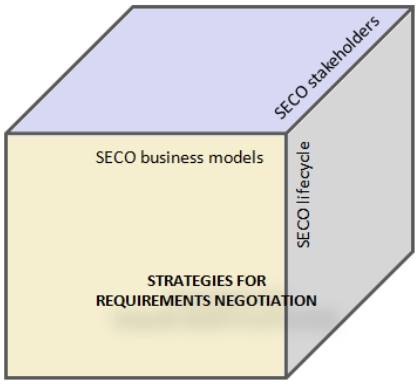


Figure 1. High-level representation of the model initial structure

- **SECO stakeholders**: it explores the relations among organisations and their influence on requirements definition in a SECO. This dimension shall describe the origins and flows of influence in requirements negotiation, as well as the steps taken by stakeholders during this activity. It must also detail the factors that should be considered during decision-making, having stakeholders' goals and constraints as parameters.
- **SECO lifecycle**: it analyses requirements negotiation along lifecycle phases. This dimension will depict how negotiation strategies assist the ecosystem to thrive by considering the health elements productivity, robustness and niche creation, as defined by Iansiti and Levien (2004). It shall explore how strategies collaborate to enhance products performance and shape SECO evolution.
- **SECO business models**: this dimension must consider the varied dynamics of cooperation and competition among organisations to propose negotiation strategies. Valença and Alves (2013) present one possible classification for SECO as commercial or social environments. The iPhone ecosystem illustrates the first type, where suppliers, external integrators and customers are related via financial transactions. The later type concerns communities such as Eclipse, where a consortium represents the wishes and commands of the members.

The requirements negotiation model shall be integrated in the Software Platform Management. Peeters (2012) presented this framework as an evolution of Software Product Management, for organisations with a directed SECO approach. However, it does not provide these companies with mechanisms for requirements negotiation. Hence, we aim to enrich its Requirements Management process by proposing negotiation strategies for SECOs. In the following section we discuss how Multi-Agent Negotiation Techniques can inspire the definition of these strategies. We outline the adaptation of these concepts to RE in SECO.

4. Applying Multi-Agent Techniques in Requirements Negotiation

SECOs are inherently distributed systems, with self-interested components that often need to reach agreements. We can thereby draw concepts from Artificial Intelligence when proposing strategies for requirements negotiation. In particular, Multi-Agent Systems (MAS) area focuses on the relations among several autonomous agents. They interact with each other to either further their own interests or in pursuit of a joint goal, states Woolridge and Wooldridge (2001). Hence, MAS provides a useful background considering its long research on models and techniques for automated negotiation.

In MAS, it is essential to establish means of coordinating efforts. This allows agents to pool their knowledge, goals and skills to solve complex problems, claim Bond and Gasser (1988). When acting in an environment, agents may act either cooperatively or competitively, states Demazeau and Muller (1990). In both situations, it is often the case that agents need to reach a consensus due to a shortage of resources, for instance. They thereby engage in the process of negotiation.

According to Raiffa (1982), negotiation can be seen as the process of making joint decisions. It involves either direct of implicit communication among individuals trying to reach an agreement for mutual benefits. To allow for negotiations in MAS or any social system, three issues should be tackled: (i) all involved parties should be aware of the object under negotiation as well as of its market value; (ii) a common language should be shared by all parties; (iii) a negotiation protocol must be established. Thus, we believe that it is possible to define a parallel between MAS and SECOs.

Depending on the attitudes displayed by components of MAS, different types of negotiation can be employed. We analyse how two of these may support requirements negotiation in a SECO. Bilateral Negotiation and Voting techniques are described in the following sections, which discuss their adoption to define negotiation strategies.

4.1. Bilateral Negotiation

According to Rosenschein and Zlotkin (1994), when only service values are being bargained and the negotiation is on a one-to-one fashion, a Bilateral Negotiation can be used. This type of negotiation involves a negotiation space (the set of possible agreements), a protocol (the rules of encounter) and a strategy (private to each agent).

This approach could be used to define a negotiation strategy for closed commercial SECOs such as the AUTOSAR Tool Platform (Artop). This ecosystem belongs to automotive tool development area and its members have to agree to a license before using the platform, details Weiss (2011). In a bilateral negotiation, the direction of the project and each developer willing to contribute to the platform could be seen as agents. The Artop Software License would act as a protocol, where the proposal of requirements would determine legal moves in the negotiation history. The application of bilateral negotiation steps would lead to an agreement or a conflict.

Strategies using bilateral negotiation shall specify how an agent uses the protocol to get the best possible payoff for themselves. A Monotonic Concession Protocol (MCP) is a form of negotiation where both agents make simultaneous proposals in several rounds, according to Rosenschein and Zlotkin (1994). Agreement is reached if one agent proposes an agreement that is at least as good for the other agent as their own proposal.

As contributors to the ecosystem, here niche players would allow end-user to include functionality in the platform after negotiating versions of the specific application. The keystone would then make a concession considering degree to which the proposal is a differentiating feature for the SECO, for instance.

4.2. Voting

When the system is pressed for time and the attitude prevailing is that of cooperation, a voting scheme can be employed. In this situation, a collective decision is taken, considering an objective holding a central position for the community. Agents are seen as voters, where candidates are decisions that lead to states of the world with well-defined utilities for them. In this scheme, each agent has an order of preference for the candidates. To choose one specific order, it is possible to adopt either (1) a social welfare function, where candidates are ranked considering a degree of preference by agents, or (2) a social function, whose result is a single candidate.

This approach can be appropriate for open social SECOs, given their community oriented development. According to Weiss (2011), products are constructed by using and creating plug-ins in the Eclipse ecosystem. The Eclipse Foundation acts as a keystone. It is responsible for technical infrastructure and coordination of SECO development processes. The direction of development efforts is set by three councils, which are responsible for requirements, planning and architecture. The requirements council collects, reviews and prioritises incoming requirements.

A voting scheme for requirements negotiation in this scenario would consider requirements as candidates, and developers and the councils as voters. Since the requirements council is comprised of strategic members and representatives of project management committee, parameters such as weights, priorities or contexts would be needed. Therefore, a voting could adopt a weighted logic and a combinatorial approach to analyse proposed requirements. These would be democratically selected, although votes would be evaluated considering actors position in the network (power, influence).

5. Conclusion and Future Work

This paper presented a preliminary model for requirements negotiation in SECOs, which is established over three dimensions: SECO stakeholders, SECO lifecycle and SECO business models. The model aims to provide strategies to support the decision-making during requirements negotiation. In this sense, we discussed how Bilateral Negotiation and Voting could be employed to support negotiation activities. We believe that the steps underlying these techniques can provide insights on how to develop such negotiation strategies. This solution can be foundational for better guidance on Requirements Engineering regarding the Software Platform Management.

Possible further research directions include a deeper analysis of how MAS negotiation techniques can be adapted to define strategies for requirements negotiation and decision-making among SECO actors. In addition we shall analyse primary studies from the systematic review in Manikas and Hansen (2013) to gather descriptions of SECOs and understand how negotiation takes place during RE. This will enable us to develop the negotiation model, which should be evaluated through case studies carried out in organisations responsible for open and closed ecosystems.

7. Acknowledgements

George Valença is a doctoral student at Centro de Informática of Universidade Federal de Pernambuco (UFPE), where he receives a scholarship from the Brazilian National Research Council (CNPq) [#141817/2012-7].

References

Alimazighi, Z. and Boumahdi, A. (2011) "Adapting goal oriented approaches in requirement engineering of Inter-Organisational Information System". In Research Challenges in Inf. Science, p.1-7.

Bond, A. H. and Gasser, L. (1988). "Readings in Distributed Artificial Intelligence". Morgan Kaufmann Publishers.

Bosch, J. (2009). "From Software Product Lines to Software Ecosystems". In Int'l Software Product Line Conference, p. 111-129.

Bosch, J. and Sijtsema, P. B. (2010) "From integration to composition: On the impact of software product lines, global development and ecosystems". Journal of Systems and Software, vol. 83 (1), p. 67-76.

Campbell, P. R. J. and Ahmed, F. (2010). "A three-dimensional view of software ecosystems". In European Conference on Software Architecture, p. 81-84.

Damian, D. (2003) "A research methodology in the study of requirements negotiations in geographically distributed software teams". In Workshop on Comparative Evaluation in RE, p. 41-52.

Damian, D., Lanubile, F. and Mallardo. T. (2008) "On the Need for Mixed Media in Distributed Requirements Negotiations". IEEE Transactions on Software Engineering, v. 34 (1), p. 116-132.

Demazeau, Y. and Müller, J. P. (1990) "Descentralized artificial intelligence". North-Holland: Elselvier Science Publishers.

Fricker, S. (2009) "Specification and Analysis of Requirements Negotiation Strategy in Software Ecosystems". In Int'l Workshop on Software Ecosystems, pp. 19-33.

Fricker, S. (2010) "Requirements Value Chains: Stakeholder Management and Requirements Engineering in Software Ecosystems". In Requirements Engineering: Foundation for Software Quality, p. 60-66.

Briscoe, G. (2010) "Complex Adaptive Digital EcoSystems". In International Conference on Management of Emergent Digital EcoSystems, p. 39-46.

Hanssen, G. K. (2012) "A longitudinal case study of an emerging software ecosystem: Implications for practice and theory". In J. Syst. Softw., vol. 85 (7), p. 1455-1466.

Hanssen, G. and Dyba, T. (2012) "Theoretical foundations on software ecosystems". In Int'l Workshop on Software Ecosystems, p. 6-17.

Huang, J. C., Jarke, M., Liu, L. and Lyytinen, K. (2013) "Requirements Management – Novel Perspectives and Challenges", Dagstuhl Reports, v. 2 (10), p. 117-152.

Iansiti, M. and Levien, R. (2004) “Strategy as Ecology”. Harvard Business Review, v. 82 (3), p. 68-78.

Jarke, M. and Lyytinen, K. (2010) “High Impact Requirements Engineering”. Business & Info. Systems Eng., vol. 2 (3), p. 123-124.

Kazman, R. and Chen, H. M. (2010). “The metropolis model and its implications for the engineering of software ecosystems”. In FSE/SDP Workshop on Future of Software Engineering Research, p. 187-190.

Knauss, A., Borici, A., Knauss, E., and Damian, D. (2012) “Towards understanding requirements engineering in IT ecosystems”. In Int’l Workshop on Empirical Requirements Engineering, p.33-36.

Kukreja, N. (2012) “Winbook: a social networking based framework for collaborative requirements elicitation and WinWin negotiations”. In Int’l Conference on Software Engineering, p. 1610-1612.

Lim, S. L., Damian, D. and Finkelstein, A. (2011) “StakeSource2.0: using social networks of stakeholders to identify and prioritise requirement”. In Int’l Conference on Software Engineering, p. 1022-1024.

Manikas, K. and Hansen, K.M. (2013) “Software Ecosystems – A Systematic Literature Review”, Journal of Systems & Software, v. 86 (5), p. 1294-1306.

Seyff, N., Hoyer, C., Kroiher, E. and Grunbacher, P. (2005) “Enhancing GSS-based requirements negotiation with distributed and mobile tools”. In IEEE Int’l Workshops on Enabling Techologies, p. 87-92.

Moore, J.F. (1993) “Predators and prey: a new ecology of competition”. Harvard Business Review, v. 71 (3), p. 75-86.

Peeters, S.A. (2012) “How Software Product Management becomes Software Platform Management in Software Ecosystems”. Master thesis, Faculty of Science Theses.

Raiffa, H. (1982) “The Art and Science of Negotiations”. Belknap Press of Harvard University Press Cambridge.

Rosenschein, J.S. and Zlotkin, G. (1994) “Rules of Encounter”. MIT Press.

Russell, S. and Norvig, P. (2010) “Artificial Intelligence – A Modern Approach”. Pearson Education, 3rd edition.

Santos, R. P., Werner, C., Barbosa, O. and Alves, C. (2012) “Software Ecosystems: Tre006Eds and Impacts on Software Engineering”. In Brazilian Symposium on Software Engineering, p. 206-210.

Valença, G. and Alves, C. (2013) Um Modelo para Negociação de Requisitos em Ecossistemas de Software. In Requirements Engineering@Brazil, p. 74-79.

Weiss, M. (2011) “Economics of collectives”. In Int’l Software Product Line Conference, Article 39.

Woolridge, M. and Wooldridge, M. J. (2001). Introduction to Multiagent Systems. John Wiley & Sons, Inc., New York, NY, USA.

Reflexões sobre Comunicação no Desenvolvimento Distribuído no Contexto de Ecossistemas de Software

Ivaldir Farias[1], Rodrigo Santos[2], Sabrina Marczak[3], Alinne Santos[1], Cláudia Werner[2], Hermano Moura[1]

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

[2]PESC/COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

[3]FACIN – Pontifícia Universidade Católica de Porto Alegre (PUCRS)

{ihfj,accs,hermano}@cin.ufpe.br, {rps,werner}@cos.ufrj.br, sabrina.marczak@pucrs.br

Abstract. *Communication is a critical challenge in Distributed Software Development. It is influenced by several factors and affects the software engineering processes adopted in the project. These factors also influence the development of Software Ecosystems. This paper presents a critical analysis about the factors that affect communication in Distributed Software Development when Software Ecosystems are the scenario of the products under development. We expect such insights to help us further understand the role of communication in such context.*

Resumo. *Um dos desafios mais críticos do Desenvolvimento Distribuído de Software (DDS) é uma comunicação efetiva, impactada por diversos fatores e, ao mesmo tempo, produzindo efeitos positivos ou negativos sobre os processos de Engenharia de Software. Esses fatores podem ser observados também no contexto dos Ecossistemas de Software (ECOSs), onde as arquiteturas e/ou plataformas têm a participação de desenvolvedores externos e usuários. Este artigo analisa os fatores e efeitos que influenciam a comunicação no DDS e discute novos fatores no contexto dos desafios de ECOSs. Espera-se compreender melhor como a comunicação pode ser impactada pelos ECOSs.*

1. Introdução

A construção dos atuais sistemas de software se tornou mais complexa dado que as empresas estão utilizando recursos distribuídos a fim de reduzir custos, atender a novos mercados, explorar o potencial da reutilização, entre outros (Carmel, 1999). No entanto, a distância dificulta a colaboração e a coordenação entre os *stakeholders*, conforme concluído por estudos que atestam que dificuldades como esta se traduzem em processos de desenvolvimento mais lentos (Silva *et al.*, 2010). De acordo com Boehm (2006), na Engenharia de Software (ES), tais problemas e desafios vêm sendo explorados devido à necessidade do tratamento de suas questões econômicas e sociais, como ocorre no Desenvolvimento Distribuído de Software (DDS).

Um dos desafios mais críticos do DDS é a comunicação efetiva, necessária para a colaboração e coordenação de atividades que possuam interdependência. Segundo Santos *et al.* (2012), por fazer parte do cenário da ES globalizada, a comunicação em DDS é impactada por diversos fatores e, consequentemente, produz efeitos positivos ou

negativos sobre os processos de ES. Além disso, a ES globalizada não vem se limitando ao desenvolvimento de projetos ou produtos únicos, mas incluindo o contexto das arquiteturas e/ou plataformas comuns e/ou relacionadas, com a participação de desenvolvedores externos e usuários, conhecidos ou não, na forma de Ecossistemas de Software (ECOSs) (Santos & Werner, 2012). Uma vez que a comunicação também sofre interferências desse novo contexto, este artigo analisa os fatores e efeitos que influenciam a comunicação no DDS, conforme identificados por Santos *et al.* (2012) (Seção 2), e discute novos fatores no contexto dos ECOSs (Seção 3). Realiza-se ainda um mapeamento entre os desafios de ECOSs e os fatores da comunicação em DDS, contribuindo para uma discussão carente netas duas áreas (Seção 4).

2. Comunicação em DDS

A comunicação efetiva é um dos desafios enfrentados em DDS, pois está presente em todo o ciclo de vida de um projeto de software e permeia todos os aspectos do desenvolvimento, tornando-se relevante para o seu sucesso. Por exemplo, Silva *et al.* (2011) relatam que a falta de comunicação tem um impacto alto no sucesso/insucesso de um determinado projeto, visto que é por meio dela que informações são compartilhadas com os *stakeholders* do projeto. Neste sentido, diversos pesquisadores afirmam que a comunicação é fator fundamental para o desenvolvimento de projetos distribuídos (*e.g.*, Carmel, 1999; Trindade *et al.*, 2008). Com o crescimento da adoção do DDS, as equipes vêm vivenciado experiências com profissionais de diferentes níveis técnicos, sociais e culturais. As possíveis soluções para lidar com esse cenário são desafiadoras quando consideradas apenas localmente, onde há comunicação face-a-face. As diferenças de vocabulário, termos técnicos e formas de abordagem social dificultam a comunicação entre integrantes da equipe desempenhando papéis distintos com responsabilidades diversas e focados em vários aspectos do desenvolvimento.

Partindo desta motivação, Santos *et al.* (2012) identificaram através fatores que influenciam a eficácia da comunicação de uma revisão sistemática da literatura, além de discutirem alguns efeitos positivos ou negativos, ocasionados pelos processos de comunicação em DDS. A relação entre os principais fatores e efeitos extraídos de Santos *et al.* (2012) está sumarizada na Tabela 1:

- **F1. Diferenças Culturais**: a cultura define os valores, modos de expressão e estilos de trabalho. Dependendo do contexto cultural, certos hábitos de trabalho (*e.g.*, trabalhar em feriados) podem ser incompreensíveis pelos demais. Culturas muito distintas trabalhando em conjunto podem ocasionar comportamentos conflitantes, mal entendidos e dificuldade em obter confiança;
- **F2. Dispersão Geográfica e Distância Temporal**: muitas equipes distribuídas estão localizadas em diferentes fusos horários, o que torna o processo de comunicação ainda mais difícil devido a possíveis ausências, pela sobreposição de horas de trabalho. Estas diferenças de horário podem acarretar em tempo significativo para os membros obterem uma resposta ou corrigir algum equívoco, levando a atrasos em tomadas de decisões e na realização de tarefas;
- **F3. Idioma e Barreiras Linguísticas**: a diferença do idioma entre as equipes dispersas é vista como uma dificuldade representativa na comunicação, pelo fato

das pessoas terem dificuldades em se comunicar com o restante da equipe, em tempo real, devido à falta do domínio pleno do idioma utilizado;

- **F4. Definição do Meio de Comunicação (síncrono e/ou assíncrono)**: a seleção do meio é importante. A seleção deve ocorrer levando em consideração o contexto em que será utilizada. Se utilizado de forma equivocada, o meio pode impor dificuldades e mal entendidos no compartilhamento de informações, prejudicando o processo de comunicação e o encaminhamento do projeto;
- **F5. Seleção das Tecnologias de Comunicação**: de forma adequada, é importante para o sucesso do projeto, bem como para minimizar as possíveis incompatibilidades na utilização de diferentes ferramentas, tecnologias de informação, linguagens de programação e ambiente de desenvolvimento e plataformas. Portanto, é importante a seleção adequada, pois um equívoco pode levar a uma dissonância nas aplicações dos diferentes envolvidos.

Tabela 1 – Fatores/efeitos de comunicação em DDS. Fonte (Santos *et al.*, 2012)

FATORES (F)	EFEITOS (E)	REFERÊNCIAS
F1. Diferenças Culturais	E1. Incertezas, Mal-Entendidos e Equívocos	Jiménez *et al.* (2009), Khan *et al.* (2009), Persson *et al.* (2009), Silva *et al.* (2010)
	E2. Compartilhamento das Informações limitado	
	E3. Falta de Confiança	
	E16. Ausência de Comunicação Síncrona	
F2. Dispersão Geográfica e Distância Temporal	E1. Incertezas, Mal-Entendidos e Equívocos	Khan *et al.* (2009), Persson *et al.* (2009) Noll *et al.* (2010)
	E2. Compartilhamento das Informações limitado	
	E3. Falta de Confiança	
	E5. Atraso das Respostas	
	E13. Redução da Frequência da Comunicação	
	E15. Perda de Informações	
F3. Idioma e Barreiras Linguísticas	E1. Incertezas, Mal-Entendidos e Equívocos	Khan *et al.* (2009), Persson *et al.* (2009)
	E2. Compartilhamento das Informações limitado	
	E3. Falta de Confiança	
	E13. Redução da Frequência da Comunicação	
F4. Definição do Meio de Comunicação (síncrono e/ou assíncrono)	E4. Qualidade da Comunicação	Trindade *et al.* (2008), Persson *et al.* (2009), Prikladnicki & Audy (2010)
	E6. Processo de Levantamento de Requisitos	
	E9. Ambiguidade das Informações	
	E10. Gerenciamento de Projetos Distribuídos	
F5. Seleção das Tecnologias de Comunicação	E2. Compartilhamento das Informações limitado	Jiménez *et al.* (2009), Khan *et al.* (2009), Persson *et al.* (2009), Noll *et al.* (2010), Prikladnicki, & Audy (2010)
	E4. Qualidade da Comunicação	
	E7. Compartilhamento do Conhecimento	
	E8. Relações Pessoais	
	E10. Gerenciamento de Projetos Distribuídos	
	E14. Colaboração entre Equipes	
	E15. Perda de Informações	
	E17. Sucesso do Projeto	
	E25. Amadurecimento da Equipe	

Apesar de vários trabalhos terem produzido resultados na direção do entendimento do processo de comunicação em DDS (*e.g.*, Trindade *et al.*, 2008; Farias Junior *et al.*, 2009; Silva *et al.*, 2011), ainda existem lacunas referentes nesse processo. Dessa forma, como um dos novos cenários, destacam-se os ECOSs, discutidos a seguir.

3. O Contexto dos Ecossistemas de Software e o DDS

O desenvolvimento de software vem se voltando mais para a composição de partes existentes, sejam estas construídas internamente pela empresa ou ainda adquiridas no mercado. A partir de uma tecnologia de software central, uma comunidade de atores e de organizações em rede passa a apoiar as relações entre eles sobre um interesse comum no desenvolvimento desses sistemas, originando os chamados ECOSs (Hanssen & Dyba, 2012). Segundo Berk *et al.* (2010), um ECOS pode ser entendido como um tipo de

ecossistema de negócio, uma analogia criada nos anos de 1990 para descrever uma nova forma de observar certas redes de negócios pelas Escolas de Administração. Estes ecossistemas consistem basicamente de elementos como um centralizador (*hub*), uma plataforma (tecnologia/mercado) e os agentes do nicho relacionado (*niche players*). O centralizador é o proprietário da plataforma e os agentes do nicho podem utilizar esta plataforma para gerar valor para ela e para si mesmo.

Por exemplo, analisando o ECOS Windows, a Microsoft é o centralizador, o Windows é a plataforma, e as outras empresas, ou agentes deste nicho, utilizam esta plataforma para construir suas aplicações. Quanto mais empresas seguem este exemplo e abrem seus negócios para outras, mais ECOSs começam a se formar. Um problema chave está no gerenciamento da comunicação e da quantidade de empresas e de suas relações em um ECOS organizado em torno de desenvolvedores (*i.e.*, programadores), parceiros (*i.e.*, fornecedores de ferramentas e de partes do software) e clientes (incluindo os usuários), o que pode dificultar *insights* relativos aos papéis que cada um possui no ECOS, bem como impactar ações e decisões (Werner *et al.*, 2012). Deve-se destacar o fato de que os fornecedores de software não atuam mais como unidades independentes que entregam produtos separados, mas se tornam cada vez mais dependentes de outros fornecedores devido ao uso de componentes e infraestruturas externas que são vitais para sua plataforma, como sistemas operacionais e bibliotecas (Berk *et al.,* 2010). Desta forma, esses fornecedores recorrem à integração virtual por meio de alianças que estabelecem redes de influência e interoperabilidade, gerando os ECOSs. Jansen *et al.* (2009) modelaram os ECOS em três níveis:

- O primeiro nível, chamado **organizacional** (*Independent Software Vendor*, ou ISV), tem como objetos de estudo os atores e seus relacionamentos no contexto da empresa inserida em algum ECOS, de forma a analisar desempenho e evolução como dois fatores dependentes dos empreendedores do ECOS. Neste nível, a abertura da empresa é a questão alvo, considerando aspectos de compartilhamento de conhecimento (*i.e.*, gerenciamento do produto, vendas, marketing e suporte) e de pesquisa, mercado e tecnologia com seus parceiros;
- No segundo nível, das **redes de produção de software** (*Software Supply Network*, ou SSN), os objetos de estudo são estas redes dentro de um ECOS, bem como os relacionamentos entre elas. O alvo é o conjunto dos participantes (*i.e.*, fornecedores, clientes, distribuidores e desenvolvedores externos ou terceiros) e as características internas relacionadas à saúde e à estabilidade do ECOS (*i.e.*, tamanho, tipo, papéis, conectividade etc.);
- O terceiro nível, dos **ECOSs**, possui como objeto de estudo os ECOSs *per si*, incluindo os relacionamentos entre eles. É importante entender que os ECOSs devem ter suas fronteiras bem definidas, *e.g.*, um mercado, uma tecnologia, uma empresa, mesmo que elas sejam sobrepostas em sua análise (*i.e.*, um ator pode estar em/representar diferentes ECOSs). Deve-se considerar ainda restrições temporais e geográficas (DDS), especificações de componentes da plataforma, disponibilidade de licenças para uso, e sua idade e história.

Por fim, alguns desafios e fatores de sucesso foram apontados por Bosch (2009) e Jansen *et al.* (2009): (i) caracterizar e modelar ECOSs, considerando diferentes perspectivas, *e.g.*, no Brasil, a criação e evolução dos ECOSs pode depender

diretamente da atuação do Estado e de suas políticas; (ii) estabelecer os relacionamentos entre as redes de produção de software; (iii) gerenciar a qualidade em ECOSs; (iv) planejar portfólios e linhas de produtos em ECOSs, incorporando elementos econômicos e sociais; (v) prover métodos, técnicas e ferramentas para desenvolver arquiteturas de sistemas que atendam à extensibilidade, à portabilidade e à variabilidade; e (vi) tratar implicações gerais em ES, incluindo mecanismos de comunicação, agilidade na concepção e desenvolvimento de soluções, e composição de produtos e serviços.

4. A Comunicação em DDS no Contexto dos Ecossistemas de Software

Conforme discutido na Seção 3, um caminho para a evolução da ES globalizada e orientada pelas questões econômicas e sociais está na aplicação do conceito de ECOSs, cuja base envolve a teoria de redes sociais e o DDS. É importante considerar que a inovação faz parte do contexto de ECOSs e que pode fortalecer as SSNs a nível local, nacional e global por meio de ações das pessoas e das tecnologias. Além disso, a colaboração dentro das SSNs pode ser ainda mais estimulada pela adoção de software social (*e.g.*, Wikis, blogs etc.), que permite aos atores interagirem e compartilharem conhecimento em uma dimensão que extrapola a social, realçando o potencial humano de criação ao invés de privilegiar somente a transmissão um para muitos.

O DDS é beneficiado pela ascensão do tema ECOSs na área de ES. No DDS, encontram-se vários desafios que impactam essa nova forma de elicitar, desenvolver e testar software. Contudo, a comunicação que está contida na dimensão social do ECOS é um dos desafios mais significativos, dado que ela existe em qualquer organização ou projeto, distribuído ou co-localizado. Em projetos distribuídos, esses desafios são potencializados, gerando vários entraves para um bom gerenciamento e engenharia (Santos & Werner, 2012). Partindo dos desafios de ECOSs apresentados na Seção 3, novos fatores que impactam a comunicação em DDS podem ser acrescentados. Esses fatores foram extraídos de Santos *et al.* (2012) e complementam os fatores apresentados na Tabela 1. Com o intuito de facilitar a compreensão, tais fatores foram numerados em sequência àqueles discutidos na Seção 2.

- **F6. Coordenação**: a coordenação de projetos e equipes em ECOSs torna-se mais difícil, em função de problemas de comunicação, influenciando principalmente o gerenciamento de projetos, conhecidos ou não, sobre a mesma tecnologia de software central. Incertezas e dependências entre os atores, típico em DDS, aumentam a necessidade de coordenação, podendo reduzir o desempenho e propiciar maior número de falhas;
- **F7. Visibilidade e Percepção**: manter a visibilidade dos trabalhos das equipes é uma tarefa difícil em um projeto distribuído, porém é extremamente importante a visibilidade das competências dos parceiros e habilidades, facilitando a atribuição de tarefas e responsabilidades nos ECOSs, dado que a abertura da plataforma sugere explorar a inovação por parte de desenvolvedores externos ou terceiros. A percepção é dependente da consciência da evolução do trabalho, do que está sendo trabalhado, bem como a evolução do projeto e da plataforma;
- **F8. Múltiplos Canais de Comunicação**: disponibilizar diferentes meios de comunicação é considerado como uma boa prática, visto que, alguns membros de equipe preferem comunicação de voz síncrona, enquanto outros, com menor

conhecimento do idioma, preferem comunicação assíncrona. Em ECOSs, mídias colaborativas, mais especificamente sites de redes sociais e *wikis*, podem contribuir para maximizar a comunicação (Lima *et al.*, 2013);

- **F9. Ferramentas de Colaboração**: são definidas como ferramentas que permitem a gestão de atividades e tarefas em equipes distribuídas, *e.g.*, editores de texto na nuvem, sistema de controle de solicitação etc. Estas devem ser simples e de fácil utilização e interação, a fim de melhorar a integração do conhecimento, sobretudo mais próximas à realidade dos diversos atores envolvidos no (e que sejam de interesse do) ECOS. O uso dessas ferramentas pode evitar ambiguidades, contribuindo para o monitoramento e controle das atividades, sem comprometer a qualidade dos resultados. Integração e interoperabilidade entre essas ferramentas também é importante para os ECOSs;
- **F10. Distribuição de Tarefas**: assim como atribuir tarefas, resolver as dependências entre tarefas é um grande desafio em ambientes distribuídos (Amrit, 2005). A falta de visibilidade das competências dos parceiros e habilidades pode ocasionar um aumento significativo na dificuldade em distribuir as tarefas, resultando na redução do nível de desempenho, bem como o aumento do número de falhas. Em ECOSs, isso se torna ainda mais dinâmico, pois diversas tarefas similares podem acontecer concorrentemente dentro da mesma plataforma, devido ao *time-to-market* e à inovação;
- **F11. Políticas de Comunicação**: adoção de políticas de comunicação, *e.g.*, rotações planejadas entre membros, restrições de tempo de respostas e postagem de resultados em tempo hábil, contribuem para o compartilhamento do conhecimento, evitam o atraso de respostas e melhoram reuniões distribuídas. Em ECOSs, isso depende dos modelos de associação que uma plataforma adota (Angeren *et al.*, 2011), *e.g.*, modelo de parceria (organização externa atua exclusivamente sobre a plataforma, *e.g.*, ECOS SAP) ou modelo de membro (organização externa atua em diversas plataformas, *e.g.*, ECOS Eclipse);
- **F12. Redes de Contato**: em ambientes distribuídos como no contexto dos ECOSs, pode ser complicado cultivar uma rede de contatos devido a diversos fatores, *e.g.*, dispersão geográfica e barreiras linguísticas, que influenciam a frequência e qualidade da comunicação. Ferramentas que identifiquem e estimulem a necessidade de comunicação entre atores do ECOS é fundamental. A integração dessas ferramentas àquelas para apoiar colaboração e comunicação facilita o acesso e identificação dos envolvidos;
- **F13. Infraestrutura**: infraestrutura não adequada ou mal organizada tem um impacto negativo no processo de comunicação em projetos distribuídos com ECOSs. Com as equipes e organizações se comunicando constantemente, toda a infraestrutura física e tecnológica deve ser muito bem organizada e gerida pelo centralizador da plataforma. Além disso, a infraestrutura precisa ser segura para garantir que a propriedade intelectual e outros conhecimentos não sejam acessados por pessoas não autorizadas, e garantir uma boa comunicação.

A essência do conceito de ECOSs encoraja um conjunto de desenvolvedores externos a utilizarem uma plataforma pertencente a um grupo de organizações e a

contribuírem para o desenvolvimento do produto (Hanssen & Dyba, 2012). Estabelece-se assim uma comunidade que acelera o compartilhamento de conteúdo, conhecimento, problemas, experiência e habilidades; que é dependente da comunicação entre os envolvidos; e que se orienta pelo fato de que os ECOSs emergem como uma abordagem para melhorar a reutilização intra e interorganizacional, no cenário da ES globalizada, conforme indicado por Santos & Werner (2012). Motivado por este contexto, os fatores e efeitos identificados no processo de comunicação no DDS por Santos *et al*. (2012) foram associados aos desafios apontados por Bosch (2009) e Jansen *et al.* (2009) e suas respectivas descrições (Tabela 2) para uma melhor compreensão da comunicação e da sua relevância em ECOSs. Alguns destes desafios foram discutidos na Seção 3.

Tabela 2 – Desafios para ECOSs frente à questão da comunicação em DDS

	DESAFIOS DE ECOSs	DESCRIÇÃO	FATORES DA COMUNICAÇÃO	EFEITOS SOBRE A COMUNICAÇÃO
***Organizacional* (ISV)**	D1: *Portfólio e planejamento da linha de produção*	Gerenciar configurações e reutilização para funcionalidades da plataforma.	F4, F5	E1, E2, E3, E14
	D2: *Gestão do conhecimento*	Aplicar mineração para extrair e visualizar *feedback* dos envolvidos.	F1, F2, F3, F5	E1, E3, E4, E5, E6, E7, E8, E9
	D3. Arquitetura estensível, portável e com variabilidade	Explorar interfaces, requisitos e comunicação sobre plataformas.	F5, F6, F7, F8, F13	E1, E2,E3, E10, E11, E12, E13
	D4: *Integração de sistema na organização*	Desenvolver *frameworks* para gerência do produto de software e seus ciclos.	F9	E1, E10, E11
***Rede de Produção* (SSN)**	D5: *Estabelecimento das relações*	Identificar papéis e relacionamentos.	F10, F12	E1, E4, E15
	D6: *Tempo de liberações*	Gerenciar versões e seus impactos.	F4	E1, E2, E14
	D7: *Gestão de qualidade*	Gerar diretrizes para certificação.	F4	E1, E2, E14
***Ecossistema* (ECOS)**	D8: *Caractetização e modelagem de ECOS*	Identificar tamanho, vizinhança, padrões e papéis no ECOS.	F10, F4	E1, E2, E4, E14, E15
	D9: *Políticas e estratégias entre ECOSs*	Determinar instrumentos para permitir orquestração em cada ECOS.	F9, F11	E1, E10, E11, E16
	D10: *Determinação da estratégia para vencer e lucrar na rede*	Integrar modelos de negócio à ES.	F4, F6. F12	E1, E2, E3, E10, E13, E14

É visível a relevância dos fatores e efeitos do processo de comunicação em ECOS. Para lidar com os desafios organizacionais, da rede produção e do ecossistema é fundamental a compreensão das diferenças culturais, dispersão geográfica e distância temporal; necessidade de uma eficiente coordenação e visibilidade e percepção; e definição dos meios de comunicação e seleção das tecnologias de comunicação a fim de minimizar comportamentos conflitantes e mal entendidos para aumentar o desempenho dos envolvidos e reduzir o número de falhas. Além disso, a definição do meio de comunicação é essencial para a definição de papéis, distribuição de tarefas e estímulo das redes de contatos para minimizar as dificuldades no compartilhamento de informações.

5. Conclusão

No cenário da indústria de software globalizada, a comunicação em DDS é impactada por diversos fatores e consequentemente produz efeitos positivos ou negativos sobre o processo de ES. Uma vez que esta indústria não se limita ao desenvolvimento de projetos ou produtos únicos, mas se organiza em ECOSs, a comunicação também sofre interferências deste novo contexto. Este artigo teve por objetivo analisar os fatores e efeitos que influenciam a comunicação no DDS e discutir novos fatores no contexto dos ECOSs e de seus desafios. Para isso, gerou-se um mapeamento entre os desafios dos diferentes níveis de escopo dos ECOSs e os elementos da comunicação em DDS, contribuindo para uma reflexão de pesquisa que integra estes temas em torno de

trabalhos que estão sendo desenvolvidos pelos grupos de pesquisa envolvidos. Como trabalhos futuros, pretende-se analisar as interferências do contexto de ECOSs em um modelo de maturidade para comunicação em DDS, bem como derivar indicadores de saúde para ECOSs. Assim, poderá ser elaborado um método para análise dos impactos relevantes de comunicação em DDS a partir dos níveis de escopo dos ECOSs.

Referências

Angeren, J., Kabbedjik, J., Jansen, S., Popp, K.M. (2011) "A Survey of Associate Models used within Large Software Ecosystems", In: 3rd IWSECO, Brussels, Belgium, pp. 27-39.

Amrit, C. (2005) "Coordination in Software Development: The Problem of Task Allocation", In: Workshop on Human and Social Factors of Soft. Eng., ACM, St. Louis, USA, pp. 1-7.

Berk, I., Jansen, S., Luinenburg, L. (2010) "Software Ecosystems: A Software Ecosystem Strategy Assessment Model", In: 2nd IWSECO, Copenhagen, Denmark, pp. 135-142.

Boehm, B. (2006) "A View of 20th and 21st Century Software Engineering", In: 28th ICSE, Shanghai, China, pp. 12-29.

Bosch, J. (2009) "From Software Product Lines to Software Ecosystem", In: 13th SPLC, San Francisco, USA, pp. 1-10.

Carmel, E. (1999) "Global Software Teams – Collaborating Across Borders and Time-Zones".

Farias Junior, I. *et al.* (2009) "Proposta de Boas Práticas no Processo de Comunicação em Projetos Distribuídos", In: III WDDS, Fortaleza, Brasil, pp. 80-88.

Hanssen, G.K., Dyba, T. (2012) "Theoretical Foundations of Software Ecosystems", In: 4th IWSECO, Boston, USA, pp. 6-17.

Jansen, S., Finkelstein, A., Brinkkemper, S. (2009) "A Sense of Community: A Research Agenda for Software Ecosystems". In: 31st ICSE, Vancouver, Canada, pp. 187-190.

Jiménez, M., Piattini, M., Vizcaíno, A. (2009) "Challenges and Improvements in Distributed Software Development: A Systematic Review", Advances in Software Engineering, v. 2009, n. 3, pp. 1-16.

Khan, S. *et al.* (2009) "Critical Success Factors for Offshore Software Development Outsourcing Vendors: A Syst. Lit. Review", In: 4th ICGSE, Washington, USA, pp. 207-216.

Lima, T., Santos, R., Werner, C. (2013) "Apoio à Compreensão das Redes Socio-técnicas em Ecossistemas de Software". In: II BraSNAM, XXXIII CSBC, Maceió, Brasil.

Noll, J., Beecham, S., Richardson, I. (2010) "Global Software Development and Collaboration: Barriers and Solutions", ACM Inroads, v. 1, n. 3 (Sep), pp. 66-78.

Persson, J. *et al.* (2009) "Managing Risks in Distributed Software Projects: An Integrative Framework", IEEE Transactions on Engineering Management, v. 56, n. 3, pp. 508-532.

Prikladnicki, R., Audy, J. (2010) "Process models in the practice of distributed software development: A Systematic review of the literature", IST Journal, v. 52, n. 8, pp. 779-791.

Santos, A., Farias Junior, I., Moura, H., Marczak, S. (2012) "A Systematic Tertiary Study of Communication in DSD Projects", In: 7th ICGSE, Porto Alegre, Brazil, p. 182.

Santos, R., Werner, C. (2012) "ReuseECOS: An Approach to Support Global Software Development through Software Ecosystems". In: VI WDDS, Porto Alegre, Brazil, pp. 60-65.

Silva, F. *et al.* (2010) "Challenges and Solutions in Distributed Software Development Project Management: a Systematic Lit. Review", In: 5th ICGSE, Washington, USA, pp. 87-96.

Silva, F., Prikladnicki, R., Franca, P., Monteiro, C., Costa, C., Rocha, R. (2011) "An evidence-based model of distributed software development project management: results from a systematic mapping study", Journal of Software Maintenance and Evolution, v. 24, n. 6, pp. 625–642.

Trindade, C. *et al.* (2008) "Comunicação em Equipes Distribuídas de Desenvolvimento de Software: Uma Revisão Sistemática", In: 5th ESELAW, Salvador, Brasil, pp. 1-10.

Werner, C., Santos, R., Alves, C. (2012) "Ecossistemas de Software: Estágio Atual, Direções de Pesquisa e a Prática na Indústria de Software", In: III CBSoft, Natal, Brasil, pp. 12-13.

Elementos que Impactam o Planejamento de Testes em Ambientes de Desenvolvimento Distribuído no Contexto de Ecossistemas de Software

Nayane Maia[1], Rodrigo Santos[2], Elisa Huzita[3], Arilo Dias-Neto[1], Claudia Werner[2]

[1]Instituto de Computação (IComp), Universidade Federal do Amazonas – Manaus – AM

[2]PESC/COPPE, Universidade Federal do Rio de Janeiro – Rio de Janeiro – RJ

[3]Departamento de Informática, Universidade Estadual de Maringá – Maringá – PR

{nayane.maia,arilo}@icomp.ufam.edu.br, {rps,werner}@cos.ufrj.br, emhuzita@din.uem.br

Resumo. *A organização de Ecossistemas de Software (ECOSs) na indústria vem interferindo nos projetos com Desenvolvimento Distribuído de Software (DDS), uma vez que demanda novos modelos abertos de negócio, que geram novos papéis e padrões para colaboração, inovação e proposição de valor. Este cenário requer atenção aos testes, tidos como a segunda maior atividade terceirizada, dado que seu planejamento e controle visam assegurar o sucesso de projetos de DDS. Este artigo tem por objetivo identificar e classificar elementos que impactam o planejamento de testes em ambientes de DDS a partir de uma agenda de pesquisa, assim como discutir seus desdobramentos no contexto dos ECOSs, contribuindo para pesquisas envolvendo esses temas.*

1. Introdução

O Desenvolvimento Distribuído de Software (DDS) tem sido adotado pelas empresas em busca de redução de custos por meio do melhor aproveitamento de recursos humanos, distribuídos ao redor do mundo. Este cenário impacta diretamente a pesquisa e prática da Engenharia de Software (ES), considerando que os desafios do DDS se relacionam com o tratamento das questões econômicas e sociais da ES (Boehm, 2006). No entanto, o DDS não vem se limitando ao desenvolvimento de projetos ou produtos de software únicos, mas incluindo o contexto de arquiteturas e/ou plataformas comuns e/ou relacionadas, com a participação de desenvolvedores externos e usuários, conhecidos ou não (Santos & Werner, 2012).

Conhecidas como Ecossistemas de Software (ECOSs), essas redes de atores e de artefatos formadas sobre uma tecnologia de software central passam a interferir nos projetos de DDS, pois requerem novos modelos abertos de negócio que geram papéis e padrões para colaboração, inovação e proposição de valor. Especificamente entre os processos de ES, um estudo revela que o teste é a segunda maior atividade terceirizada, após a codificação (Shah *et al.*, 2011), de modo que planejamento e controle são necessários para assegurar o sucesso de projetos de DDS (Collins *et al.*, 2012). Assim, torna-se importante compreender que elementos impactam o planejamento de testes nas combinações de cenários de DDS em geral e que são afetados pelo contexto dos ECOSs.

Este artigo tem por objetivo identificar e classificar elementos que impactam o planejamento de testes em ambientes de DDS a partir de uma agenda de pesquisa,

proposta por Huzita *et al.* (2012), assim como discutir seus desdobramentos no contexto dos ECOSs (Santos & Werner, 2012). O artigo está dividido da seguinte forma: a Seção 2 discute os ECOSs e o DDS; a Seção 3 apresenta uma classificação dos elementos que impactam o planejamento de testes em ambientes de DDS; a Seção 4 discorre sobre os desdobramentos dos ECOSs; e a Seção 5 conclui o artigo, enunciando trabalhos futuros.

2. Ecossistemas de Software e o Cenário de DDS

O DDS tem sido cada vez mais adotado pelas empresas, de forma que desenvolver software se torna uma tarefa quase ininterrupta. No entanto, devido às diferenças culturais, políticas e organizacionais, bem como às distâncias temporal e espacial, emergem também desafios relacionados a cooperação, coordenação e comunicação. A fim de reduzir estas diferenças e distâncias e assegurar que indivíduos geograficamente distribuídos estão colaborando nos projetos de software, é essencial uma infraestrutura que garanta a troca de informação e de conhecimento entre os envolvidos (Chaves *et al.*, 2010). Nesse sentido, Huzita *et al*. (2012) discutem uma agenda de pesquisa envolvendo gerenciamento de informação contextual e de conhecimento em DDS (Tabela 1).

Tabela 1 – Agenda de pesquisa para DDS. Fonte: (Huzita *et al.*, 2012)

ID	Desafio	Descrição
A1	*Explorar gestão de conhecimento em projetos de DDS*	Identificar, extrair, ordenar, evoluir e analisar a importância e impacto de conhecimentos relacionados a domínio, processo de desenvolvimento, recursos técnicos, recursos não técnicos e tecnologias.
A2	*Definir mecanismos para apoiar a captura de informação contextual que possa vir de fontes heterogêneas*	Como o cenário considerado é o de DDS, os dados poderão ser provenientes de vários locais. Além disso, é importante também estabelecer a frequência e formato que estes dados serão capturados.
A3	*Explorar computação em nuvem sob os aspectos de: software como serviço, plataforma como serviço e infraestrutura como serviço*	Pelo volume de dados gerados, é importante que sejam usadas tecnologias que garantam confiabilidade e disponibilidade de informação. O foco seria explorar a definição de estratégias adequadas para armazenar, nas nuvens, informações contextuais provenientes de diferentes locais.
A4	*Explorar meios adequados para a representação, processamento e disseminação de conhecimento e informação contextual*	Uma ideia é fazer uso de ontologia (Chaves *et al*., 2010). Possíveis benefícios: definição de um vocabulário de consenso; possibilidade de desenvolver software mais complexo ao associar técnicas de inferência; e reutilização do conhecimento modelado em novas aplicações.
A5	*Explorar meios para apresentação, visualização e disseminação de informação contextual de acordo com o objetivo, perfil e formato*	A ideia é estudar técnicas adequadas que permitam a visualização, disseminação e apresentação de informações de acordo com parâmetros e características que sejam de interesse no cenário de DDS.
A6	*Mapear os aspectos socioculturais capturados em projetos DDS para cada uma das fases do modelo de conversão de conhecimento*	Estudar como as informações socioculturais capturadas durante o desenvolvimento de software podem ser mapeadas ou alinhadas com as diferentes fases do modelo de conversão de conhecimento: socialização, externalização, combinação e internalização.
A7	*Definir indicadores e o subsequente gerenciamento de desempenho em DDS*	Definir as dimensões de desempenho para o processo de tomada de decisão e, também, para a contínua melhoria de processo e produto em DDS, a fim de projetar mecanismos para apoiar o desenvolvimento e a reutilização de conhecimento.
A8	*Prover o tomador de decisão com ferramentas para gestão de informação de contexto e de conhecimento*	Desenvolver ferramentas que ofereçam o apoio necessário para capturar, recuperar, reutilizar e integrar o conhecimento e as informações de contexto geradas em DDS.

Paralelamente, o desenvolvimento de software requer pensar cuidadosamente as plataformas que vão apoiá-lo e as redes de artefatos e envolvidos, i.e., relacionamentos de conectividade e dependência entre produtos e organizações, que afetam (e são afetados por) esse cenário (Jansen *et al.*, 2009). Isso corrobora com o fato de que as inovações não mais se originam de uma organização, mas da chamada "co-inovação" a partir de diferentes agentes da indústria de software, além dos processos denominados de "co-evolução", possíveis somente a partir da reunião de organizações colaborativamente focadas em dar suporte a novos produtos, a satisfazer as necessidades dos clientes e a incorporar novas rodadas de inovação (Arndt & Dibbern, 2006).

Neste caso, o sistema de software representa uma combinação de software,

hardware e "peopleware", constituída sobre um ambiente comum, i.e., uma plataforma. Pode-se identificar, então, uma tendência na ES: o nascimento, desenvolvimento, amadurecimento e eventual "morte" ou transformação de plataformas levam à "desconstrução" da visão tradicional da ES. Privilegia-se assim uma nova visão, que contempla a colaboração e a interoperabilidade entre os atores (Campbell & Ahmed, 2010). Na década de 2000, essas estruturas foram alvo de pesquisa na ES, chamadas de Ecossistemas de Software ou ECOSs (Messerschmitt & Szyperski, 2003). Segundo Hanssen & Dyba (2012), um ECOS consiste em uma comunidade de atores e de organizações em rede, que apoia as relações entre eles sobre um interesse comum no desenvolvimento e no uso de uma tecnologia de software central.

Inspirado nos ecossistemas biológicos, sociais e de negócios, o tema ECOSs favorece o uso de analogias para aplicar novas visões e abstrações sobre os desafios da indústria de software (Jansen *et al.*, 2013). Por exemplo, a influência da visão de negócio pode ser notada no fato de que as equipes de desenvolvimento têm trabalhado geograficamente dispersas. Isso permite a expansão da indústria para novos mercados, embora necessite de um olhar para questões socioculturais (Pinto *et al.*, 2009). Como um dos desafios, Santos *et al.* (2012) apontam a decisão sobre qual é a estratégia mais inovadora no desenvolvimento de software globalizado. É importante mapear os pontos fortes e fracos das plataformas de ECOSs para definir e automatizar um *framework* para o seu gerenciamento e monitoração, local e globalmente.

Emergem três tendências que aceleram a complexidade do novo contexto da ES (Bosch & Bosch-Sijtsema, 2010): (1) ampla adoção de *Linhas de Produtos de Software* (LPSs): como expor os ativos reutilizáveis e abrir a arquitetura da plataforma; (2) *Globalização do Desenvolvimento de Software* (GDSs): como gerenciar a dependência socio-técnica em suas redes de produção; e (3) *organização de ECOSs* em comunidades de desenvolvedores externos sobre produto de sucesso: como entender as relações entre as partes sob a perspectiva da reutilização mais ampla. Nesse sentido, Santos & Werner (2012) propõem um *framework* para classificar a iniciativas de pesquisa em ECOSs em quatro dimensões de análise, chamado *ReuseECOS*, três destas inicialmente exploradas por Campbell & Ahmed (2010). As duas primeiras (transacional e técnica) são normalmente alvo de desejo de controle pelo centralizador, ao passo que a terceira (social) é dinâmica e a "pedra angular" para o sucesso do ECOS, ao impactar as fases do seu ciclo de vida, estruturado pela quarta (engenharia e gerenciamento):

- *técnica*: dimensão voltada ao entendimento da abertura das plataformas e do envolvimento de atores externos. É dirigida por três fatores: (1) engenharia da plataforma (sistema de software); (2) arquitetura (estrutura dos componentes, seus relacionamentos e princípios e diretrizes que norteiam a sua evolução); (3) gerenciamento de requisitos (funcionalidades comuns ou específicas);
- *transacional*: dimensão voltada ao entendimento da colaboração e/ou competitividade, e nas expectativas dos envolvidos e o impacto na produtividade no ambiente destes. É dirigida por três fatores: (1) visão do ambiente; (2) inovação; e (3) planejamento estratégico;
- *social*: dimensão voltada ao entendimento da abertura da empresa como um todo, nas comunidades originadas ao redor das plataformas e no desenvolvimento colaborativo. É dirigida por três fatores: (1) utilidade (compensações e recompensas, esperadas e percebidas pelos membros); (2) promoção

(reconhecimento implícito/explícito de capacidades/habilidades do colaborador); e (3) ganho de conhecimento (armazenamento e manutenção de experiências, e oportunidades para as partes interessadas se envolverem com novas tecnologias);

- *engenharia e gerenciamento*: dimensão focada em intercambiar e integrar as três dimensões apresentadas a fim de viabilizar uma infraestrutura de apoio. Utiliza as relações que correspondem a: (1) motivar o desenvolvimento e evolução da plataforma (**técnica-transacional**); (2) contribuir para o estabelecimento da plataforma (**técnica-social**); e (3) mapear as proposições e percepções de valor (**transacional-social**).

3. Práticas em Ambientes de DDS Aplicáveis no Planejamento de Testes

Segundo Juristo *et al.* (2004), o teste de software é considerado uma das práticas mais custosa do processo de desenvolvimento e, por isso, requer recursos adequados, cuja utilização efetiva necessita de um bom planejamento e controle (McGregor & Sykes, 2001). O planejamento garante que o critério de teste seja especificado e o conjunto de casos e procedimentos de testes seja desenvolvido antes do fim da fase de implementação do produto, evitando testes tendenciosos. Por sua vez, a atividade de controle assegura que os testes planejados sejam monitorados constantemente e que seus resultados sejam registrados. Alguns trabalhos discutidos a seguir tratam da atividade de teste de software no processo de DDS, destacando o planejamento e controle como fatores de impacto no sucesso dos projetos.

No estudo de Collins *et al.* (2012), foram apresentados os desafios e as lições aprendidas no contexto de um projeto de desenvolvimento ágil utilizando testes distribuídos. Constatou-se neste estudo que algumas práticas de planejamento e controle asseguram o sucesso do projeto, tais como: (i) comunicação e coordenação são fatores essenciais para o sucesso de testes distribuídos; (ii) as informações do projeto devem estar disponíveis com detalhes a todos os envolvidos; (iii) a automação reduz as necessidades de presença física no processo de teste; e (iv) as ferramentas de apoio são sempre importantes, mas a organização da equipe de teste é mais significativa. Por sua vez, em Causevic *et al.* (2010), constatou-se, entre outras coisas, que a ausência de infraestrutura é considerada um fator importante para retardar a implantação de software em projetos de desenvolvimento distribuído.

Maia *et al.* (2012) apresentam um relato de experiência no contexto de um projeto que executou testes ágeis com equipes distribuídas, composto por uma equipe de teste local e outra remota, mostrando que a atividade de teste requer colaboração e cooperação entre os envolvidos. No entanto, esta colaboração e cooperação se tornam mais difíceis quando os membros da equipe de teste estão separados geograficamente da equipe de desenvolvimento. Para contornar a distância de parte da equipe de testes, foi necessário fornecer uma infraestrutura de testes, que permitisse o acesso às informações do projeto por todos os envolvidos de forma eficiente, constituída por: ferramenta de gestão de tarefas Scrum (*FireScrum*), ferramenta de gestão de teste (*TestLink*), ferramenta de gestão de defeitos (*Mantis*), ferramenta de versionamento (*Subversion*), servidor de aplicação para testes e ferramentas de comunicação (*e-mail* e *chats*).

A fim de compreender os temas afins que impactam os testes em DDS, a Tabela 2 reúne os desafios encontrados durante esta fase, a partir dos estudos de Jiménez *et al.* (2009) e Rocha *et al.* (2010), e mostra o seu cruzamento com as práticas a serem

aplicadas em DDS, extraídas de (Huzita *et al.*, 2008): (P1) oferecer mecanismos para facilitar a comunicação; (P2) desenvolver um modelo de produtos; (P3) gerenciar processo/projeto; (P4) praticar a co-alocação temporária; (P5) estabelecer critérios para constituição de equipes e estabelecer senso de equipe; (P6) disponibilizar e compartilhar informações de projeto; (P7) lidar com heterogeneidade; (P8) distribuir responsabilidades; (P9) apoiar a colaboração por meio de *awareness* e *group awareness*; (P10) distribuir atividades; (P11) definir métricas; e (P12) estabelecer senso de confiança. Essas práticas resultaram de uma análise considerando os desafios em DDS, tais como aqueles relativos às distâncias observadas (geográfica, cultural e temporal) e, também, aqueles relacionados a coordenação, colaboração e controle.

Tabela 2 – Desafios introduzidos pelo DDS na atividade de testes

Temas Afins	Desafios dos Testes em DDS	Práticas
Comunicação	Demora para resposta via *e-mails*. Idioma utilizado. Ausência de contato visual.	P1, P4, P9, P12
Stakeholders	Definição de responsabilidades. Difícil determinação da fronteira entre as atividades dos membros.	P5, P8, P9, P10
Plano de Testes	Definir planos de testes e executá-los, de forma que todos os envolvidos participem efetivamente dos testes.	P2, P3, P6
Ambiente Técnico	Prover uma infraestrutura computacional adequada para apoiar os testes e ter um nível de conhecimento técnico em todos os sítios (locais) envolvidos.	P1, P7, P11
Complexidade	Decidir como dividir o trabalho em diversas equipes ou centros distribuídos geograficamente, considerando os diferentes níveis técnicos das equipes.	P5, P6, P8, P9, P10,
Equipes	Dependência de equipes externas.	P1, P4, P9, P12
Gestão de Conhecimento	Compartilhar informação entre os membros do projeto e manter a documentação constantemente atualizada.	P3, P6, P8, P9
Gestão do Projeto/Processo	Processos diferentes aplicados nos diferentes sítios.	P2, P3, P5, P12
Questões Culturais	Unificar as culturas existentes nos diferentes sítios onde os testes são aplicados.	P1, P9, P12

A título de ilustração, ao considerar os temas *stakeholders*, *equipes* e *questões culturais* presentes na Tabela 2, verifica-se que uma forma de mitigá-los seria aplicar as práticas P1, P4, P5, P8, P9, P10, P12, o que consiste na união das práticas associadas a estes três temas. Pode ser observado ainda que, na junção destes temas, o interesse é que se consiga uma equipe de teste com os membros distribuídos geograficamente, que possa gerar informações contextuais que devam ser adequadamente capturadas e usadas. Isto pode motivar pesquisas que vão de encontro ao item A2, presente na Tabela 1.

A partir da análise dos trabalhos relacionados, é possível perceber que um dos desafios em DDS é a comunicação. Para se estabelecer a comunicação, é necessária uma infraestrutura adequada, quer seja de redes como também de ferramentas de apoio. No entanto, apenas a disponibilidade de infraestrutura não seria suficiente pois as pessoas envolvidas nas equipes necessitam deter um conhecimento técnico específico, particularmente em testes, em nível adequado e suficiente para que se estabeleça a colaboração. Neste contexto, Shah *et al.* (2011) descrevem um estudo em que o teste distribuído de software é estruturado em torno de três fases: preparação dos testes, execução dos testes e gestão de clientes, surgindo a necessidade de se criar mais um papel (gestor de cliente), que normalmente não existe no processo tradicional de desenvolvimento. Este, por sua vez, irá funcionar como uma "ponte" entre a equipe de testes e o cliente. Por se tratar de DDS, estas pessoas podem estar em locais distintos e atribuir tarefas a pessoas com quem muitas vezes não se teve ainda a oportunidade para um encontro face a face – bem diferente de atribuir tarefas quando se conhece a pessoa.

O estudo de Jain *et al.* (2011) revela que em DDS o time de teste passa a ter um papel mais estratégico (planejar e controlar) do que meramente técnico (executar e reportar defeitos). Portanto alocar pessoas para testes e acompanhar o trabalho delas

requer um cuidado maior. Com isso, pode-se dizer que que todas as fases do desenvolvimento sofrem algum impacto quando o contexto é DDS. Contudo, a fase de testes possui suas especificidades, pois depende diretamente dos resultados providos em outras fases do processo de desenvolvimento para finalizar a atividade de validação.

4. Desdobramentos do Contexto de Ecossistemas de Software

Partindo dos desafios introduzidos pelo DDS nas atividades de teste, os temas afins identificados podem ser inicialmente classificados nas quatro dimensões do *framework ReuseECOS*, conforme ilustra a Figura 1. Na dimensão técnica, o tema *ambiente técnico* é considerado por tratar da concepção de uma infraestrutura de apoio aos testes em DDS, que se relaciona diretamente com engenharia da plataforma e com o suporte ao envolvimento de atores externos frente a certificação e testes de suas contribuições na arquitetura da plataforma. Na dimensão transacional, ressalta-se o tema *comunicação*, por representar um elemento chave para viabilizar a colaboração entre equipes de teste distribuídas, tanto dos componentes, serviços e aplicações desenvolvidos por terceiros como na sua integração e/ou execução sobre a plataforma. Na dimensão social, o tema *questões culturais* demanda atenção especial, sobretudo pelo tratamento dos diversos desenvolvedores e usuários da plataforma, conhecidos ou não, a fim de mantê-los ativos.

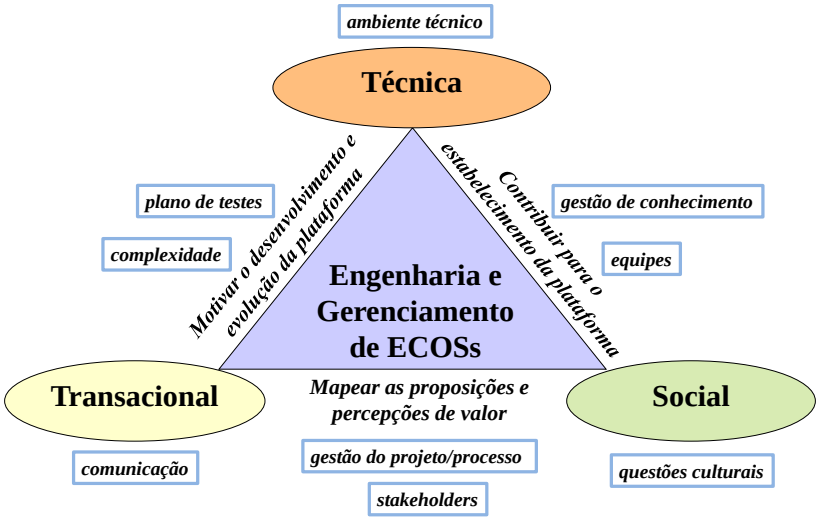


Figura 1 – Temas afins dos desafios introduzidos pelo DDS na atividade de teste e sua classificação nas dimensões de ECOSs

A dimensão engenharia e gerenciamento permite a compreensão dos elementos que existem nas relações entre as outras dimensões. Na relação técnica-transacional, dois temas são discutidos: *plano de testes* e *complexidade*. Ambos os temas aparecem nesta relação por dependerem diretamente do modelo de negócio de DDS adotado pelo centralizador da plataforma de um ECOS, i.e., empresa que lidera o ECOS (e.g., Apple, Microsoft, Nokia). Por exemplo, segundo Angeren *et al.* (2011), caso o modelo seja de parceria, os desenvolvedores externos atuam somente na plataforma do ECOS, o que permite gerir de maneira mais padronizada o planejamento e execução dos testes. Por outro lado, caso o modelo seja de membro, os desenvolvedores podem atuar em diversas plataformas, o que pode impactar as atividades de teste pelas experiências vividas pelas equipes na atuação em diferentes ECOSs. A complexidade pode ser gerida por meio de políticas e *roadmaps* em estratégias de governança (Albert *et al.*, 2012), podendo produzir desdobramentos sobre a comunicação e, consequentemente, sobre a interação

humana (complexa por natureza), atingindo, assim, a dimensão social.

Na relação técnica-social, os temas *gestão de conhecimento* e *equipes* aparecem devido ao fato de que recursos, informações e artefatos disponíveis e/ou compartilhados na plataforma de um ECOS devem ser gerenciados e mantidos, não somente pelo centralizador da plataforma como também pela comunidade subjacente. Novamente, a dimensão social pode ser alcançada pois pensar a gestão de conhecimento no ECOSs envolve pensar a forma com as transações acontecem. Com a existência de equipes de desenvolvimento distribuídas, as equipes de teste, também distribuídas, devem ser capazes de realizar um planejamento dos testes, que possa permitir e certificar as contribuições dos diversos *stakeholders*, como o caso da AppleStore. As redes que se formam entre atores e artefatos devem ser identificadas e analisadas com base na rede sócio-técnica (Lima *et al.*, 2013), a fim de verificar quais são as equipes com maior potencial de impacto sobre a plataforma por desenvolverem o maior percentual de contribuições e/ou por requererem maior demanda de testes. Isso pode mostrar também os principais benefícios e fraquezas em um ambiente de DDS no contexto de ECOSs.

Por fim, na relação transacional-social, os temas *gestão do projeto/processo* e *stakeholders* emergem como cruciais para o sucesso de projetos com DDS em ECOSs, pois isso requer uma análise e manutenção do mapeamento entre as proposições e percepções de valor dos *stakeholders*, visando maximizar esta relação (Santos & Werner, 2012). No caso, a gestão deve considerar os diferentes níveis em que os ECOSs podem ser observados (Jansen *et al.*, 2009), i.e., o nível da organização (aquela que centraliza a plataforma), o nível da rede de produção de software (as organizações que fornecem e que consomem da plataforma) e o nível dos ECOS (os concorrentes e demais ECOSs que coexistem na indústria de software globalizada). Para isso, um *framework* para monitoramento e controle dos testes deve ser estabelecido, tanto considerando o nível local (desenvolvedores externos e usuários) como global (organização centralizadora), a fim de manter a qualidade dos produtos e serviços construídos a partir da plataforma. Estabelece-se assim a ligação entre as questões culturais e a comunicação, dado que elas impactam a forma das pessoas de comunicarem.

5. Conclusão

Os desafios do DDS ressaltam o tratamento das questões econômicas e sociais da ES e ainda incluem o contexto das arquiteturas e/ou plataformas comuns e/ou relacionadas com a formação de redes de desenvolvedores externos e usuários, conhecidos ou não, gerando os ECOSs. Neste contexto, o sucesso de projetos com DDS depende do planejamento e controle dos testes, de modo que se torna necessário compreender que elementos impactam o planejamento de testes nas diferentes combinações de cenários de DDS em geral e que também são afetados pelo contexto dos ECOSs. Este artigo identificou e classificou elementos que impactam o planejamento de testes em ambientes de DDS a partir de uma agenda de pesquisa, e também discutiu seus desdobramentos no contexto dos ECOSs. Considerando a carência de pesquisas sobre testes e DDS em ECOSs, este trabalho contribui no sentido de integrar grupos de pesquisa em torno desses temas a fim de construir soluções nessa direção. Como trabalhos futuros, pretende-se adaptar um processo de teste para contemplar os elementos discutidos nas Seções 3 e 4, indicando artefatos, papéis, atividades e ferramentas que possam apoiar o tratamento de questões econômicas e sociais no planejamento de teste com DDS.

Referências

Albert, B., Santos, R., Werner, C. (2012) "A Study on Software Components Governance Based on SOA Governance Elements". In: VI SBCARS, Natal, Brasil, 120-129.

Angeren, J., Kabbedjik, J., Jansen, S., Popp, K.M. (2011) "A Survey of Associate Models used within Large Software Ecosystems". In: 3rd IWSECO, ICSOB Workshops, Brussels, Belgium, 27-39.

Arndt, J., Dibbern, J. (2006) "Co-Innovation in a Service Oriented Strategic Network". In: IEEE International Conference on Services Computing, Chicago, USA, 285-288.

Bosch, J., Bosch-Sijtsema, P. (2010) "From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems", Journal of Systems and Software 83, 1, 67-76.

Causevic, A., Sundmark, D., Punnekkat, S. (2010) "An Industrial Survey on Contemporary Aspects of Software Testing". In: 3rd International Conference on Software Testing, Verification and Validation, Paris, France, 393-401.

Campbell, P., Ahmed, F. (2010) "A Three-dimensional View of Software Ecosystems". In: 2nd IWSECO, ECSA Workshops, Copenhagen, Denmark, 81-84.

Chaves, A., Huzita, E., Vieira, V., Steinmacher, I. (2010) "A Context Conceptual Model for a Distributed Software Development Environment". In: 22nd SEKE, Redwood City, USA, 437-442.

Collins, E., Macedo, G., Maia, N., Dias-Neto, A. (2012) "An Industrial Experience on the Application of Distributed Testing in an Agile Software Development Environment". In: 7th IEEE ICGSE, Porto Alegre, Brasil, 190-194.

Hanssen, G., Dyba, T. (2012) "Theoretical Foundations of Software Ecosystems". In: 4th IWSECO, ICSOB Workshops, Boston, USA, 6-17.

Huzita, E., Silva, C., Wiese, I., Tait, T., Quinaia, M., Schiavoni, F. (2008) "Um Conjunto de Soluções para Apoiar o Desenvolvimento Distribuído de Software". In: II WDDS, Campinas, Brasil, 101-110.

Huzita, E., Leal, G., Balancieri, R., Tait, T., Cardoza, E., Penteado, R., Vivian, R. (2012)"Knowledge and Contextual Information Management in Global Software Development: Challenges and Perspectives". In: VI WDDS, ICGSE Workshops, Porto Alegre, Brasil, 43-48.

Jain, R.; Poston, R. & Simon, J. (2011) "An Empirical Investigation of Client Managers' Responsibilities in Managing Offshore Outsourcing of Software-Testing Projects", IEEE Transactions on Engineering Management, v. 58, n. 4 (November), 743-757.

Jansen, S., Cusumano, M., Brinkkemper, S. (eds.) (2013) "Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry". Edward Elgar Publishing.

Jansen, S., Finkelstein, A., Brinkkemper, S. (2009) "A Sense of Community: A Research Agenda for Software Ecosystems". In: 31st ICSE, Vancouver, Canada, 187-190.

Jiménez, M., Piattini, M., Vizcaíno, A. (2009) "Challenges and Improvements in Distributed Software Development: A Systematic Review", Advances in Software Engineering, 3 (Jan), 1-16.

Juristo, N., Moreno, A., Vegas, S. (2004) "Reviewing 25 Years of Testing Technique Experiments", Empirical Software Engineering 9, 1-2 (Mar), 7-44.

Lima, T., Santos, R., Werner, C. (2013) "Apoio à Compreensão das Redes Socio-técnicas em Ecossistemas de Software". In: II BraSNAM, XXXIII CSBC Workshops, Maceió, Brasil.

McGregor, J., Sykes, D. (2001) "A Practical Guide to Testing Object-oriented Software".

Maia, N., Dias Neto, A., Collins, E,. Macedo, G. (2012) "Aplicando Testes Ágeis com Equipes Distribuídas: Um Relato de Experiência". In: XI SBQS, Fortaleza, Brasil, 365-372.

Messerschmitt, D., Szyperski, C. (2003) "Software Ecosystem: Understanding an Indispensable Technology and Industry". The MIT Press.

Pinto, A., Almeida, A., Morais, E. (2009) "Collaborative Software Development Process for Geographically Distributed Teams". In: III WDDS, Fortaleza, Brasil, 89-97.

Rocha, R., Costa, C., Prikladnicki R., Azevedo, R., Farias Jr, I., Meira, S. (2010) "Desafios nas Fases do Ciclo de Vida de Projetos Distribuídos". In: IV WDDS, CBSoft Workshops, Salvador, Brasil, 34-41.

Santos, R., Werner, C. (2012) "ReuseECOS: An Approach to Support Global Software Development through Software Ecosystems". In: VI WDDS, ICGSE Workshops, Porto Alegre, Brasil, 60-65.

Santos, R., Werner, C., Barbosa, O., Alves, C. (2012) "Software Ecosystems: Trends and Impacts on Software Engineering". In: XXVI SBES/ST, Natal, Brasil, 206-210.

Shah, H., Sinha, S., Harrold, M. (2011) "Outsourced, Offshored Software-Testing Practice: Vendor-Side Experiences". In: 6th IEEE ICGSE, Helsink, Finland, 131-140.

Desenvolvimento de *Handoffs* em Projetos de Software *Follow-the-Sun:* Um Relato de Experiência

Josiane Kroll, Jorge Luis Nicolas Audy

Faculdade de Informática (FACIN)
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
90.619-900 – Porto Alegre – RS – Brasil

josiane.kroll@acad.pucrs.br, audy@pucrs.br

Abstract. *In follow-the-sun (FTS) software projects, handoffs are performed daily at the beginning and at the end of each working day in each production site. However, handoffs development requires a great coordination, communication and collaboration effort from all team members involved in the project. In the literature, studies report handoffs development as one of the main challenges for FTS practice in the software industry. Thus, this study report the experience obtained developing handoffs in a software project with teams distributed in three production sites. Results obtained in this study describe challenges and solutions for handoffs practice in FTS software projects.*

Resumo. *Em projetos de software follow-the-sun (FTS), handoffs são realizados diariamente no início e no final de cada dia de trabalho em cada local de produção. Entretanto, o desenvolvimento de handoffs requer um grande esforço de coordenação, comunicação e colaboração de toda a equipe envolvida no projeto. Na literatura, estudos apontam o desenvolvimento de handoffs com um dos principais desafios para a prática de FTS na indústria de software. Dessa forma, este estudo relata a experiência obtida com o desenvolvimento de handoffs em um projeto de software com equipes distribuídas em três locais de produção. Os resultados obtidos neste estudo reportam desafios e soluções para a prática de handoffs em projetos de software FTS.*

1. Introdução

O desenvolvimento *follow-the-sun* (FTS) explora as diferenças de fusos horários entre locais de produção para acelerar o desenvolvimento de software [Tang et al. 2011]. Um dos principais desafios para a prática do desenvolvimento FTS é a realização de *handoffs*. Segundo Carmel, Espinosa e Dubinsky (2010), o desenvolvimento de *handoffs* requer um grande esforço de coordenação, comunicação e colaboração de toda a equipe envolvida no projeto. Além disso, *handoffs* devem ser rápidos e eficientes para que o ciclo de desenvolvimento do projeto seja reduzido. Se dificuldades de coordenação de *handoffs* afetarem a produtividade da equipe ao longo do projeto, o desenvolvimento FTS pode não trazer os benefícios esperados [Setamanit, Wakeland e Raffo 2007].

Neste estudo, é reportada a experiência obtida com o desenvolvimento de *handoffs* em um projeto de software FTS. Seguindo as regras do desenvolvimento FTS [Carmel e Espinosa 2011], *handoffs* foram realizados no início e no final de cada dia

trabalho em cada local de produção. Dados dos *handoffs* realizados entre a equipe foram coletados e são apresentados e analisados neste estudo. A principal contribuição deste estudo está relacionada a identificação dos desafios enfrentados pela equipe FTS para o desenvolvimento de *handoffs* e as soluções dadas a estes.

Este artigo está organizado da seguinte forma: na seção 2 são apresentados os principais conceitos envolvidos neste estudo e os trabalhos relacionados. Na seção 3, é descrito o método de pesquisa utilizado. Na seção 4, os resultados obtidos são apresentados e analisados. Por fim, na seção 5 são feitas as considerações finais acerca do estudo.

2. Desenvolvimento de Software *Follow-the-Sun*

Follow-the-sun (FTS) é uma estratégia de desenvolvimento de software aplicada para o contexto de projetos de *Global Software Development*[1] (GSD). O FTS se caracteriza pelo desenvolvimento de software vinte quatro horas com equipes distribuídas em diferentes locais de produção e fusos horários [Visser e Solingen 2009].

No desenvolvimento FTS, as equipes são distribuídas de maneira que quando uma equipe encerra o seu dia de trabalho, a outra equipe localizada em um diferente local de produção e fuso horário assume as tarefas dando continuidade ao dia de trabalho [Treinen e Miller-Frost 2006].

O principal objetivo da estratégia FTS é reduzir o ciclo de desenvolvimento do projeto ou *time-to-market*. Segundo Carmel e Espinosa (2011), o desenvolvimento FTS não oferece outras vantagens além da redução do tempo de desenvolvimento do projeto.

2.1. *Handoffs*

No desenvolvimento FTS, no final de cada dia de trabalho, tarefas inacabadas são transferidas para o próximo local de produção, o qual receberá as tarefas para dar início ao seu dia de trabalho. Os locais de produção são separados por diferentes fusos horários, o que permite que uma equipe encerre o seu dia de trabalho enquanto a outra equipe inicia [Visser e Solingen 2009].

A continuidade do trabalho envolve ciclos de transferência de tarefas entre as equipes, os quais são chamados de *handoffs* [Solingen e Valkema 2010]. O termo *handoff* é utilizado na literatura para designar o processo de transição de tarefas entre locais de produção [Carmel e Espinosa 2011].

O desenvolvimento de *handoffs* diários criam dependências entre locais de produção. A equipe que inicia o dia de trabalho depende das informações de *handoff* para dar continuidade ao trabalho. Na literatura, o desenvolvimento de *handoffs* é apontado com um dos principais desafios para a prática de FTS [Sooraj e Mohapatra 2008; Carmel e Espinosa 2011].

2.2. Trabalhos Relacionados

Na literatura de FTS, o desenvolvimento de *handoffs* ainda é tema pouco explorado. O principal estudo que discute o desenvolvimento de *handoffs* foi realizado

[1] O termo será mantido em inglês, visto que a tradução em alguns casos pode comprometer o uso do termo.

por Hess e Audy (2012). Nesse estudo, os autores propõem um processo para o desenvolvimento das sessões diárias de *handoff*. O processo proposto visa amenizar as dificuldades inerentes aos projetos que utilizam FTS como estratégia de software, focando na fase de desenvolvimento do ciclo de vida de software. Esse processo foi desenvolvido baseado nos conceitos *Composite Persona* (CP), *24hr Design and Development* e também utilizou algumas técnicas provenientes das metodologias ágeis, como por exemplo, *Test-driven development (TDD)* e *stand-up meetings*. Resultados encontrados nesse estudo mostram que o processo contribui para amenizar as dificuldades encontradas para o desenvolvimento das sessões de *handoff*.

Outros estudos como o de Ramesh e Dennis (2002) e Carmel (2006), embora não tenham focado diretamente no desenvolvimento de *handoffs* citam características e problemas enfrentados pelas equipes para o desenvolvimento das sessões de *handoff* em projetos FTS. Nesses estudos são observados principalmente problemas referentes à coordenação de *handoffs* diários. Ramesh e Dennis (2002) reportam problemas enfrentados pela equipe para gerenciar versões de código e documentos. Além disso, foram observados muitos problemas de sincronismo na comunicação diária realizada por telefone e por email. No estudo realizado por Carmel (2006), também são citados muitos problemas relacionados ao gerenciamento de *handoffs* envolvendo a falta de *overlap* de tempo entre os locais de produção e a falta de comunicação em tempo real.

3. Metodologia de Pesquisa

Essa seção fornece detalhes de um estudo de caso realizado em uma grande empresa de desenvolvimento de software global. O estudo de caso constou do desenvolvimento de um projeto de software no modo FTS. O principal objetivo deste estudo é analisar o desenvolvimento e gerenciamento de handoffs em projetos de software FTS.

O projeto adotou práticas do método Scrum e se restringiu a fase de desenvolvimento do ciclo de vida do software. Seguindo o método Scrum, o desenvolvimento do projeto foi dividido em *sprint* 1 e *sprint* 2. Neste estudo é apresentada a experiência obtida no *sprint* 1. Nas próximas subseções são dados detalhes do planejamento e desenvolvimento das atividades do projeto.

3.1. Configuração da equipe

A equipe designada para desenvolver o projeto de software foi distribuída em três locais de desenvolvimento: México, Austrália e Índia. Para cada local de desenvolvimento foram alocados dois desenvolvedores.

Os desenvolvedores do México estavam em fase de treinamento e possuíam menos de um ano de experiência. Na Austrália estavam os desenvolvedores mais experientes, entre 8 e 10 anos de experiência. Na Índia, os desenvolvedores tinham entre dois e um ano de experiência, respectivamente. A equipe alocada para o projeto também constou de um gerente de projeto e um *scrum master* ambos localizados na Índia.

3.2. Planejamento das sessões de *handoff*

Para o desenvolvimento das sessões de *handoff* optou-se por chamadas telefônicas no final e no início de cada dia de trabalho em cada local de produção. Também foi disponibilizada a ferramenta *Microsoft Office Communicator* e um repositório de dados da própria empresa.

A ferramenta *Microsoft Office Communicator* foi disponibilizada com o objetivo de permitir que a equipe resolvesse questões pontuais do projeto durante as sessões de *handoff* ou para suprir problemas de conexão, caso estes ocorressem.

O repositório de dados utilizado constou de um portal da empresa que tinha a finalidade de armazenar documentos do projeto e código fonte. No portal, a equipe FTS podia ver os documentos e código fonte gerados pela equipe antes de cada sessão de *handoff*.

Para armazenar os dados de *handoff* foi criado um documento no Microsoft Excel chamado *Template Handover*. Esse documento era preenchido pelos membros da equipe que estavam finalizando o dia de trabalho com todas as informações necessárias para que a próxima equipe desse continuidade ao dia de trabalho. Esse documento era utilizado pela equipe durante as sessões de *handoff* para discutir as tarefas que haviam sido realizadas, as tarefas em progresso e como o trabalho deveria ser continuado. Na Figura 1 é apresentado o *Template Handover,* o qual foi utilizado pela equipe durante as sessões de *handoff*.

1		**Task A: *<Enter Name of Task here>***		**Status of Task**	**Allocated to ()**	
2						
3						
4		**Sub-Task Breakdown**	**Status** (Complete, In Progress, Yet to Start)	**Last Updated by**	**Date of Last Update**	
5	1					
6	2					
7	3					
8	4					
9	5					
10	6					
11		**Task Handover form to be used for each handover**				
12			**Country Name**	**CP member name**		
13		From Location A				
14		To Location B				
15		Date of handover				
16						
17		**Planned Tasks for the Day**	**Actual Work carried out**	**Work that could not be done**	**Issues / Problems encountered**	**Suggested Work items for the next location**
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28		**Any overall Comments / Suggestions / Issues faced**				
29						
30						

Figura 1. *Template Handover.*

As tarefas do projeto foram disponibilizadas no portal do projeto em um arquivo *backlog*. A criação desse arquivo foi baseada nas práticas do método ágil Scrum.

A alocação de tarefas seguiu o conceito CPro proposto por Denny et. (2008). O CPro é um processo de software ágil que considera a distribuição de tarefas para CPs (*Composite Personas).* Cada CP é formado por pelo menos um membro de cada local de desenvolvimento.

3.3. Coleta de dados

Sessões de treinamento foram realizadas com os membros da equipe para explicar a teoria de FTS e fornecer detalhes da abordagem de software que seria utilizada para desenvolver o projeto. Guias e documentos utilizados durante as sessões de treinamento foram disponibilizados no portal do projeto e dúvidas remanescentes do

treinamento foram respondidas por email. Também foram realizadas sessões de treinamento para clarificar o método Scrum adotado no desenvolvimento do projeto, visto que somente os desenvolvedores da Austrália tinham experiência com o uso de métodos ágeis.

Para gerenciar as sessões de *handoff* e assegurar que as regras do desenvolvimento FTS seriam seguidas pela equipe, um documento com 12 itens de verificação foi utilizado para a coleta de dados de *handoffs* entre México e Índia e entre Índia e Austrália. O documento utilizado é apresentado na Figura 2.

Itens observados	Uso (Sim/Não)	Observações
A qualidade da ligação foi boa?		
Tarefas extras foram enviadas por email?		
O *template handoff* foi usado?		
Tarefas foram explicadas pelo site anterior?		
Tarefas foram sumarizadas pelo site receptor?		
Todas as dúvidas foram respondidas?		
A reunião foi feita por chamada telefônica?		
A tela foi compartilhada?		
IM chat foi usado?		
Todos estavam presentes?		
Duração de ligação		
Notas/ dúvidas		

Figura 2. Documento para a coleta de dados de *handoffs*.

Também foram realizadas reuniões com o gerente do projeto com objetivo de identificar melhorias para o desenvolvimento de *handoffs*. O gerente do projeto participava das sessões de *handoff* entre Austrália e Índia e Índia e México. Ele participava no início das sessões de *handoff* para discutir tarefas alocadas, tirar dúvidas e verificar o progresso do projeto.

Ao final do *sprint* 1, seguindo as recomendações do método Scrum foi realizada a retrospectiva do *sprint* 1. A retrospectiva do *sprint* 1 consistiu da aplicação de um questionário para avaliar as atividades dentro do projeto, a qual incluiu as atividades de *handoff*. Os membros da equipe responderam o que foi bem, o que não foi bem e o que deveria ser feito para melhorar as práticas de software adotadas para o desenvolvimento do projeto. Nesse estudo são reportados os resultados obtidos relacionados ao desenvolvimento das atividades de *handoff*. Informações sobre os resultados obtidos com outras atividades avaliadas e referentes ao *sprint* 2 são reportadas no estudo de Kroll et al. (2013).

4. Resultados e Análise

4.1. Dados das sessões de *handoff*

Com os resultados obtidos com a aplicação do documento para a coleta de dados das sessões *handoff* pode-se observar que a qualidade da comunicação da equipe, considerando diferentes sotaques, linguagens e o meio de comunicação utilizado, foi mais baixa durante as primeiras sessões de *handoff*. Observaram-se principalmente problemas de comunicação relacionados aos diferentes sotaques e a rápida dicção dos

participantes. Após minimizar esses desafios fornecendo instruções para a equipe falar devagar e claramente, pode-se observar que fluxo de comunicação entre a equipe melhorou.

O arquivo *backlog* disponibilizado no início do projeto, causou problemas relacionados ao não entendimento das tarefas que deveriam ser realizadas pela equipe. Para minimizar esse problema, o gerente do projeto assumiu a responsabilidade de enviar um email diário alocando tarefas para cada CP.

Com relação ao uso do *template handover* não foram reportados problemas. Também não foram reportados problemas com relação à transferência de informações das tarefas. Cada CP discutia com o seu par CP as tarefas realizadas e os próximos passos seguindo o *template handover*.

O recurso de compartilhamento de telas oferecido pela ferramenta *Microsoft Office Communicator* foi utilizado somente em algumas situações. Entretanto, a equipe avaliou positivamente o uso do recurso de compartilhamento de telas para fornecer detalhes de documentos e código fonte.

Observou-se que em poucas sessões de *handoff* um dos membros CP não estava presente. Para suprir a falta de um membro CP, um email com informações detalhadas do *handoff* era enviado para o próximo CP depois da sessão de *handoff* ser encerrada.

As sessões de *handoff* duraram em média 38 minutos. A sessão mais longa foi entre Austrália e a Índia. Isso ocorreu porque os membros da Austrália e da Índia usaram algumas das sessões de *handoff* para realizar o planejamento de tarefas. A sessão de *handoff* mais longa entre esses dois locais de produção durou 1 hora e 10 minutos. Já a sessão mais curta foi entre o México e a Austrália com a duração de 11 minutos.

Todas as sessões de *handoff* foram realizadas por meio de chamadas telefônicas. O uso do *Microsoft Office Communicator* foi usado poucas vezes pela equipe.

O principal benefício obtido com a aplicação do documento para a coleta de dados das sessões *handoff* foi a identificação de problemas que estavam ocorrendo ao longo do projeto. Esses problemas foram minimizados a medida que eram identificados.

4.2. Reuniões realizadas com o gerente do projeto

As reuniões com o gerente do projeto contribuíram para identificar duas principais melhorias nas práticas de software adotadas no projeto:

- *Atribuição de um CP responsável pela tarefa:* consiste na eleição de um CP, o qual terá o papel de assegurar que a tarefa será completada pela equipe. No início do projeto muitas tarefas não estavam sendo completadas pela equipe, o que resultou em horas de trabalho extra no final do projeto.
- *Alocação de tarefas por dia de trabalho:* consiste no envio de um email diário alocando tarefas para cada CP. Essa solução contribui para definir prioridades de tarefas e reduzir problemas enfrentados pela equipe para categorizar uma tarefa dentro de um arquivo *sprint backlog*.

4.3. Aplicação do questionário de retrospectiva do *sprint* 1

Os resultados obtidos nos questionários com relação a avaliação das atividades de *handoff* mostram que a equipe FTS enfrentou dificuldades principalmente para realizar *handoffs* de final de semana e feriados. Os membros da equipe que assumiam as tarefas após o final de semana ou feriado não recebiam as informações de *handoff*, visto que a equipe anterior havia cumprido o seu dia de trabalho até data anterior.

Na avaliação geral da equipe, *handoffs* foram realizados adequadamente durante o projeto. Entretanto, a equipe reportou algumas dificuldades no início do projeto. Uma das respostas obtidas reporta a dificuldade para compartilhar a informação com o local que está recebendo a tarefa. Já outra resposta sugere que todas as tarefas planejadas sejam disponibilizadas no início do projeto como forma de melhorar o entendimento da equipe com relação ao produto final.

As ferramentas de comunicação utilizadas pela equipe para realizar as sessões de *handoff* foram positivamente avaliadas. Entretanto alguns aspectos foram observados para melhorar o fluxo de comunicação entre a equipe:

- Sessões de *handoff* mais curtas com uma melhor distribuição do tempo entre os membros CPs;
- Redução de sotaques e uso de linguagem adequada ao contexto.

O repositório de dados utilizado para compartilhar o código fonte e os documentos gerados pela equipe contribuiu para o controle de versões e para manter a pontualidade da equipe com relação ao *upload* de arquivos do projeto. Também foi reportada a falta de experiência com o uso do repositório de dados, o que dificultou o desenvolvimento de algumas atividades no início do projeto.

A equipe também mencionou pontos fortes e fracos na abordagem adotada para o desenvolvimento das sessões de *handoff*. Entre os pontos fortes estão à pontualidade da equipe e a boa utilização do tempo. Já os pontos fracos reportam a falta de padrões e *templates*, inapropriada alocação de tarefas no início do projeto, intensa comunicação e falta de guias do projeto.

5. Considerações finais

Neste estudo é reportada a experiência obtida com o desenvolvimento de *handoffs* em um projeto de software FTS. O principal desafio encontrado foi referente a criação de uma abordagem para o desenvolvimento de *handoffs*, visto que este é um tema ainda pouco explorado na literatura da área. Além disso, os desafios de coordenação e comunicação que envolve a prática de *handoffs* ficaram muito mais evidentes ao longo do projeto.

A experiência obtida com o desenvolvimento deste estudo mostra que o desenvolvimento de *handoffs* deve considerar aspectos culturais, de colaboração e principalmente de comunicação e coordenação. Os aspectos culturais e de colaboração não são menos importantes, mas desafios relacionados a estes podem ser reduzidos se boas práticas de comunicação e coordenação forem empregadas.

A equipe alocada para o desenvolvimento do projeto teve um papel importante para minimizar as dificuldades encontradas. A criação de um email diário para a alocação de tarefas e a eleição de um responsável por cada tarefa, são alguns exemplos.

Em trabalhos futuros, serão investigadas práticas de software que permitam que *handoffs* sejam praticados efetivamente em projetos FTS. Dessa forma, resultados referentes ao aprimoramento e aprofundamento da área de pesquisa podem surgir.

Referências

Carmel, E. (2006) "Building your Information Systems from the Other Side of the World: How Infosys manages time differences", *MIS Quarterly Executive*, 5(1).

Carmel, E. and Espinosa, J. A. (2011) I'm Working While They're Sleeping: Time Zone Separation Challenges and Solutions, *Kindle Edition*, 188 p.

Carmel, E., Espinosa, J. and Dubinsky, Y. (2010) "Follow the Sun Workflow in Global Software Development", *Journal of Management Information Systems,* Vol. 27 No. 1, 17 – 38.

Denny, N., Mani, S., Nadella, R. S., Swaminathan, M. and Samdal, J. (2008) "Hybrid Offshoring: Composite Personae and Evolving Collaboration Technologies", IRMJ 21(1): 89-104.

Hess, E. and Audy, J. L. N. (2012) "FTSProc: a Process to Alleviate the Challenges of Projects that Use the Follow-the-Sun Strategy", *International Conference on Global Software Engineering (ICGSE)*, Porto Alegre, Brazil.

Kroll, J., Prikladnicki, R., Audy, J. L. N, Carmel, E. and Fernandez, J. (2013) "A Feasibility Study of Follow-the-Sun Software Development for GSD Projects" *25th International Conference on Software Engineering & Knowledge (SEKE 2013).*

Ramesh, V. and Dennis, A. (2002) "The object oriented team: Lessons for virtual teams from global software development", In HICSS '02: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, volume 1.

Setamanit, S. O., Wakeland, W. and Raffo D. (2007) "Improving Global Software Development Project Performance Using Simulation. Management of Engineering and Technology", Portland International Center, pp. 2458-2466.

Solingen, V. R. and Valkema, M. (2010) "The Impact of Number of Sites in a Follow the Sun Setting on the Actual and Perceived Working Speed and Accuracy: A Controlled Experiment", *5th International Conference on Global Software Engineering (ICGSE),* 165- 174.

Sooraj P. and Mohapatra P. K. J. (2008) "Modeling the 24-h software development process", *Strategic Outsourcing: An International* Journal, 122-141.

Tang, J. C., Zhao, C., Cao, X. and Inkpen, K. (2011) "Your time zone or mine?: a study of globally time zone-shifted collaboration", *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW '11).* ACM, New York, NY, USA, 235-244.

Treinen, J. J. and Miller-Frost, S. L. (2006) "Following the Sun: Case Studies in Global Software Development", *IBM Systems Journal*, Volume 45, Number 4.

Visser, C. and Solingen, V. R. (2009) "Selecting Locations for Follow-the Sun Software Development: Towards A Routing Model", *Fourth International Conference on Global Software Engineering (ICGSE).*

CollabCode – Ferramenta para apoio ao Desenvolvimento Distribuído de Software

Alexandre S. Wolf[1], Maurício S. Silva[1]

[1]Centro Universitário UNIVATES (UNIVATES)
Rua Avelino Tallini, 171 | Bairro Universitário – 95.900-000 – Lajeado – RS – Brazil
{awolf,mss}@univates.br

Abstract. *The current global economic environment is causing more and more companies make their software development processes global. This adaptation behind a number of challenges for development teams, since the communication, collaboration and management activities should continue to exist even in geographically distributed teams. This article aims to present the CollabCode, tool engineered to create an online development environment capable of meeting the requirements of geographically distributed development teams.*

Resumo. *O atual cenário da economia mundial está fazendo com que cada vez mais as empresas de desenvolvimento de software tornem seus processos globais. Essa adaptação trás uma série de desafios para as equipes de desenvolvimento, uma vez que a comunicação, colaboração e o gerenciamento de atividades devem continuar a existir mesmo em equipes geograficamente distribuídas. O presente artigo tem como objetivo apresentar o CollabCode, ferramenta projetada visando criar um ambiente de desenvolvimento online capaz de suprir as necessidades de equipes de desenvolvimento geograficamente distribuídas.*

1. Introdução

O desenvolvimento de software de forma distribuída já é uma realidade para muitas empresas. Esta mudança na forma como o software é desenvolvido trás novos desafios, uma vez que em um cenário globalizado é preciso haver coordenação, gerenciamento, colaboração e comunicação. Estes quatro requisitos precisam ser constantemente verificados e equalizados, de forma que as potencialidades providas pela distribuição do processo de desenvolvimento de software não tornem-se armadilhas capazes de colocar em risco os projetos ou até mesmo gerar atrito entre as equipes.

Neste cenário, é possível encontrar ferramentas que atendam isoladamente cada um desses requisitos. Isso faz com que muitos desenvolvedores precisem ter em suas estações de trabalho uma série de aplicações que permitam a realização de suas atividades. É comum a utilização de sistemas de troca de mensagens, sistemas de compartilhamento de tela, softwares gerenciadores de atividades, sistema de compartilhamento de arquivos, IDEs, dentre outros, por parte das equipes de desenvolvimento distribuído.

Esta necessidade de instalar aplicações para que o trabalho de desenvolvimento ocorra, limita a mobilidade dos membros da equipe. Em um cenário hipotético onde um desenvolvedor está em um congresso quando recebe um chamado urgente para solucionar algum problema ocorrido no sistema que está/foi desenvolvido, ele só poderá trabalhar em uma solução caso estiver com seu notebook e possuir todas as ferramentas necessárias instaladas. Caso esse desenvolvedor tenha optado por não levar seu notebook, apenas seu *tablet* ou *smartphone*, a equipe precisaria aguardar até que ele tivesse acesso às ferramentas que o possibilitem trabalhar na solução do problema.

Para solucionar o problema de mobilidade já descrito, centralizando todas as ferramentas necessárias para o desenvolvimento de sistemas *web*, foi projetado o CollabCode. O CollabCode é uma aplicação *web* que possui um conjunto de ferramentas para comunicação, gerenciamento de atividades e colaboração capazes de suprir as ferramentas instaladas nas estações de trabalho, sendo necessário apenas um navegador atualizado para sua utilização.

Na próxima seção será apresentada uma revisão de literatura que tem por objetivo proporcionar uma visão geral sobre o estado da arte no desenvolvimento distribuído de software, também apresentar as ferramentas necessárias para que a colaboração entre membros de equipes distribuídas, além de expor a problemática que motivou o desenvolvimento do CollabCode. Após a revisão de literatura, a ferramenta CollabCode é apresentada e, ao final, são apresentadas as considerações finais deste artigo.

2. Revisão de literatura

Nesta seção é apresentada uma breve descrição sobre a colaboração entre membros de equipes, o modelo de colaboração 3C e o que é necessário em um software colaborativo. Ao final é apresentada visão geral sobre o que é o Desenvolvimento Distribuído de Software (DDS).

2.1. Colaboração

Quando existe colaboração entre os membros de uma equipe na execução de uma determinada tarefa, os resultados obtidos são melhores do que cada membro trabalhando de forma individual [FUKS, RAPOSO e GEROSA 2002]. Isso ocorre porque o processo de colaboração possibilita a complementação dos conhecimentos e capacidades dos membros em cooperação. Na medida em que o trabalho é executado, cada membro pode expor seu ponto de vista, suas vivências e experiências, o que contribui de forma positiva para o surgimento de formas mais eficientes de resolver determinados problemas.

A colaboração também possibilita o refinamento dos conhecimentos de cada membro, além de ter caráter motivador [FUKS, RAPOSO e GEROSA 2002]. Em um processo colaborativo, cada membro avalia e é constantemente avaliado pelas demais pessoas do grupo, mesmo que indiretamente, fazendo com que a equipe não fique acomodada. Além disso, a troca de ideias e experiências permite que cada membro aprimore seus conhecimentos.

2.1.1. Modelo 3C

O modelo 3C, apresentado por Ellis et al. (1991), define que a colaboração é sustentada pela comunicação, coordenação e cooperação. Neste modelo, a cooperação só existe quando os membros da equipe realizarem a troca de informações (comunicação), gerando compromissos e atividades. Estas atividades devem ser coordenadas de forma que possam ser executadas em cooperação. A Figura 1, extraída de Gerosa et al. (2005), apresenta detalhes do funcionamento do modelo 3C de colaboração.

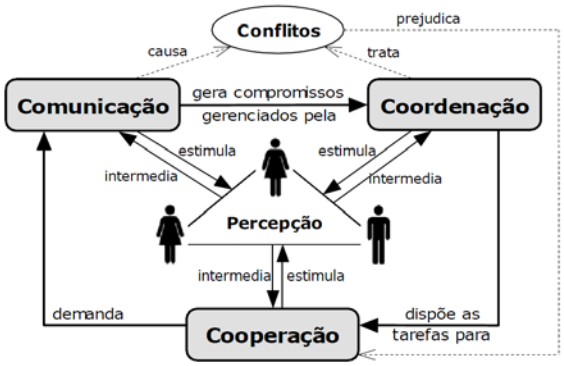


Figura 1 - Modelo de colaboração 3C

A percepção neste processo é uma forma de adquirir informações por meio dos sentidos, através de elementos não-verbais existentes no ambiente [GEROSA 2006]. Estes elementos podem ser cores, expressões faciais, linguagem utilizada na conversação entre outros. Nesse sentido, a comunicação deve ser constantemente estimulada e intermediada pela coordenação para não gerar conflitos, que por sua vez prejudiquem a cooperação entre os membros [FUKS, RAPOSO e GEROSA 2002].

2.2. Software Colaborativo

Um software colaborativo precisa ser capaz de dar suporte às atividades dos membros da equipe. Deve possibilitar a cooperação, comunicação e coordenação das atividades dos grupos de trabalho, independente de seu tamanho, composição ou localização.

O software colaborativo é um modelo de software que mudou a forma como o trabalho é visto nos escritórios [TOMMARELLO e DEEK 2002]. Segundo Nunamaker (1999), o crescimento constante do teletrabalho e da utilização de consultorias externas, influencia na dispersão física dos membros das equipes de trabalho em reuniões. Segundo o autor, a presença quase onipresente do acesso à *web* acelerou ainda mais o processo de dispersão das equipes e, segundo ele, a tendência é que cada vez mais as redes de comunicação e de trabalho, que antes eram locais, migrem para redes de longa distância.

Gerosa (2006) sustenta a tese de que a utilização do modelo 3C de colaboração no desenvolvimento de softwares colaborativos proporciona os recursos necessários para que o software desenvolvido evolua também no suporte à colaboração.

2.3. Desenvolvimento Distribuído de Software (DDS)

Empresas dos mais variados nichos de mercado, inclusive de desenvolvimento de software, estão encontrando na globalização um diferencial estratégico [AUDY e

PRIKLADNICKI 2008]. Segundo Audy e Prikladnicki (2008), a pressão por custos e a sofisticação dos meios de comunicação tem tornado cada vez menos competitivo desenvolver software no mesmo espaço físico, organização ou país. Os autores afirmam ainda que melhorias feitas na engenharia de software em conjunto com o surgimento de novas ferramentas e métodos, tem facilitado o surgimento de equipes de trabalho em âmbito global, diminuindo custos e agregando qualidade ao processo de desenvolvimento de software.

São vários os fatores que tornam o desenvolvimento distribuído de software atraente às empresas. Lopes (2004) apresenta o que acreditar serem as principais razões, são elas:

- Sinergia cultural: a diversidade cultural amplia a criatividade;
- Mercado global: incentivos econômicos e diminuição de custos como diferenciais competitivos;
- Escala: equipes muito grandes são difíceis de serem gerenciadas e coordenadas. Equipes menores estrategicamente distribuídas podem facilitar esse gerenciamento;
- *Time-to-market*: diminuir o tempo de desenvolvimento do software distribuindo equipes ao redor do mundo e desenvolvendo o software 24 horas por dia;
- Rigor e experiência: equipes centralizadas de desenvolvimento tendem a utilizar técnicas informais e descuidar tanto no uso de metodologias quanto na qualidade. Equipes de DDS, por procurar a excelência em comunicação, tendem a melhorar os processos relacionados à documentação;
- Demanda e custos: a demanda do mercado de desenvolvimento de software é muito maior do que o recurso humano disponível, o que impacta diretamente no custo do desenvolvimento.

3. CollabCode

Em virtude da não ter sido encontrado uma solução que contemple todas as necessidades do desenvolvimento distribuído de software, foi projetado o CollabCode. Existem diversas soluções no mercado que atendem cada qual a uma necessidade específica, seja ela comunicação, coordenação ou cooperação. A proposta do CollabCode é fornecer em uma única solução todas as ferramentas necessárias para que equipes geograficamente distribuídas possam trabalhar, sem a necessidade de instalação de qualquer aplicação na estação de trabalho, requerendo apenas um navegador e acesso a internet.

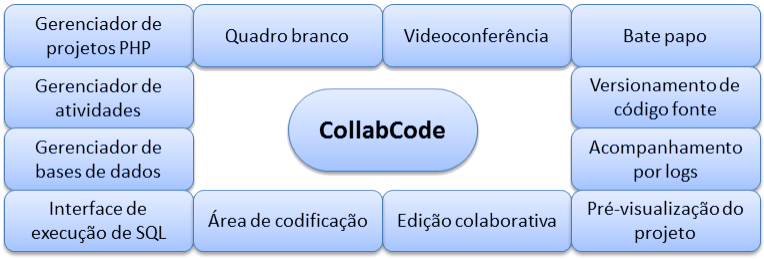
O CollabCode foi projetado considerando o modelo 3C de colaboração. Isso faz com que o CollabCode seja uma solução que preocupa-se diretamente com a comunicação e cooperação entre os membros da equipe, além de permitir o gerenciamento de atividades inerentes ao projeto.

A ferramenta é uma aplicação *web* desenvolvida em PHP com suporte para o banco de dados PostgreSQL. Toda a estrutura do CollabCode foi criada utilizando o Sistema Operacional Linux, que segundo Nemeth, Snyder e Hein (2007) é o Sistema Operacional mais utilizado em servidores *web*. Além disso, o CollabCode faz uso de

tecnologias como WebSockets, que é utilizado para a colaboração durante a escrita de código, e Subversion, utilizado para fazer o versionamento do código fonte.

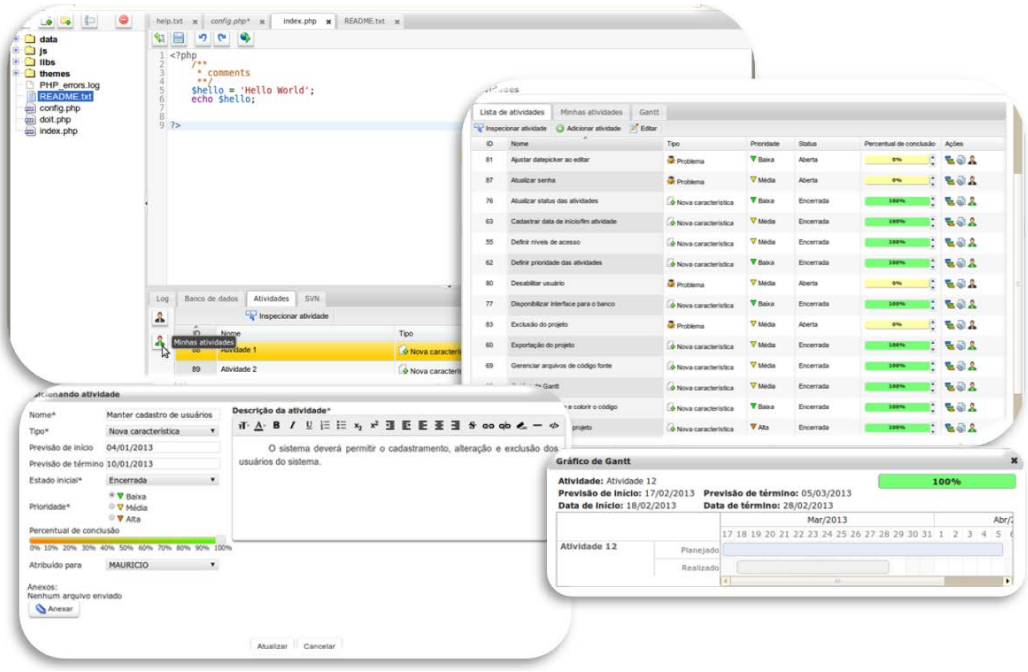
Dentre os principais objetivos do CollabCode, destacam-se dois, onde o primeiro é fornecer o ferramental necessário para que o desenvolvedor consiga codificar dentro do navegador sem a necessidade de utilização de qualquer outra ferramenta. Para isso a solução dispõe de uma IDE de desenvolvimento com a possibilidade de trabalhar com versionamento de arquivos e utilizar interface com banco de dados. O segundo objetivo é criar uma área de gerenciamento que possibilite a criação de projetos *web*, que utilizam como base a linguagem de programação PHP, e o gerenciamento das atividades inerentes ao projeto. Uma visão geral de todas as ferramentas disponíveis no CollabCode pode ver visualizada na Figura 2.

Figura 2 - Ferramentas disponíveis no CollabCode



O CollabCode foi desenvolvido utilizando diversas bibliotecas e componentes de interface. Dentre elas temos: o framework JQuery UI, a biblioteca Flexigrid, o editor CodeMirror e GanttView. Além disso, diversos *plugins* do JQuery foram utilizados, tais como: JQuery UI Layout e JQuery TE. A Figura 3 apresenta algumas das interfaces do CollabCode construída com estes componentes.

Figura 3 - Interfaces do CollabCode



3.1. Ferramentas de comunicação:

As ferramentas de comunicação existentes no CollabCode permitem que a comunicação entre os membros da equipe transcorra de diversas formas. As ferramentas de comunicação síncronas servem de apoio para que todos os envolvidos possam trocar informações de forma online e no momento em que for necessário. Já as ferramentas de comunicação assíncronas ajudam a identificar as interações dos usuários com as atividades e fornecem subsídios para a documentação do sistema, o qual está sendo desenvolvido. As ferramentas de comunicação disponíveis no CollabCode são:

- Bate papo: A solução possui um sistema de bate papo onde todos os membros do projeto podem interagir de forma síncrona. Além disso, o sistema cria salas que apenas quem está em edição colaborativa também possa utilizar;
- Videoconferência: O CollabCode dispõe de uma ferramenta que permite a conversação por áudio e vídeo. Esta ferramenta possui também opções de compartilhamento de tela e quadro branco;
- Comentários em atividades: Quando uma atividade é criada no CollabCode os membros da equipe podem trocar informações através de uma área de comentários relacionada a ela. É possível ainda adicionar arquivos e trechos de código fonte.

3.2. Ferramentas de Coordenação

O gerenciamento e coordenação em projetos DDS são tarefas que precisam funcionar corretamente para o bom andamento dos projetos. Em um software que propõe-se a gerenciar projetos DDS, são necessárias ferramentas que possibilitem a coordenação de atividades, onde estas devem permitir ao gerente de projetos, aferirem métricas, acompanhar a execução de cada atividade, realocar sempre que necessário o recurso humano disponível, entre outras possibilidades. Como forma de proporcionar estes recursos aos gerentes de projetos, o CollabCode dispõe das seguintes ferramentas:

- Gerenciamento de projetos: A ferramenta dispõe de uma interface para a criação e gerenciamento de projetos *web*. Ao adicionar um projeto, o usuário pode optar por importar um projeto através de repositório SVN ou criar um novo projeto. É possível também criar bases de dados e acompanhar estatísticas sobre o andamento do projeto;
- Gerenciamento de atividades: O CollabCode dispõe de uma interface onde é possível criar e gerenciar atividades, além de permitir seu acompanhamento através de gráficos de Gantt. Esta ferramenta possibilita também o acompanhamento da execução das atividades através de um percentual de conclusão e dos comentários adicionados pelos membros da equipe.

3.3. Ferramentas de colaboração

O CollabCode implementa várias ferramentas que possibilitam a colaboração entre os membros da equipe. Estas ferramentas permitem que as interações muitas vezes existentes em equipes presenciais sejam possíveis em equipes distribuídas geograficamente. As ferramentas disponíveis no CollabCode para a colaboração são:

- Edição colaborativa: A edição de código pode ser compartilhada entre os desenvolvedores. Caso dois ou mais desenvolvedores abram o mesmo arquivo, uma sessão de edição colaborativa é iniciada, permitindo a eles a edição cooperativa em tempo real do código fonte;
- Compartilhamento de tela: Esta ferramenta permite que um membro da equipe possa compartilhar área de trabalho com os demais membros da equipe. A ferramenta permite também que o computador que está sendo compartilhado possa ser controlado remotamente por outro membro da equipe;
- Quadro branco: O quadro branco é um recurso disponível na ferramenta de videoconferência. Com ele é possível que os membros da equipe interajam em tempo real na construção de artefatos de software que facilitem a compreensão de processos e de atividades a serem desenvolvidas.

4. Considerações finais

A dispersão geográfica dos membros de equipes de desenvolvimento está cada vez mais presente nas empresas de desenvolvimento de software. Essa realidade é favorável ao surgimento de novas soluções que possibilitem transformar os desafios gerados pelo DDS em diferenciais estratégicos.

Existem diversas ferramentas que atendem cada qual, a um determinado desafio gerado pelo DDS. Estas ferramentas foram estudadas e ajudaram a elucidar os requisitos considerados indispensáveis nesta primeira versão do CollabCode. Além disso, os estudos teóricos sobre o DDS e os modelos de colaboração, facilitaram o entendimento de como cada ferramenta deve comportar-se para que o ambiente de trabalho não interfira de forma negativa na realização das atividades.

Nesse sentido, o CollabCode é uma opção às atuais ferramentas existentes no mercado. Sua estrutura foi projetada de modo a tornar o processo de desenvolvimento de software mais ágil e facilitar a coordenação, colaboração e comunicação em equipes DDS, permitindo ainda a expansão de suas funcionalidades em trabalhos futuros.

Referências

Audy, Jorge L. N.; Prikladnicki, Rafael. (2008) “Desenvolvimento distribuído de software: desenvolvimento de software com equipes distribuídas”. Rio de Janeiro: Elsevier.

Ellis, C.A., Gibbs, S.J. & Rein, G.L. (1991) “Groupware - Some Issues and Experiences”. Communications of the ACM, Vol. 34, No. 1, pp. 38-58.

Fuks, H.; Raposo, A.B.; Gerosa, M.A. (2002) “Engenharia de Groupware: Desenvolvimento de Aplicações Colaborativas”. XXI Jornada de Atualização em Informática Anais do XXII Congresso da Sociedade Brasileira de Computação, Florianópolis.

Gerosa, Marco A. (2006) “Desenvolvimento de Groupware Componentizado com Base no Modelo 3C de Colaboração”. Tese (Doutorado) – Pós-Graduação em Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro.

Gerosa, M.A.; Pimentel, M.G.; Filippo, D.; Barreto, C.G.; Raposo, A.B.; Fuks, H. ; Lucena, C.J.P. (2005) “Componentes Baseados no Modelo 3C para o Desenvolvimento de Ferramentas Colaborativas”. Anais do 5º Workshop de Desenvolvimento Baseado em Componentes. Juiz de Fora.

Lopes, Leandro T.; (2004) “Um modelo de processo de engenharia de requisitos para ambientes de desenvolvimento distribuído de software”. Dissertação (Mestrado) – Ciência da Computação, Universidade Católica do Rio Grande do Sul (PUCRS).

Nemeth, Evi; Snyder, Garth; Hein, Trent R. (2007) “Manual completo do Linux guia do administrador”. 2ª ed. São Paulo: Pearson Prentice Hall.

Nunamaker, Jay F. (1999) “Collaborative Computing: The Next Millennium. Computer”, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=789753

Tommarello, Joana D; Deek, Fadi P. (2002) “Collaborative Software Development: A Discussion of Problem Solving Models and Groupware Technologies”. 35th Hawaii International Conference on System Sciences, Hawaii, p. 568 – 577, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=993937

Ferramentas Web 2.0 como suporte a comunicação em Desenvolvimento Distribuído de Software

Ivaldir H. de Farias Junior[1], Alinne C. Corrêa dos Santos[2], Catarina Costa[3], Sara Carvalho[1], Ryan R. Azevedo[4], Hemano P. de Moura[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 15.064 – 91.501-970 – Recife – PE – Brasil

[2]Instituto de Ciências Matemáticas e de Computação –
Universidade de São Paulo (USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brasil

[3]Centro de Ciências Exatas e Tecnológicas – Universidade Federal do Acre (UFAC)
Caixa Postal 500 – 69.920-900 – Rio Branco – AC – Brasil

[4]Departamento de Ciência da Computação – Universidade Federal Rural de Pernambuco - (UAG-UFPE) - Garanhuns – PE

ihfj@cin.ufpe.br,alinne@icmc.usp.br,catarina@ufac.br, {scrb2, rra2,Hermano}@cin.ufpe.br

Resumo. *A alta competitividade do mercado globalizado fez as empresas procurarem por soluções de desenvolvimento de software que garantissem uma maior competitividade, como o Desenvolvimento Distribuído de Software (DDS). Porém, obter comunicação de qualidade, fator crítico principal desse tipo de projeto, em função da distância espacial entre elas, torna-se uma tarefa difícil. Com o surgimento da Web 2.0 e suas ferramentas, foi notado que as características dessas ferramentas supriam a necessidade do auxílio à comunicação do DDS. Este artigo tem como objetivo efetuar um levantamento e analisar o uso das ferramentas da Web 2.0 com suporte à comunicação no DDS.*

1. Introdução

Projetos com equipes distribuídas têm se tornando comum no desenvolvimento de software. No início da década, Herbsleb e Moitra [13] afirmaram que a influência da globalização estava mudando a dinâmica dos negócios, e como resultado, o desenvolvimento de software estava cada vez mais distribuído, com empresas operando com sucesso além das fronteiras geográficas e culturais. Gomes e Marczak [14], reforçam essa afirmação quando falam que o processo de globalização que o mundo tem vivido por mais de uma década impôs efeitos colaterais em empresas de TI, com muitas organizações terceirizando seus processos de desenvolvimento de software ao redor do mundo.

Esse cenário de desenvolvimento de software, no qual a equipe encontra-se dispersa fisicamente e muitas vezes temporal e culturalmente, é conhecido no Brasil como Desenvolvimento Distribuído de Software (DDS). Um dos principais desafios nesse tipo de ambiente, relatados por alguns estudos [1],[2][7], diz respeito a

comunicação. Em uma revisão sistemática da literatura sobre o tema [1], comunicação efetiva foi citada por 34 estudos como um desafio em projetos distribuídos.

Ferramentas podem ajudar a reduzir os impactos da distribuição relacionados a redução na comunicação, principalmente quando considera-se a comunicação face a face o meio mais rico de comunicação[3]. As ferramentas Web 2.0, como blogs, wikis, entre outros, podem ser capazes de fornecer apoio para melhorar a comunicação. Porém, não se sabe ao certo quais dessas ferramentas são mais usadas e indicadas para esse tipo de ambiente.

Assim, o objetivo desse trabalho é realizar um levantamento e analisar o uso das ferramentas da Web 2.0 com suporte à comunicação no DDS. Para isso, foi realizada uma pesquisa com 60 participantes, sendo citadas 10 ferramentas da Web 2.0 como comumente usadas em projetos distribuídos. Além disso, 6 ferramentas comuns de comunicação foram citadas.

O trabalho está organizado da seguinte maneira: na seção 2 são apresentados os conceitos; na seção 3 são apresentandos os trabalhos relacionados, na seção 4 é apresentado o método; na seção 5 são relatados e discutidos os resultados do estudo; na seção 6 são apresentadas as considerações finais.

2. Web 2.0 no Contexto de Desenvolvimento Distribuído de Software

As ferramentas da Web 2.0 estão sendo amplamente úteis dentro das empresas, pois permitem agilidade, baixo custo (quando não são gratuitas) e não exigem treinamento formal, manutenção ou envolvimento do departamento de TI [4]. Essas ferramentas permitiram a melhora no diálogo empresa-cliente na colaboração e na comunicação tanto interna da empresa (entre seus membros), quanto externa.

No DDS a comunicação face a face é reduzida, assim como a comunicação utilizando ferramentas [5], [6]. Vale ressaltar que a qualidade desse último tipo de comunicação é menor, obrigando que os grupos se comuniquem ainda mais [7]. Caso contrário torna-se difícil saber o progresso do projeto e como contribuir para ele [8].

Portanto, tendo em vista a necessidade de comunicação e a qualidade das ferramentas da Web 2.0, é possível perceber que as vantagens de sua utilização são muitas no DDS. Com a utilização dessas ferramentas, o custo da comunicação é reduzido, em função de tais ferramentas serem ágeis, de fácil manipulação, flexíveis, permitirem a construção coletiva do conhecimento e aumentarem a comunicação entre os *stakeholders* do projeto [9], [10].

Apesar das vantagens proporcionadas da Web 2.0, é importante destacar duas desvantagens que possuem impacto no contexto do DDS: (i) falta de segurança dos dados [10] e (ii) a estrutura reduzida das ferramentas. A insegurança pode expor dados confidenciais da empresa e isto pode trazer um prejuízo de milhões [10]. A própria tecnologia AJAX, muito usada na Web 2.0, é muito vulnerável a ataques a segurança dos dados [9]. Esses ataques podem ocorrer de duas maneiras:

- Por meio de uma falha ou infiltração da própria ferramenta da Web 2.0;
- Por meio da entrada de vírus e instalação de *malwares* a partir do mau uso das ferramentas pelos funcionários das empresas [10]. Este uso incorreto pelos colaboradores pode ser inconsciente ou até mesmo consciente.

Quanto às estruturas reduzidas, ferramentas como *wikis* e *blogs* foram criadas com a finalidade de serem simples quando manipulados. No entanto, ao mesmo tempo em que muitos limites (ou estruturas) prejudicam a criatividade, a falta delas leva ao caos [9]. A liberação de ferramentas como as redes sociais, por exemplo, podem diminuir a produtividade, pois os colaboradores podem acabar se desviando dos assuntos profissionais para os pessoais [10].

3. Trabalhos Relacionados

Nesta seção serão descritas pesquisas diretamente relacionada ao estudo em questão com o intuito de apresentar a relevância do tema.

2.2.1 Turban et al (2010)[11]

O objetivo deste artigo foi a criação de um framework a partir do estudo de compatibilidade entre os softwares sociais (ferramentas da Web 2.0) e as atividades do processo de decisão em grupos virtuais e como estas ferramentas podem ser adotadas de modo bem sucedido. Para alcançar este objetivo, a metodologia adotada foi o estudo e a utilização do modelo Fit-Viability (Compatibilidade-Viabilidade) para estimar se uma ferramenta era compatível com uma determinada atividade de decisão e quais eram os fatores organizacionais para que estas ferramentas fossem efetivas.

2.2.2 Ben Al-Ani et al (2012)[12]

Este artigo teve como objetivo analisar a utilização de ferramentas tecnológicas na criação da confiança entre os grupos de projetos de DDS. A metodologia utilizada para alcançar esta meta foi a produção de um estudo empírico. Esse foi realizado através de entrevistas presenciais ou remotas com membros das organizações do Fortune 500[1].

Os resultados e conclusões, obtidos com as entrevistas, podem ser resumidos nos seguintes pontos (Tabela 1):

Tabela 1 – resultados das categorias relativas a web 2.0 e DDS

Categorias	Descrição
Comunicação e colaboração	Os entrevistados listaram tecnologias como e-mail, IM (instant messaging) e teleconferência, e raramente citaram ferramentas da Web 2.0.
Alinhamento	Os entrevistados afirmaram que essas ferramentas também não são alinhadas as práticas do trabalho.
Desconfiança	Não confiam nas informações disponíveis nessas ferramentas.
Cultura organizacional	Esta incompatibilidade com as ferramentas da Web 2.0 está relacionada a inadequação a cultura organizacional das empresas
Valor final	Todas essas desvantagens das ferramentas da Web 2.0 acabam por superar os benefícios que poderiam ser trazidos para o DDS.

É possível perceber que nenhum dos trabalhos relacionados foca nas pessoas (profissionais) que trabalham com as ferramentas da Web 2.0 no DDS. Outro ponto

[1] Fortune 500 é uma lista anual criada pela revista Fortune com o ranking das quinhentas organizações que possuem as maiores receitas brutas dos EUA.

interessante é que o foco nos fatores que influenciam a implantação dessas ferramentas no DDS é abordado timidamente pelos trabalhos. Por este motivo, o presente trabalho visa suprir essas questões.

3. Metodologia de Pesquisa

Para a condução deste trabalho foi realizada uma pesquisa de natureza qualitativa objetivando compreender o contexto atual da utilização das ferramentas da Web 2.0 no DDS. A pesquisa teve quatro fases foi definido. Na primeira foi realizada uma pesquisa bibliográfica sobre o DDS, a Web 2.0. Na segunda fase foi definido como método para coleta de dados o questionário. O questionário foi planejado com objetivo de elucidar e validar aspectos levantados na fundamentação teórica sobre os assuntos deste trabalho. Na terceira fase foi realizada a extração dos dados a partir dos dados coletados por meio da aplicação do questionário. Além disso, foram gerados gráficos para apresentação dos resultados da pesquisa. Por fim, na quarta fase foi realizada a análise dos dados extraídos do questionário, verificando quantidade de respostas válidas, sendo consolidadas 60 respostas válidas.

4. Resultados do Estudo Qualitativo

Esta seção tem como objetivo apresentar a análise dos resultados obtidos por meio das respostas dos 60 participantes, bem como os resultados da pesquisa em questão a qual elucida a teoria e a prática do uso das ferramentas Web 2.0 no contexto DDS.

4.1. Características dos Participantes

Nesta seção são ilustradas características dos participantes como idade, cargo, escolaridade, tempo de experiência em TI e DDS, bem como o tipo de dispersão que tais participantes se encontram.

Uma visão geral das idades dos participantes e sua escolaridade foi realizada. Foi possível perceber que a maioria dos participantes 38% (23/60) possuem idades entre 30 e 35 anos e 31% (19/60) são participantes com idades entre 25 e 30 anos. Por meio desses dados, é possível notar que as pessoas que trabalham com DDS são profissionais jovens. Quanto a escolaridade dos participantes foi possível observar que a maioria dos participantes 45% (27/60) possui mestrado. A graduação é o segundo nível de escolaridade mais obtido 28% (17/60) pelos participantes. Vale ressaltar que poucos participantes possuem doutorado 8% (4/60) e 2% (1/60) dos participantes possuem outros níveis de escolaridade como pós-doutorado, por exemplo.

Outras informações coletadas foram referentes à distribuição dos cargos entre os participantes e seus respectivos tempos de experiência em TI. É notável que cerca de 45% (27/60) dos participantes trabalham com desenvolvimento de software. Os participantes que compõem esse cargo são desenvolvedores, arquitetos de software, analistas de sistema, engenheiros de software, entre outros. A pesquisa é o segundo cargo mais ocupado com 25% (15/60) dos participantes, incluindo professores e pesquisadores. Dentre tais cargos, o que menos tem sido ocupado pelos participantes é o de gestão de TI com 5% (3/60), sendo formado por consultores e diretores de TI.

Em relação ao tempo de experiência (em anos) em TI de cada participante, foi observado visível que cerca de 42% (25/60) dos participantes possuem experiência entre 5 e 10 anos. O segundo índice apresenta 22% (13/60) dos participantes com menos de 5

anos de experiência. Dentre os anos de experiência, somente 5% (3/60) dos participantes possuem mais de 20 anos de experiência.

A Figura 1 ilustra o tempo de experiência (em anos) dos participantes em DDS. É notável que mais da metade dos participantes 53% (32/60) apresentam menos de 3 anos de experiência em DDS. Os participantes com experiência entre 3 e 6 anos representam 32% (20/60) do total. Os participantes com experiência entre 6 e 9 anos, 9 e 12 anos e mais de 12 anos ficaram com respectivamente 5% (3/60), 8% (4/60) e 2% (1/60) .

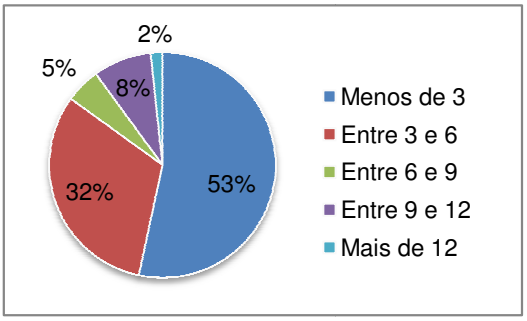


Figura 1. Tempo de Experiência em DDS dos participantes

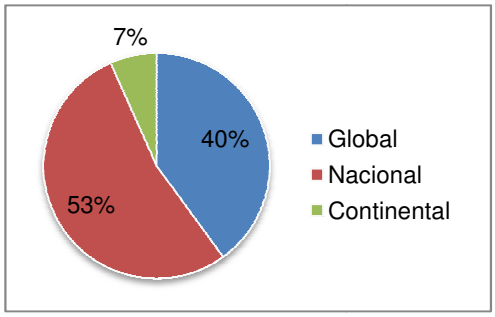


Figura 2. Tipo de Dispersão em DDS

A última característica coletada aborda o tipo de dispersão na qual o participante trabalhou em DDS (Figura 2). É importante destacar que mais da metade dos participantes 53% (32/60) trabalhou em projetos dispersos nacionalmente. Outros 40% (24/60) dos participantes trabalharam em projetos dispersos globalmente e somente 7% (4/60) dos participantes afirmaram que trabalharam em projetos com dispersões continentais.

4.2. Ferramentas utilizadas no DDS

Nesta seção são apresentas ferramentas utilizadas no DDS. Primeiramente, será destacado ferramentas que não são da Web 2.0 e, em seguida, serão detalhadas as ferramentas de Web 2.0.

As ferramentas que não são da Web 2.0 e estão sendo utilizadas na comunicação do DDS são apresentadas na Figura 3. É possível perceber que o e-mail é a ferramenta mais usada entre os tipos de ferramenta identificados, sendo mencionada por 57 participantes dos 60. A segunda ferramenta mais utilizada é o Skype, sendo citada por 45 participantes. Vale ressaltar que as ferramentas telefone, teleconferência e videoconferência são usadas por cerca da metade dos entrevistados.

As ferramentas da Web 2.0 utilizadas na comunicação em projeto de DDS (Figura 4). É observável que as wikis são as ferramentas mais cotadas com 63% (38/60). Em seguida são utilizadas as redes sociais com 26% (16/60), os blogs com 16% (10/60) e os serviços Web com 15% (9/60). No fim do ranking, são identificados o RSS com 5% (3/60), ferramentas de *podcast* com pouco mais de 1% (1/60) e as aplicações de inteligência coletiva e os mash-ups que não foram mencionadas.

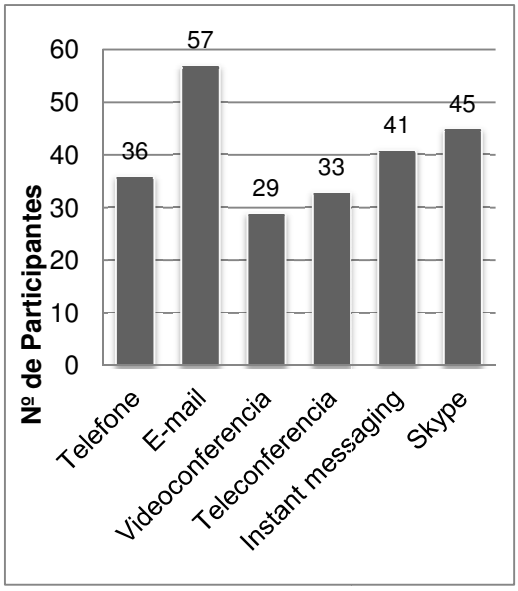


Figura 3. Ferramentas em geral utilizadas na comunicação e DDS

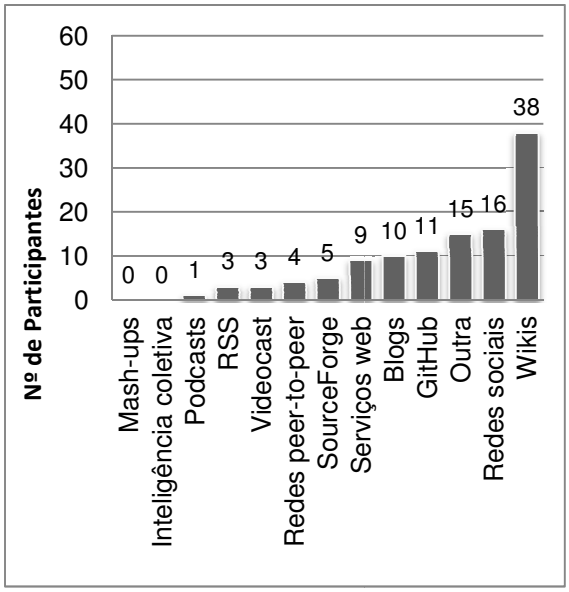


Figura 4. Ferramentas Web 2.0 utilizadas na comunicação em DDS

A partir dos dados obtidos, ilustrados na Figura 4, é possível concluir que também no cenário atual de DDS no Brasil, os *mash-ups* e os blogs são pouco utilizados, assim como concluído em uma pesquisa mundial feita pelo primeiro trabalho relacionado [7]. Com as ferramentas Web 2.0 identificadas, algumas características dessas ferramentas são ilustradas na Figura 5. A questão respondida foise tais ferramentas eram pagas ou *opensources*. 63% (38/60) dos participantes trabalharam em projetos DDS utilizando ferramentas *opensources* e 37% (22/60) afirmaram que as ferramentas utilizadas eram pagas.

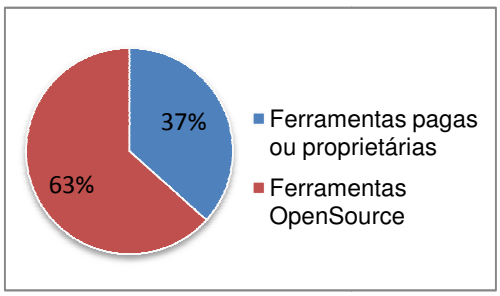


Figura 5. Características das Ferramentas Web 2.0

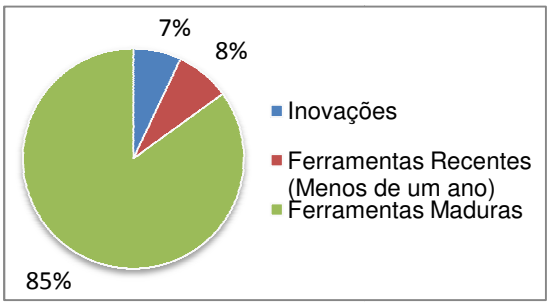


Figura 6. Tipo de Amadurecimento das Ferramentas Web 2.0

A Figura 6 aborda o amadurecimento das ferramentas utilizadas. É ilustrado que 85% (53/60) dos participantes que trabalharam ou trabalham com DDS, utilizam ferramentas maduras (com mais de 2 anos de mercado). Apenas 8% (4/60) disseram que as empresas utilizam ferramentas recentes (com menos de 1 ano de mercado) e 7% (3/60) disseram que utilizam inovações nos projetos de DDS. No mercado atual brasileiro, a maioria das ferramentas usadas são *opensource*, pois são ferramentas maduras o suficiente e reconhecidas no mercado.

A partir da identificação das ferramentas Web 2.0, alguns dados referentes à utilização dessas ferramentas no DDS foram coletados. A Figura 7 apresenta que 83%

(49/60) dos participantes afirmaram que esse tipo de ferramenta agrega valor aos times de DDS. Apenas 17% (11/60) dos participantes disseram que talvez essas ferramentas conseguissem agregar valor e nenhum dos participantes acredita que as ferramentas da Web 2.0 não agreguem valor às equipes em projetos de DDS.

A Figura 8 apresenta as sugestõesdos participantes pelas ferramentas que gostariam de utilizar em projetos de DDS, caso tivessem autonomia para escolhê-las. *Wikis* e redes sociais foram as mais bem cotadas e foram escolhidas por mais da metade dos participantes. O gerenciador de versões *Github*, as aplicações de inteligência coletiva e os serviços Web também foram citadas por um bom número de participantes. Enquanto que RSS, *mash-ups* e as redes *peer-to-peer* foram pouco mencionadas.

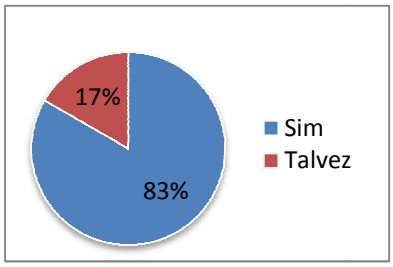


Figura 7. Valor agregado pelas ferramentas Web 2.0

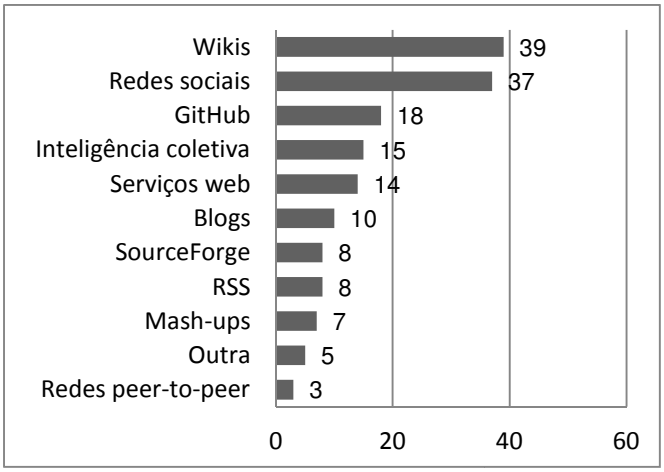


Figura 8. Autonomia na seleção das ferramentas Web 2.0

Outro enfoque foi quanto à implantação das ferramentas nas empresas. Foi levado em consideração fatores que influenciam na seleção da ferramenta, quem escolhe e se é realizado algum treinamento ou campanha de conscientização durante a implantação e quais fatores da implantação poderiam trazer resultados melhores sendo refeitos. A Figura 9 retrata os principais fatores que influenciam a implantação, sendo a integração e a usabilidade os mais citados.

É importante destacar que 70% (42/60) dos participantes teve/passou por algum tipo de treinamento ou campanha de conscientização sobre as ferramentas antes de começarem a trabalhar com as mesmas. Isto vai de encontro à opinião de alguns trabalhos que dizem que uma das vantagens das ferramentas da Web 2.0 é a não necessidade de treinamento pela sua usabilidade acessível.

A Figura 10 apresenta que 70% (42/60) dos participantes mencionaram que as ferramentas utilizadas na comunicação em projetos de DDS eram geralmente sugeridas pelos membros do time ligados diretamente ao projeto, o gerente do mesmo e os demais integrantes: desenvolvedores, arquitetos, entre outros.

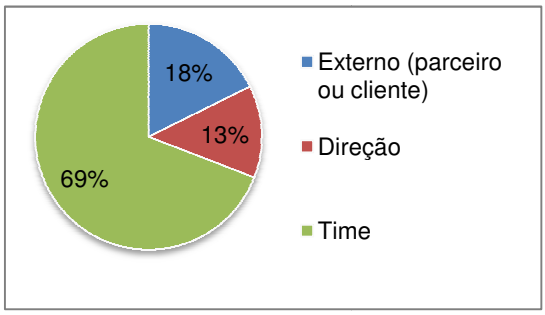


Figura 9. Fatores que influenciam a implantação das ferramentas

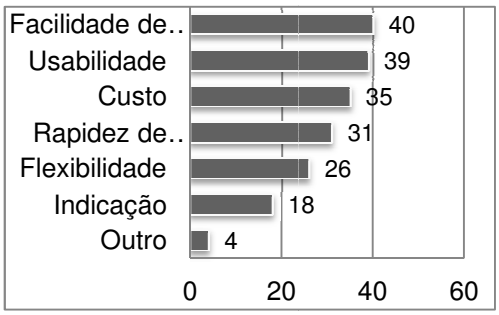


Figura 10. Sugestões da utilização de ferramentas Web 2.0

Na Figura 11 mais da metade dos entrevistados responderam que um treinamento mais eficiente melhoraria a implantação das ferramentas nas empresas. Uma melhor campanha de conscientização segue em segundo lugar, citada por 46% dos entrevistados. Isso mostra novamente que apesar da reconhecidavantagem no uso das ferramentas da Web 2.0 no DDS [25], muitos dos entrevistados acreditam ser muito importante o treinamento ou campanha de conscientização no processo implantação das mesmas nas empresas.

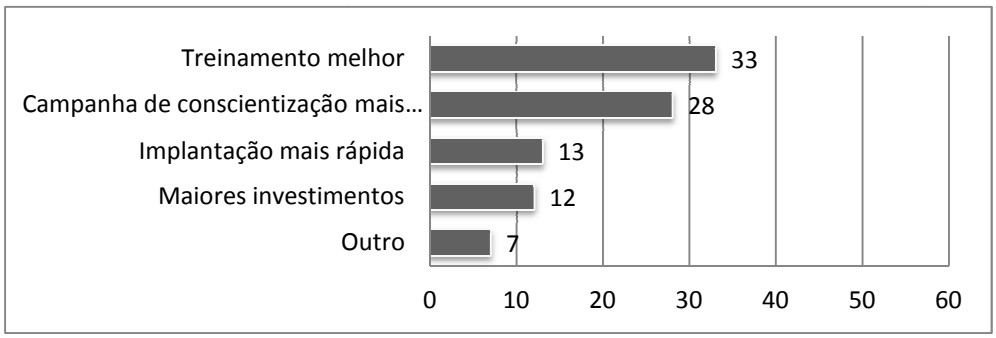


Figura 11. Aspectos da implantação que ao serem refeitos podem melhorar os resultados.

6. Considerações Finais

A presente pesquisa apresentou como principal contribuição um conjunto de ferramentas da Web 2.0 usadas para comunicação em projetos distribuídos, citadas por profissionais que atuam ou já atuaram nesse tipo de ambiente. Além disso, algumas ferramentas comuns foram reunidas e diversas caractérias sobre as ferramentas, como licença, fatores que influenciam na escolha, entre outros.

Através da pesquisa foi possível perceber que, apesar das qualidades das ferramentas da Web 2.0, elas foram timidamente implantadas nas empresas de DDS brasileiras conforme relatos da literatura disponível. Ferramentas que não são da Web 2.0 ainda possuem mais atuação no dia-dia dos profissionais de DDS, quando se trata de comunicação. Para mudar este cenário, a pesquisa mostrou que treinamentos e campanhas de conscientização mais eficientes podem ser aliados na implantação bem sucedida das ferramentas da Web 2.0 em projetos de DDS.

Como trabalhos futuros, propõe-se que seja realizada uma pesquisa qualitativa sobre a mesma perspectiva, utilizando etnografia (observação) para confrontar com a teoria e as pesquisas já executadas por outros pesquisados. Além disso, a observação de

diferentes tipos de projetos distribuídos, organizando as ferramentas pelas caractísticas desses projetos.

Referências

[1]DA SILVA, F. Q. B. *et al.* Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review. In: 2010 5TH IEEE INTERNATIONAL CONFERENCE ON GLOBAL SOFTWARE ENGINEERING (ICGSE), ago. 2010, [S.l: s.n.], ago. 2010. p. 87 –96.

[2]JIMÉNEZ, M.; PIATTINI, M.; VIZCAÍNO, A. Challenges and Improvements in Distributed Software Development: A Systematic Review. *Advances in Software Engineering*, v. 2009, p. 1–14, 2009. Acesso em: 9 set. 2012.

[3]PORTILLO-RODRÍGUEZ, J. *et al.* Tools used in Global Software Engineering: A systematic mapping review. *Information and Software Technology*, v. 54, n. 7, p. 663–685, jul. 2012. Acesso em: 30 maio 2013.

[4] ALBUQUERQUE, Rogério. A web 2.0 invade as empresas. , 2007. Disponível em: <http://info.abril.com.br/aberto/infonews/042007/07032007-23.shl>. Acesso em: 06 jun. 2012.

[5] SIQUEIRA, F. L., SILVA, P. S. M. As Características do Desenvolvimento Distribuído de Software. SBSI, 2004.

[6] ALLEN, T. J.; HENN, G. The Organization and Architecture of Innovation: managing the flow of technology. 1a edição. Elsevier: Butterworth-Heinemann, 2007. 136p.

[7] JUNIOR, I. H. de F., AZEVEDO, R. R. de, DANTAS, E. R. G., ROCHA, R. G. C., VERAS, W. C., FREITAS, F., GOMES, J. O. Proposta de Boas Práticas no Processo de Comunicação em Projetos Distribuídos. III Workshop de Desenvolvimento Distribuído de Software.

[8] DAMIAN, D., MARCZAK, S., KWAN I. Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks. Proc. of International Conference on Requirements Engineering (RE ’07), New Delhi, India, October, 2007, pp. 59-68.

[9] R, Revathi; DESAI, Pradeep. Effective collaboration and enabling innovations in globally distributed working (gdw) environment. In: International conference on management of globally distributed work, 2., 2007, Bangalore, India.

[10] WEB 2.0: Um equilíbrio complexo. , 2010. Disponível em: <http://www.mcafee.com/br/resources/reports/rp-first-global-study-web-2.0-usage.pdf>. Acesso em: 20 jun. 2012.

[11] TURBAN, Efraim; LIANG, Ting-peng; WU, Shelly P. J.. A framework for adopting collaboration 2.0 tools for virtual group decision making. Group decision and negotiation., p. 137. 15 out. 2010

[12] AL-ANI, B. et al. Distributed developers' perspectives of web 2.0 technologies in supporting the development of trust. In: The future of collaborative software development workshop, 2012, Nova Iorque, Estados Unidos

[13]HERBSLEB, J. D.; MOITRA, D. Global software development. *Software, IEEE*, v. 18, n. 2, p. 16 –20, abr. 2001.

[14]GOMES, V.; MARCZAK, S. Problems? We All Know We Have Them. Do We Have Solutions Too? A Literature Review on Problems and Their Solutions in Global Software Development. ago. 2012, [S.l.]: IEEE, ago. 2012. p. 154–158.

Desenvolvendo um Modelo de Maturidade para Comunicação em Desenvolvimento Distribuído de Software

Ivaldir H. de F. Junior[1], Sabrina Marczak[2], Hermano P. de Moura[1]

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

[2]FACIN – Pontifícia Universidade Católica de Porto Alegre (PUCRS)

{ihfj,hermano}@cin.ufpe.br, sabrina.marczak@pucrs.br

Abstract. *Communication is still one of the main challenges in distributed software development. A distributed software team needs to be able to communicate properly and in a timely manner to facilitate coordination. This paper presents a research under development that aims at developing a maturity model to support communication in distributed software development. The model will be inspired in quality maturity models such as the CMMI and the MPS.BR. We expect that the model will support project managers to improve communication in distributed projects.*

Resumo. *Comunicação ainda se encontra entre os principais desafios do desenvolvimento distribuído de software. É importante para uma equipe distribuída ser capaz de comunicar-se corretamente em tempo hábil visando facilitar a coordenação de atividades do projeto. Este trabalho apresenta uma pesquisa em desenvolvimento que visa conceber um modelo de maturidade para dar apoio à comunicação no desenvolvimento distribuído de software inspirado em modelos de qualidade tais como o CMMI e o MPS.BR. Espera-se que este modelo auxilie gerentes de projeto a melhorar continuamente a comunicação neste tipo de projeto.*

1. Introdução

Na última década a comunicação vem sendo apontada como o principal problema em desenvolvimento distribuído de software (DDS). Em um ambiente distribuído, a frequência da comunicação é baixa quando comparada com o desenvolvimento tradicional ou co-localizado [Farias Junior et al., 2012]. A distância geográfica torna-se um fator crítico para o sucesso do projeto, pois diminui fortemente encontros presencias [Farias Junior et al., 2010][da Silva, 2010] e reduz oportunidades da comunicação informal. Neste sentido, percebemos que a comunicação, por ser um ato social, está fortemente dependente da maturidade da equipe. Seja através da tecnologia ou não, precisamos de metodologias, padrões ou modelos que nos auxiliem a evoluir a maturidade da comunicação nos projetos distribuídos de software. Diante deste contexto, esse estudo propõe responder a seguinte questão de pesquisa: "*Como mensurar a maturidade da comunicação em projetos distribuídos de software?*". Como resultado, esta pesquisa vai propor um modelo de maturidade para auxiliar na melhoria dos processos de comunicação em tal tipo de projeto.

2. Comunicação no Desenvolvimento Distribuído de Software

Em DDS a comunicação assume um papel ainda mais importante em relação ao sucesso do projeto do que em relação a projetos co-localizados. Por exemplo, ela torna-se uma forma importante de compartilhar informações entre os membros da equipe e com os *stakeholders* envolvidos no projeto. Também é esperada uma comunicação frequente para o estabelecimento das definições do projeto no seu início quando os membros estão em geral se conhecimento e estabelecendo processos de trabalho e relações sociais para apoiar o desenvolvimento do software proposto [Perry, Staudenmayer & Votta, 1994]. Os desafios associados a comunicação também aumentam visto que os meios adotados como correio-eletrônico, *chats* e ligações telefônicas não são tão ricos quanto a comunicação face a face [Herbsleb & Moitra, 2001][Damian & Zowghi, 2002]. A comunicação em tais projetos segue dois formatos complementares: o formal e o informal. A partir do momento que a comunicação estiver mais efetivamente presente entre os membros e que as ferramentas colaborativas possam prover comunicação informal síncrona, haverá um aumento na percepção da equipe distribuída começando a criar importantes relações de confiança mútua na comunicação remota [Herbsleb & Moitra, 2001]. Não existe uma regra para gerenciar projetos distribuídos, mas existem boas práticas que são pontos relevantes e que ajudam os projetos a chegarem a seus objetivos. A comunicação segue o mesmo modelo: boas práticas podem auxiliar na sua maturidade e apropriado apoio ao desenvolvimento do projeto como um todo.

3. Trabalhos Relacionados

Os pesquisadores Soon e Park (2011) buscaram verificar empiricamente a possibilidade de gerenciar conflitos em equipes virtuais de forma indireta, utilizando a comunicação frequente como forma mediadora. Identificaram que o efeito mediador da frequência de comunicação ao conflito de tarefas é positivo para alguns conflitos mas negativo para outros. Eles também identificaram que a frequência de comunicação varia de acordo com o tipo de conflito, e que a frequência afeta o desempenho da equipe. Zhang, Min e Wu (2008) propuseram a criação de um modelo conceitual de gerenciamento de comunicação em equipes globais. Esse modelo considera as características da equipe como variáveis independentes, incluindo grau de virtualidade e diversidade. O estudo divide a comunicação em dois estilos positivamente relacionados: comunicação de tarefas e comunicação social, e visa ainda aumentar a compreensão do gerenciamento de comunicação neste tipo de equipe. O modelo foi criado baseado em literatura e apresenta fatores de sucesso para melhorar a eficiência da comunicação. O modelo não foi validado empiricamente.

4. Metodologia de Pesquisa

A pesquisa desenvolvida neste artigo faz parte do doutorado do primeiro autor. A natureza de pesquisa deste artigo classifica-se em aplicada, com uma estratégia qualitativa em retrospectiva do tipo exploratória e baseada no método de Yin (2008) de estudo de caso com a utilização de instrumentos de coleta de dados qualitativos. É uma pesquisa não experimental, desenvolvida no campo. A opção por utilizar o estudo de caso como referência foi realizada em função do estudo fazer questionamentos do tipo "como" e "por que" num contexto de desenvolvimento distribuído de software.

O objetivo geral desta pesquisa é propor um modelo de maturidade para comunicação em DDS com o intuito de auxiliar empresas inseridas nesse contexto a melhorarem seus processos de comunicação. Para atingir tal objetivo, o método de pesquisa esta organizado em duas grandes fases, conforme apresentado na Figura 1.

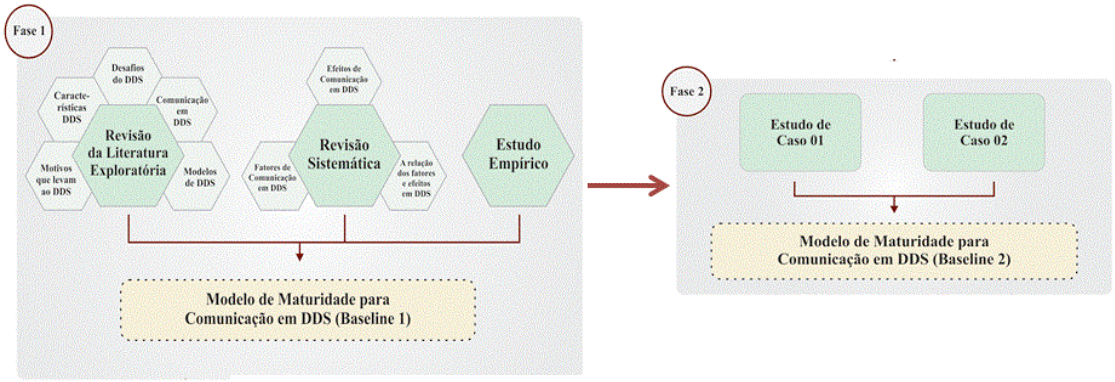


Figura 1 - Desenho de pesquisa

Fase 1: Esta fase tem como objetivo a definição do problema de pesquisa e do método a ser seguido bem como a coleta de dados iniciais. Desta forma, a fase é considerada exploratória e é organizada em três etapas. Na primeira etapa buscou-se estudar o referencial teórico, envolvendo inicialmente os conteúdos de DDS, comunicação e comunicação no contexto de DDS, além dos principais modelos de maturidade e modelos de negócio aplicados a esse contexto. Na segunda etapa planejou-se a execução de uma revisão sistemática da literatura de DDS com o objetivo de se aprofundar o entendimento sobre os fatores e efeitos de comunicação em DDS com o intuito de contribuir para o desenvolvimento do modelo de maturidade proposto. A revisão sistemática foi incluída visando ampliar a cobertura da revisão de literatura inicial. Uma versão resumida dos resultados foi relatada em (Santos, Farias Junior, Moura & Marczak, 2012). Na terceira etapa desenvolveu-se um estudo empírico com 12 profissionais de empresas que atuam em DDS. O objetivo deste estudo foi de identificar empiricamente problemas e fatores que impactam o sucesso da comunicação em DDS, além de soluções para minimizar tais problemas. Este artigo foca na concepção preliminar do modelo de maturidade baseada nos resultados destas três etapas. Um futuro segundo passo do estudo empírico visa avaliar as soluções propostas pelo reduzido grupo de profissionais entrevistados inicialmente com especialistas em DDS buscando identificar a completude destas soluções. Os especialistas serão consultados durante a realização da oitava edição do *International Conference on Global Software Engineering* (ICGSE), a ser realizado na Itália. Espera-se ter uma lista mais abrangente e exaustiva das soluções.

Fase 2: A segunda etapa foi planejada para múltiplos estudos em duas empresas visando uma investigação em maior profundidade. As empresas foram selecionadas baseadas na conveniência de acesso às mesmas e pela representativa atuação em DDS. Os estudos de casos têm como objetivo avaliar as soluções propostas e o escalonamento dos fatores em níveis, fornecendo mais subsídios para a concepção do modelo proposto.

4.1. Estudo Empírico

Seleção da Unidade de Análise e das Organizações: A unidade de análise do estudo foi definida como sendo profissionais com experiência em DDS. Por conveniência, foram selecionados profissionais de dez organizações. Algumas só aceitaram participar mediante a assinatura de um termo de confidencialidade.

Fontes dos Dados e Seleção dos Participantes: A coleta de dados foi constituída por fontes primárias através de entrevistas semiestruturadas. Foram realizadas 12 entrevistas individuais. Partiu-se de um roteiro básico, previamente validado através da avaliação de face e de pré-teste com questões formuladas aos entrevistados e adequadas conforme seu desenvolvimento. Dentre as questões, encontravam-se duas que listavam os fatores e soluções identificados na literatura e pedia que os respondentes indicassem quais eram os considerados relevantes na sua opinião pessoal e que adicionassem fatores e soluções que não haviam sido incluídas nas listas providas. O tempo total gasto com a coleta de dados foi de 15 horas e 12 minutos, tendo uma média de 1 hora e 26 minutos por participante. Logo no início do estudo, foi realizado o convite para cada profissional solicitando o seu aceite e a concordância da organização onde o mesmo trabalhava para participar da pesquisa. Os participantes da pesquisa estão distribuídos da seguinte forma: um profissional estava localizado no Canadá, um nos Estados Unidos, e dez estavam no Brasil nos seguintes estados: dois na Paraíba, cinco em Pernambuco, um no Rio Grande do Sul, e dois em São Paulo. Ainda, sete dos participantes são gerentes de projeto e cinco são líderes técnicos e desenvolvedores de software com média de 15 anos de experiência na área.

Análise de Dados: As entrevistas foram gravadas, transcritas e analisadas posteriormente. Primeiramente, os dados foram preparados; após as transcrições uma leitura cuidadosa foi realizada, de modo a buscar a familiarização do pesquisador com os dados antes de iniciar a codificação das categorias. Logo após, os textos foram codificados, seguindo duas etapas. A primeira etapa envolveu a classificação de trechos dos textos em uma das seguintes categorias: desafios de comunicação em DDS, boas práticas de comunicação em DDS e fatores que influenciam o processo de comunicação em DDS. Utilizou-se do Microsoft Excel e software de análise textual *Weft QDA* para apoi*o à tabu*lação e análise dos dados. A segunda envolveu a compreensão da análise.

5. Resultados Preliminares

O estudo realizado evidenciou diversos problemas e fatores que influenciam a comunicação em DDS bem como solução para as questões mencionadas. Esta seção apresenta os resultados preliminares do estudo.

5.1 Problemas Enfrentados e Fatores que Influenciam a Comunicação em DDS

De acordo com a análise dos dados coletados, pode-se dizer que os fatores ou problemas de comunicação em DDS estão centrados na ausência do entendimento das atividades, ausência de mecanismo (guias, processos, modelos) para o planejamento de comunicação nos projetos, falta de uma padronização das atividades entre as equipes distribuídas e a ausência de um processo bem definido refletindo nas atividades de engenharia de requisitos. Em seguida iremos apresentar alguns trechos das entrevistas

transcritas que permitem ilustrar estes resultados, como por exemplo, a citação de alguns dos respondentes da pesquisa:

Ausência do entendimento das atividades: Respondente 5 mencionou que "*As equipes por muitas vezes não tem o mesmo entendimento do todo que está sendo desenvolvido, o que causa bastante mal entendido na concepção do projeto*".

Falta de uma padronização das atividades entre as equipes distribuídas: Respondente 3 citou que "*Padronizar as atividades é uma boa prática para o DDS*", enquanto Respondente 4 disse que "*O mais difícil de se resolver é a padronização dos documentos, os documentos mínimos que possam orientar a especificação do produto*".

Ausência de um mecanismo (guias, processos, modelos) para o planejamento de comunicação nos projetos: Respondente 2 acredita que "*Um modelo de comunicação deve ter foco em minimizar os impactos da má interação entre as equipes*".

Além disso, também se verificaram fatores em relação a barreiras de idioma, diferenças culturais, dentre outros problemas indicados na Figura 2.

Áreas	Cód.	Fatores ou Problemas	Respondentes (R)
Recursos Humanos	F1	Ausência do entendimento das atividades;	[R1], [R2], [R4], [R5], [R6], [R9], [R10], [R11], [R12]
	F2	Idioma (jargões, gírias, verbetes, etc);	[R3], [R4], [R8], [R9], [R12]
	F3	Diferenças culturais;	[R3], [R4], [R8], [R9],[R12]
	F4	Ausência de comunicação frequente;	[R2], [R5], [R11], [R12]
	F5	Cordialidade entre as partes interessadas.	[R7], [R12]
Local e InfraEstrutura	F6	Ausência de reuniões face a face (presencial);	[R2], [R3], [R5], [R7]
	F7	Fuso horário/ sincronização de horários;	[R2], [R3], [R8], [R12]
	F8	Infraestrutura de comunicação;	[R1], [R2], [R3], [R7]
	F9	Gestão de conhecimento.	[R6], [R8], [R10]
Processos e Tecnologia	F10	Ausência de mecanismo (guias, processos, modelos) para o planejamento de comunicação nos projetos.	[R1], [R2], [R3], [R4], [R5], [R8], [R10],
	F11	Ausência de padronização das atividades.	[R1], [R2], [R4], [R5], [R6], [R7]
	F12	Elicitação e mudanças de requisitos.	[R2], [R3], [R4], [R8], [R9], [R11]
	F13	Ausência de padronização dos documentos.	[R1], [R2], [R4], [R5], [R6]

Figura 2 – Fatores/Problemas de comunicação

5.2. Soluções Encontradas para os Problemas de Comunicação em DDS

Encontra-se na literatura (*e.g.*, [Carmel & Agarwal, 2001][da Silva et al., 2010]), soluções para problemas de comunicação que são voltados mais especificamente para casos específicos, sem uma formalização ou validação da eficiência dessas soluções sugeridas. Nessa pesquisa, buscou-se identificar soluções complementares as já existentes na literatura que são utilizadas pelas empresas para minimizar os fatores apresentados na Figura 2. As soluções destacadas pelos respondentes deste estudo se concentraram principalmente na necessidade de definir ferramentas para comunicação síncrona, reuniões presencias, reuniões diárias, adoção de ferramentas de colaboração e

padrões de trabalho, conforme apresentado na Figura 3. Em seguida explicitamos a relação entre os fatores a as soluções que foram extraídas das entrevistas. Essa relação tem como objetivo identificar qual solução pode ser aplicado para minimizar (ou mesmo resolver em definitivo) o impacto do fator ou problema.

Soluções implementadas pelos profissionais (respondentes)	Fatores (F)	Respondentes (R)
Criar pontos focais ou interlocutores	F1, F2	R1, R2, R4
Reuniões diárias	F1, F2, F4	R1, R5, R8, R11
Adoção de ferramentas síncronas	F4, F5, F8	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12
Programas de capacitação	F12, F9, F10, F11, F13	R3, R4
Intercambio dos membros entre as equipes dispersas	F2, F3, F5	R3, R4, R5, R8
Adoção de ferramentas de colaboração	F4, F7, F8, F9	R2, R3, R5, R6, R11
Realização de workshops técnicos sobre tecnologias utilizadas no projeto	F2, F3, F9, F10, F13	R4
Criar mecanismos para confirmar o entendimento das atividades	F1, F11, F13	R2, R8
Reuniões presenciais periodicamente	F1, F3, F4, F5, F6, F7, F12	R5, R7, R8, R10, R12
Institucionalizar o contexto cultural de cada equipe pertencente ao projeto	F3, F5, F12	R9
Utilizar boas práticas para planejar a comunicação do projeto	F1, F2, F3, F6, F7, F8, F9, F10, F11 F12, F13	R2, R10
Definir e institucionalizar o vocabulário do projeto para as equipes	F1, F2, F3, F5, F9, F11, F12, F13	R10
Padronizar das atividades e *reports* para a distribuição de informações aos interessados	F1, F2, F3, F5, F9, F10, F11, F12, F13	R4, R6, R10, R11
Manter um ambiente de trabalho cordial e harmonioso	F3, F5	R3, R7, R8, R9
Discutir a melhoria da comunicação no SEPG	F10, F11, F13	R4

Figura 3 - Relação das soluções com os fatores identificados

6. Modelo Preliminar Proposto: C2M

De acordo com o CMMI (2006), "a qualidade de um sistema é amplamente influenciada pela qualidade do processo utilizado", além de fornecer uma base para aumentar a produtividade das pessoas e potencializar o uso da tecnologia para tornar a organização mais competitiva. A intensa busca por modelos de qualidade está diretamente ligada à demanda organizacional, uma vez que a efetiva gestão dos ativos organizacionais é fator crítico para o sucesso do negócio. Nesse contexto, os processos advindos de modelos de maturidade têm por objetivo ajudar às organizações a alcançarem os resultados esperados através da eficiente execução das atividades planejadas, além de minimizar os impactos quando da introdução e uso de novas tecnologias. Neste sentido, o modelo de maturidade de comunicação desta pesquisa pretende tratar a melhoria dos processos de comunicação através de melhores práticas relativas às atividades desenvolvidas ao longo do ciclo de vida do projeto.

Diante do exposto, entende-se que o modelo deve ser organizado da seguinte maneira: as áreas de conhecimento representam a categorização dos tipos de problemas e fatores que influenciam a comunicação em grandes grupos. Utilizou-se o termo "área de conhecimento" para esta classificação. Para cada área de conhecimento existem fatores (pode representar problemas a serem resolvidos) de maturidade, que por sua vez possuem objetivos. Cada fator de maturidade possui apenas um objetivo, que representa a sua descrição em si. Para cada fator de maturidade, uma ou mais práticas devem ser implementadas conforme a estrutura apresentada na Figura 5. O cumprimento das práticas previstas determina o nível de maturidade de cada fator. Além disso, o

cumprimento de determinadas práticas previstas em um conjunto de fatores de maturidade de uma determinada área de conhecimento determina o nível de maturidade daquela área. Como exemplo, ao implementar todas as práticas dos fatores da área de maturidade Y de um nível hipotético X, uma empresa pode ter seus projetos avaliados como nível X na área Y. Sendo assim, o modelo C2M possui uma estrutura com três dimensões: áreas ou categorias de conhecimento, fatores e níveis de maturidade.

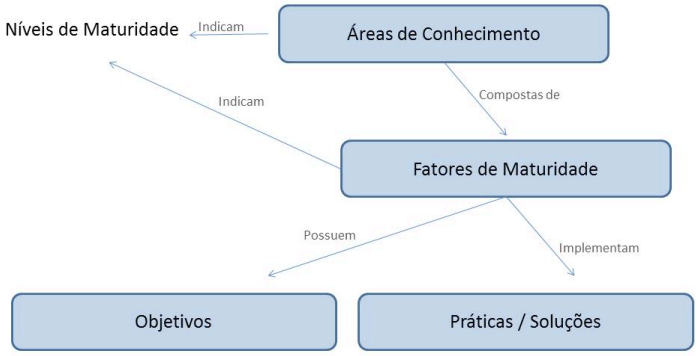


Figura 5 – Estrutura preliminar do modelo

Para identificar as áreas ou categorias de conhecimento, bem como os fatores de maturidade , foram utilizadas as informações da revisão de literatura (RSL) e do estudo empírico apresentado na Seção 5. As áreas ou categorias de conhecimento foram inicialmente classificadas a partir da revisão sistemática da literatura. Com a coleta de dados realizada no estudo empírico, complementou-se a classificação a partir da opinião de especialistas na área. Sendo assim, três categorias dos fatores foram identificadas: fatores humanos, local e infraestrutura e processos e tecnologia. Já os fatores de maturidade foram coletados a partir da RSL e corroborados pelo estudo empírico (vide Figura 2). Para cada fator de maturidade ainda foi definido um objetivo a ser satisfeito. Atualmente estamos coletando dados para validar as soluções práticas que devem ser implementadas para eliminar os problemas de comunicação em DDS conforme mencionado na Seção 4 (passo 2 do Estudo Empírico).

7. Conclusão

O desenvolvimento de software sempre se apresentou como uma atividade complexa. Portanto, ainda existem diversos problemas e desafios inerentes ao processo a serem resolvidos [Prikladnicki, Marczak, & Audy 2006]. O modelo proposto é uma contribuição que visa obter resposta referente a uma categoria de problemas ainda um tanto negligenciada pelas organizações e pelos gerentes de projetos no ambiente de trabalho distribuído. Os resultados encontrados nessa pesquisa mostram que a área de Engenharia de Software necessita de mais pesquisas voltadas para esta categoria de problemas que o DDS potencializa, a comunicação. Do ponto de vista científico, a qualidade do modelo de maturidade proposto é decorrente do rigoroso processo de pesquisa seguido pela pesquisa. Sendo assim, a principal contribuição é a proposta de um modelo de maturidade para comunicação em projetos de desenvolvimento distribuído de software que visa responder a questão de pesquisa descrita na Seção 1. É

através da definição das práticas específicas do modelo (que ainda estão em desenvolvimento) que iremos elevar a maturidade do projeto DDS. Em seguida, cada área de processo passará por uma avaliação juntamente com suas práticas específicas para a definição do nível de maturidade do projeto. Esse modelo soma-se aos existentes na literatura como mais uma contribuição para responder aos desafios da área. Por exemplo, o modelo teórico proposto por Zhang, Min e Wu (2008), apresentado na Seção 2, não havia sido validado. Os problemas citados empiricamente em nosso estudo estendem consideravelmente as categorias indicadas neste modelo, o que reforça a importância da consulta a profissionais que vivenciam situações reais corriqueiramente.

Uma das principais limitações da pesquisa refere-se ao número de respondentes entrevistados na parte empírica do estudo, restringindo a generalização dos resultados obtidos. Como trabalhos futuros pretende-se ampliar a pesquisa com mais respondentes, executar alguns estudos de casos com o objetivo de evoluir o modelo para conceber a versão final do mesmo e em seguida avaliá-lo com profissionais de DDS.

Referências

Carmel, E. and Agarwal, R. (2001), Tactical approaches for alleviating distance in global software development. IEEE Software, vol. 18, no. 2, p. 22–29.

CMMI Product Team (2006), CMMI for development, version 1.2. Technical report, Software Engineering Institute.

Damian, D. and Zowghi, D. (2002), The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. IEEE Joint Int. Conf. on Requirements Engineering, p. 319–328, Washington, DC, USA.

Farias Junior, I. et al. (2010), Best Practices to Enhance the Communication in Distributed Software Development Environments. Int.Conf. on Research and Practical Issues of interprise Information Systems, RN, Brazil.

Farias Junior, I. et al. (2012), Elicitation of Communication Inherent Risks in Distributed Software Development. 6th Workshop on Distributed Software Development, Porto Alegre Brazil.

Herbsleb, J. and Moitra, D. (2001). Global software development. IEEE Software, March/April, p. 16-20.

Perry, D.et al., (1994), People, organizations, and process improvement. IEEE Software, vol. 11, no. 4, p. 36–45.

Prikladnicki, R.; Marczak, S. and Audy, J. (2006), MuNDDoS: A Research Group on Global Software Development. In: ICGSE, v. 1. Florianópolis, Brazil, p. 251-252.

Santos, A., Farias Junior, I., Moura, H. and Marczak, S. (2012), "A Systematic Tertiary Study of Communication in Distributed Software Development Projects", In: 7th ICGSE, Porto Alegre, Brazil, p. 182.

Silva, F. et al. (2010), "Challenges and Solutions in Distributed Software Development Project Management: A Systematic Literature Review", In: 5th ICGSE, Washington, USA, p. 87-96.

Yin, R. (2008), Case study research: Design and methods. Thousand Oaks, CA. Sage.

Zhang, Y. et al. (2008), GVTs communication management: A conceptual model. Service Operations and Logistics, and Informatics, IEEE International Conference on V. 1. Beijing, China, p. 583–587.

Desafios no Gerenciamento de Conflitos em Projetos de Desenvolvimento Distribuído de Software

João Paulo N. de Oliveira, Ariádnes N. Dantas, Ivaldir H. de Farias Junior, Jefferson F. Barbosa, Dennis Savio, Hermano P. de Moura

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Caixa Postal 15.064 – 91.501-970 – Recife – PE – Brasil
{jpno, andr, ihfj, jfb2,hermano}@cin.ufpe.br

Abstract. *In the last decades it was observed that major companies around the world started to use the strategy of Distributed Software Development (DSD) as a way of building competitive advantages in the global market. However, managing DSD teams reveals some particularities that should be taken into account. This work in development presents the preliminary results of a systematic literature review of Distributed Software Development (DSD) Conflicts Management. The objective is to collect and systematize reported knowledge in terms of what are the difficulties in managing the conflicts in DSD projects. We found 72 papers between 1998 and 2012. Using the data were systematically extracted from these works, we proposed a set of 11 categorizations as the mainly challenges that impacts the conflicts management in DSD projects. Our desire is that this categorization can support practitioners and researchers to better understand the landscape of DSD projects challenges so that they can be aware of the mainly difficulties they will face during conflict management activities in a distributed setting.*

Resumo. *Nas últimas décadas, foi possível observar que grandes empresas ao redor do mundo passaram a utilizar a estratégia de Desenvolvimento Distribuído de Software (DDS) como uma forma de construção de diferenciais competitivos no mercado global. No entanto, o gerenciamento das equipes de DDS revela algumas particularidades que devem ser levadas em consideração. Este trabalho em desenvolvimento apresenta resultados preliminares de uma revisão sistemática da literatura sobre gerenciamento de conflitos em equipes de desenvolvimento distribuído de software. O objetivo é coletar e reportar o conhecimento adquirido sobre as principais dificuldades encontradas no gerenciamento de conflitos em projetos de DDS. Nós encontramos um total de 72 trabalhos entre os anos de 1998 e 2012. Usando os dados que foram sistematicamente extraídos desses trabalhos, nós propusemos uma categorização dos principais desafios encontrados em 11 fatores que impactam o gerenciamento de conflitos em projetos de DDS. Nosso desejo é que esta categorização possa apoiar profissionais e pesquisadores no melhor entendimento dos desafios que permeiam os projetos de DDS para que eles possam estar cientes das principais dificuldades que irão enfrentar durante as atividades de gerenciamento de conflitos em um ambiente distribuído.*

1. Introdução

A adoção de DDS vem mostrando ser não mais que uma tendência das grandes empresas e sim uma realidade para todas as empresas, independente de seu porte. Essa nova forma de se fazer software ou gerenciar projetos vem trazendo várias vantagens, inclusive um dos resultados é a baixa dos custos e o aumento da produtividade. Entretanto, o DDS também apresenta uma série de desafios como efeito colateral. Em um projeto distribuído, a comunicação entre as partes interessadas é comprometida, e a separação existente aumenta a complexidade envolvida nas atividades de liderança e coordenação de projetos, gerando conflitos organizacionais que podem afetar o alcance dos objetivos do projeto.

Pesquisas realizadas no contexto de equipes virtuais enfatizam que o gerenciamento dos conflitos, principalmente em um ambiente distribuído é extremamente importante, uma vez que estudos já demonstram que o ato de gerenciar conflitos está associado aos processos e fatores de desempenho dos membros das equipes de projetos e quando gerenciados de forma efetiva podem aumentar a sinergia entre os membros das equipes, aumentando assim as chances de sucesso do projeto (VAN DE VLIERT; DE DREU, 1994).

Dada a importância do gerenciamento efetivo dos conflitos em um ambiente distribuído, foi realizada uma revisão sistemática da literatura com o objetivo de identificar quais os principais desafios enfrentados no gerenciamento de conflitos no contexto dos projetos de desenvolvimento distribuído de software que podem impactar diretamente o sucesso dos projetos. Esta revisão sistemática teve como objetivo responder a seguinte questão de pesquisa: *"Quais são os principais desafios no gerenciamento de conflitos em equipes de desenvolvimento distribuído de software?"*

2. Conflitos no Ambiente de Desenvolvimento Distribuídos de Software

O Desenvolvimento Distribuído de Software pode ser definido como a atividade de desenvolvimento de software realizada por uma equipe distribuída geograficamente. Dentre as características dos projetos DDS, Prikladinick (2003) apresenta a colaboração e cooperação entre as partes da equipe que estão distribuídas como sendo a principal delas.

O termo conflito é originado da palavra latina conflictus, que significa discordância ou choque. Dessa forma ele representa um choque entre aspectos divergentes, tais como perspectivas, interesses, objetivos ou comportamentos (MELE, 2011). Amason (1996) afirma que quando um conflito é ignorado, isto pode desencadear um processo de redução da comunicação efetiva, diminuindo as interações que são realizadas de forma eficiente e por fim prejudicando o desempenho e coesão da equipe. Em um ambiente de projetos é relevante que todos os membros estejam alinhados na busca por um mesmo objetivo, logo o gerenciamento de conflitos é de extrema importância.

No ambiente distribuído, Hinds e Mortensen (2001, 2005) já apontam que o gerenciamento de conflitos é uma tarefa mais difícil, devido à própria natureza e desafios da virtualização das equipes, fazendo com que o acompanhamento dos mesmos seja mais difícil, o que ocorre muitas vezes decorrente da falta de uma ação imediata, devido a problemas como diferenças de fuso horários entre as equipes, ou até mesmo devido a própria ausência de contato face a face (NAUMAN; IQBAL, 2005), o que pode limitar a

habilidade de comunicação, potencializando os conflitos e gerando prejuízos para o projeto.

Diante desse cenário é observado que o gerenciamento dos conflitos, principalmente em um ambiente distribuído é extremamente importante, porquanto estudos demonstram que o ato de gerenciar conflitos está associado aos processos e fatores de desempenho dos membros das equipes de projetos e quando gerenciados de forma efetiva podem aumentar a sinergia entre os membros das equipes, aumentando assim as chances de sucesso do projeto (VAN DE VLIERT; DE DREU, 1994).

3. Metodologia

Esta pesquisa opta por um método de abordagem indutivo baseado em dados de natureza qualitativa com o apoio dos métodos de procedimento meta-etnografia e revisão sistemática da literatura. A primeira etapa para realização de uma revisão sistemática é a definição do protocolo de pesquisa, o qual descreve em detalhes as etapas a serem seguidas. Estas etapas estão descritas de forma sumarizada nas subseções 3.1, 3.2, 3.3, 3.4 e 3.5 e foram elaboradas de acordo com guia proposto por Kitchenham e Charters (2007), com exceção da avaliação de qualidade dos trabalhos que não fora realizado nestes resultados preliminares.

3.1. Questão de Pesquisa

Esta revisão sistemática da literatura tem como objetivo responder a seguinte questão?

- Quais são os principais desafios no gerenciamento de conflitos em equipes de desenvolvimento distribuído de software?

3.2. Estratégia de Busca

Nossa estratégia de busca foi dividada em duas etapas, busca automática e manual. Para a busca automática foi elaborada uma string de busca a partir dos conceitos chaves de nossa questão de pesquisa (Tabela 1):

Tabela 1 – Conceitos chaves utilizados na elaboração da string de busca

Gerenciamento de conflitos	("Conflict* Management" OR "Manag* conflict*" OR "Manag* of conflict*" OR "Conflict* Resolution" OR "Dispute* Management" OR "Dispute* Resolution" OR "Res* Dispute*" OR "Res* of Dispute*")
Desafios	("Challenge*" OR "Difficult*" OR "Problem*" OR "Critical Factor*")
Desenvolvimento Distribuído de Software	("Distributed software development" OR "Global software development" OR "Collaborative software development" OR "Dispersed software development" OR "Virtual software development" OR "Dispersed development" OR "Global software engineering" OR "Dispersed software engineering" OR "Globally distributed work" OR "Collaborative software engineering" OR "Virtual software engineering" OR "Distributed development" OR "Virtual development" OR "Dispersed teams" OR "Distributed teams" OR "Virtual teams" OR "Global software teams" OR "Globally distributed development" OR "Geographically distributed software development" OR "Geographically dispersed development teams" OR "Offshore software development" OR "Offshoring" OR "Offshore" OR "Offshore outsourcing")

A busca manual foi feita a partir de análises nos sites dos *journals* para cada um dos anos de publicação, volumes e edições; além de busca em anais de eventos importantes sobre o tema de pesquisa, complementando com uma busca por referências indicadas nos estudos já incluídos na pesquisa, chamada bola de neve.

3.3. Fontes de Busca

Para a execução das buscas automáticas e manuais foram utilizadas as seguintes fontes de pesquisa (Tabela 2):

Tabela 2 – Fontes de busca utilizadas na pesquisa

Busca Automática	ACM Digital Library, El Compedex, Elsevier ScienceDirect, IEEEXplore Digital Library, SCOPUS e WEB of Knowledge
Busca Manual	Information and Software Technology (IST), IEEE Transactions on Software Engineering (TSE), Software Practice and Experience (SPE), ACM Transactions on Software Engineering Methodology (TOSEM), Journal of System and Software (JSS), International Symposium on Empirical Software Engineering and Measurement (ESEM), Evaluation and Assessment of Software Engineering (EASE), International Conference on Software Engineering (ICSE) e International Conference on Global Software Engineering (ICGSE)

3.3. Critérios de seleção dos estudos

Para a realização da Revisão Sistemática, foram pesquisados na literatura existente os estudos considerados relevantes à pesquisa. Com o intuito de aprimorar a seleção desses estudos alguns critérios de exclusão e inclusão foram definidos e serão apresentados a seguir (Tabela 3):

Tabela 3 – Critérios de seleção dos estudos

Critérios de Inclusão	(1) Artigos completos publicados em revistas ou conferências revisadas, que relatem o gerenciamento de conflitos no desenvolvimento distribuído de software e respondam a, pelo menos, uma das questões de pesquisa específicas; (2) Estudos experimentais ou Empirical Studies; (3) Estudos Secundários, aqueles que dependem de estudos primários; (4) Estudos Teóricos, que apresentem conceitos baseados em um entendimento de uma área, referenciando outros trabalhos.
Critérios de Exclusão	(1) Estudos não escritos em inglês, (2) Artigos convidados, tutoriais, key-note speech, relatórios de workshop, teses, dissertações, relatórios técnicos e livros; (3) Artigos que expressam apenas pontos de vistas pessoais, opiniões de especialistas ou relatos de experiências; (4) Documentos que não sejam artigos completos; (5) Artigos duplicados, que já foram encontrados em outras fontes; (6) Estudos relacionados a Ciência da Computação que não sejam claramente da área de Engenharia de Software ou Sistemas de Informação; (7) Estudos que não respondem a nenhuma das questões da pesquisa; e (8) Trabalhos que não tiveram seus arquivos encontrados/não recuperados.

3.4. Processo de seleção dos estudos relevantes

Após a execução da consulta, os estudos resultantes foram selecionados independentemente por 05 pesquisadores de acordo com o procedimento na tabela 4. A realização do processo de revisão sistemática da literatura através da composição de uma equipe é de fundamental importância para aumentar a confiabilidade dos resultados da pesquisa, uma vez que com uma equipe é possível associar os pesquisadores aos pares, diminuindo desta maneira o viés no processo, caso apenas um pesquisador fosse o agente individual do processo (KITCHENHAM, B.; CHARTERS, S., 2007).

Tabela 4 – Etapas da seleção dos estudos relevantes (Cont.)

Passo 1	Com os resultados da busca automática e manual foi alcançada a quantidade inicial de trabalhos, denominada ***Estudos Encontrados***, os quais servirão como entrada para as etapas seguintes de seleção dos estudos.
Passo 2	Após a determinação dos ***Estudos Encontrados***, foi feita uma pré-seleção dos artigos, onde determinamos os ***Estudos Potencialmente Relevantes*** após a leitura de título e resumo de cada trabalho.

Passo 3	Com os ***Estudos Potencialmente Relevantes*** selecionados, é chegada a etapa de seleção dos estudos, onde utilizamos os critérios de seleção dos trabalhos. Para isso se fez necessária a leitura da introdução, metodologia e conclusão dos ***Estudos Potencialmente Relevantes***, aplicando os critérios estabelecidos, com o objetivo de ter como resultado final apenas os denominados ***Estudos Relevantes***, os quais serão utilizados na pesquisa para a etapa de extração dos dados.
Passo 4	Para finalizar, foi realizada uma busca manual complementar nas referências indicadas nos ***Estudos Relevantes*** já incluídos na pesquisa, chamada bola de neve (do inglês, snowball) de forma a aumentar a cobertura do processo e definirmos a quantidade final dos ***Estudos Relevantes*** para a pesquisa.

As 4 etapas da pesquisa tiveram uma duração total de 12 meses (maio/2012 à abril/2013). Para apoiar todas as etapas deste processo foram utilizadas algumas ferramentas, tais como o Mendeley (www.mendeley.com), Dropbox (www.dropbox.com) e ZOHO Projects (https://projects.zoho.com).

3.5. Extração e síntese dos dados

Após a definição dos estudos relevantes para a pesquisa, cada trabalho foi lido e analisado em sua totalidade por 4 pesquisadores e os trechos considerados mais importantes para responder a questão de pesquisa foram extraídos e documentados em uma planilha Excel™ de forma que fossem organizados e sintetizados para a apresentação dos resultados finais da pesquisa.

A síntese dos dados foi realizada a partir da extração dos dados através do método de procedimento de meta-etnografia, como proposto por Noblit e Hare (1988). De forma sucinta este procedimento consiste na identificação de conceitos chaves nos textos selecionados, que fornecem informações relevantes para a pesquisa, categorizando-os em formas de tabelas de acordo com as questões de pesquisa. A esses trechos, são associados códigos que indicam a que tipo ou categoria de informação o texto se refere, de forma que as informações possam ser interpretadas.

4. Análise dos Resultados

Esta seção está dividida em Análise descritiva da revisão sistemática, a qual mostra os resultados gerais da revisão sistemática da literatura; e Análise das evidências, a qual foca na análise dos dados para a resposta à pergunta que norteia a pesquisa.

4.1. Análise Descritiva da Revisão Sistemática

A revisão sistemática foi executada de acordo com o que foi definido de forma simples no protocolo da seção anterior. As buscas iniciais (Passo 1) retornaram um total de 2.784 Estudos Encontrados, sendo 2.670 trabalhos retornados da busca automática (pesquisa nos engenhos de busca relevantes da área) e 114 retornados da busca manual (pesquisa nos *journals* e anais de conferências relevantes da área) ainda sem os estudos provenientes do processo de busca nas refências dos estudos relevantes (*snowball)* (Passo4). Na etapa seguinte, também chamada de Pré-Seleção (Passo 2), foi realizada a análise dos estudos encontrados através da leitura do título e resumo de cada um dos estudos encontrados, resultando em 570 trabalhos selecionados, denominado Estudos Potencialmente Relevantes. Em seguida, com a leitura da introdução, metodologogia e conclusão, além da utilização dos critérios de inclusão/exclusão (Passo 3), 506 estudos foram excluídos, chegando ao total de 64 estudos relevantes, sendo 53 provenientes da busca automática e 11 da busca manual, como podem ser verificados na Tabela 5.

Tabela 5 – Resultados das Etapas de Busca Automática e Manual

Busca Automática	Etapa 1	Etapa 2	Etapa 3
ACM Digital Library	377	205	18
El Compedex	7	5	1
ScienceDirect	1021	45	1
IEEEXplore	365	206	27
SCOPUS	892	102	5
WEB of Knowledge	8	7	1
Total Consolidado	**2670**	**570**	**53**
Busca Manual	**Etapa 1 e 2**		**Etapa 3**
EASE	8		1
IST	22		3
ICGSE	26		2
ICSE	36		1
ESEM	1		0
JSEP	10		3
JSS	7		0
SPE	3		0
TSE	1		1
TOSEM	0		0
Total Consolidado	**114**		**11**

De forma a aumentar a cobertura foi realizada uma busca manual complementar nas referências indicadas dos 64 estudos relevantes já incluídos na pesquisa, chamada bola de neve (Passo 4), o que gerou uma seleção de mais 8 trabalhos, perfazendo um total de 72 estudos relevantes selecionados para a etapa de extração e síntese dos dados.

4.2. Análise das Evidências

Nesta subseção serão apresentados os resultados relacionados com a resposta para a pergunta da pesquisa. Nossa questão de pesquisa tem como objetivo identificar os principais desafios no gerenciamento de conflitos em equipes distribuídas. A identificação dos desafios foi realizada a partir da análise dos estudos relevantes que foram selecionados para a pesquisa. Apesar de haver casos de uma relação entre os desafios encontrados e uma contribuição mista para o surgimento dos conflitos, foram categorizados um total de 11 desafios, como sumarizados na Tabela 6.

Tabela 6 – Desafios no gerenciamento de conflitos de projetos de DDS (Cont.)

Desafios no Gerenciamento de Conflitos em Projetos de DDS	Descrição	Referências – ER: Estudos Relevantes	#
D01: Perfil da coordenação	Um perfil inadequado do gerente de projetos para coordenar as equipes distribuídas	ER01,ER10,ER11,	03
D02: Agendamento de reuniões	Impossibilidade de conversa da equipe devido à demandas conflitantes entre as equipes distribuídas	ER02	01
D03: Problemas de comunicação	Dificuldades de comunicação devido a ausência de comunicação síncrona, presencial (face a face) e utilização de meios de comunicação inadequados para resolver determinados tipos de questões	ER02,ER10,ER13,ER15, ER20,ER68,ER33,ER69, ER36, ER39,ER46,ER70, ER57,ER59,ER60,ER65, ER66	17

D04: Diferenças culturais	Diferenças de opiniões, comportamento e valores pessoais	ER03,ER06,ER12,ER14, ER16,ER18,ER22,ER23, ER25, ER26,ER30,ER37, ER38,ER40,ER47,ER49, ER70,ER57,ER58,ER62, ER64	21
D05: Gerenciamento	Dificuldades de alocação, monitoramento e controle das atividades entre os membros das equipes virtuais	ER01,ER03,ER08,ER09, ER14,ER17,ER24,ER27, ER39, ER41,ER42,ER43, ER44,ER48,ER55, ER61	16
D06: Falta de processos, políticas e metodologias comuns	Ausência de padronização nos processos e demais regras entre as equipes distribuídas	ER04,ER13,ER16,ER18, ER21,ER29, ER55	07
D07: Coesão das equipes	Dificuldades em construir o espírito de equipe devido a ausência de contato físico mais próximo	ER05,ER08,ER10,ER27, ER28,ER71,ER34,ER45, ER72, ER50,ER53,ER54, ER57,ER60,ER66	15
D08: Confiança	Problemas na relação de confiança, uma vez que os membros não estão próximos para confiarem nos resultados das atividades que estão sendo conduzidas pelos seus colegas remotos	ER07,ER13,ER35,ER45, ER51,ER56,ER57,ER60, ER62, ER63,ER65	11
D09: Diferença temporal	Projetos podem ter membros em diferentes fusos horários, fazendo com que a sincronização e execução das atividades tornem-se mais difíceis	ER12,ER13,ER34,ER46, ER52,ER59, ER65	07
D10: Distância física	Prejudica o processo de identificação e resolução dos conflitos mais rapidamente, uma vez que todas as partes interessadas não estão presentes no mesmo local de trabalho	ER12,ER13,ER16,ER19, ER31,ER32,ER49,ER56, ER59, ER67	10
D11: Diferenças de infraestrutura e qualidade técnica	Dificuldades relacionadas à questão da qualidade da infraestrutura que suporta as atividades das equipes e dos problemas com a qualificação dos membros das equipes para execução das tarefas atribuídas	ER10,ER34,ER46,ER55	04

Para a visualização das referências dos estudos relevantes podem acessar o seguinte endereço: https://dl.dropboxusercontent.com/u/14825304/ERs.pdf.

Os 11 desafios categorizados demonstram a dificuldade do gerenciamento de conflitos em equipes de DDS, levantando a necessidade de utilização de boas práticas para melhor entendê-los e gerenciá-los com o objetivo de conduzir o projeto ao seu sucesso. É importante que haja um entendimento de como as distâncias físicas e sócio culturais, por exemplo, se relacionam e qual a melhor abordagem de gerenciamento para tratar estes desafios. Por exemplo, Damien e Zowghi (2003) acreditam que os membros das equipes em um ambiente virtual precisam conhecer as diferenças culturais que podem existir, e devem investir em treinamento sobre essas diferenças de forma a melhorar suas abordagens na resolução de conflitos durante o gerenciamento de requisitos.

5. Considerações Finais

Este trabalho teve como objetivo identificar os principais desafios no gerenciamento de conflitos em equipes de desenvolvimento distribuído de software. A maioria dos estudos selecionados datam do ano de 2005 até o presente. Estes dados demonstram que a pesquisa em gerenciamento de conflitos em DDS é algo ainda recente, mas que cresceu muito nos últimos anos, uma vez que as empresas continuam investindo neste contexto de trabalho.

Os 11 desafios encontrados demonstram quão difícil é o processo de gerenciamento dos conflitos em um contexto virtual de desenvolvimento de software. São enfrentados desde problemas de comunicação até problemas de diferenças cultural e temporal, os quais podem atrasar a resolução dos conflitos, gerando problemas que podem resultar em atrasos nas atividades e por consequência comprometendo a entrega final do projeto. Baseado nestes resultados é reforçado como trabalhos futuros a importância da manutenção das pesquisas na área de gerenciamento de conflitos em equipes virtuais, buscando diferentes formas de minimizar o impacto desses desafios, uma vez que o modelo de trabalho distribuído tende a se manter devido à irreversibilidade do processo de globalização.

6. Referências

Amason, A. C. Distinguishing effects of functional and dysfunctional con flict on strategic decision making: Resolving a paradox for top management teams. Academy of Management Journal, v. 39, p.123-148, 1996.

Damien, D. E.; Zowghi, D. An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In 36th Annual Hawaii International Conference on System Sciences (HICSS'03), 2003, Hawaii. Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003. p. 19-28.

Hinds, P. J.; Mortensen, M. Conflict and Shared Identity in Geographically Distributed Teams. The International Journal of Conflict Management, v. 12, n. 3, p. 212-238, 2001.

Hinds, P. J.; Mortensen, M. Understanding Conflict in Geographically Distributed Teams: An Empirical Investigation. Organization science, v. 16, n. 3, p. 290-307, 2005.

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

Mele, C. Conflicts and value co-creation in projects networks. Industrial Marketing Management, v. 40, n. 8, p. 1377-1385, 2011.

Nauman, S; Iqbal, S. Challenges of virtual project management in developing countries. In IEEE International Engineering Management Conference, 2005,Newfoundland. Proceedings of IEEE Intern. Engineering Management Conference, 2005. p. 579-583.

Noblit, G. W.; Hare, R. D. Meta-Ethnography: Synthesizing Qualitative Studies, Newbury Park, CA. Sage Publications Inc.: 1988. p. 9-37.

Prikladinicki, R. MuNDDoS: um modelo de referência para desenvolvimento distribuído de software. 2003. 144 f. Dissertação - Faculdade de Informática, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2003.

Van De Vliert, E.; De Breu, C. K. W. Optimizing Performance by Conflict Stimulation. International Journal of Conflict Management, v. 5, p. 211-222, 1994.

Planejamento de Teste no Desenvolvimento Distribuído de Software: Uma Revisão Sistemática sobre Ferramentas

Scott Y. Takahashi[1], **Tatiane M. P. Lopes**[1]

[1]Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo (UNIFESP) – São José dos Campos, SP – Brasil

Resumo. *Planejar o teste de software desenvolvido no contexto distribuído gera preocupação principalmente em relação à Controle e Coordenação, Colaboração e Comunicação entre os membros das equipes de teste distribuídas: alocar os componentes de software entre as equipes, disponibilizar os artefatos de teste às equipes com permissão para acessá-los, planejar o teste de acordo com os componentes que formam o software, especificar estratégias e técnicas de teste empregadas que possibilite a sua reprodução pelas equipes distribuídas. Este artigo descreve uma revisão sistemática sobre ferramentas para o planejamento de teste dentro do contexto de Desenvolvimento Distribuído de Software considerando tais preocupações.*

1. Planejamento e Condução da Revisão

A presente Revisão Sistemática (RS)S foi planejada com base no modelo proposto em [Kitchenham 2007] em que se apresenta o objetivo da pesquisa, quais as principais questões que guiaram a pesquisa bem como a condução da revisão.

1.1. Objetivo

O objetivo desta RS foi identificar ferramentas para o planejamento da atividade de teste que sejam aplicáveis ao ambiente DDS.

1.2. Questões de pesquisa

As questões que guiaram a condução da RS foram

- Questão Primária: Quais ferramentas para o planejamento da atividade de teste suportam o uso em um contexto de DDS?
- Questão Secundária: Quais características específicas de DDS são abordadas pelas ferramentas encontradas?

1.3. Condução da revisão

A estratégia de busca a ser adotada utilizará o termo de busca, que pode ser definido em conjunto com os operadores lógicos *AND* e *OR*. O termo de busca será dado pelas combinações sequenciais a seguir:

- **S1**: *distributed OR global*
- **S2**: *software development OR software engineering*
- **S3**: *planning OR management*
- **S4**: *testing tool OR testing software*

Neste caso, o termo de busca será formado por: **S1** *AND* **S2** *AND* **S3** *AND* **S4**.

Na Tabela 1 mostram-se as características referentes a DDS tais como Controle e Coordenação, Colaboração e Comunicação e à Cobertura do planejamento de teste das ferramentas encontradas na RS.

Table 1. Ferramentas com suporte ao planejamento da atividade de teste no contexto de DDS.

Ferramenta	Objetivo	Controle e Coordenação	Colaboração	Comunicação	Cobertura do planejamento
Test-Link	Suporte à especificação, execução e monitoramento da atividade de teste.	Por meio de permissões de usuário	Permite o gerenciamento remoto	Não possui um meio específico. É feito através da atualização das tarefas	Permite criar planos de teste, controlar o nível de acesso do mesmo e obter a cobertura dos testes.
SpiraTeam	Gerenciamento de projeto	Por meio de permissões de usuário	Por meio de um *dashboard* e um gerenciador de *issues*	Não existe uma funcionalidade própria. É feito pelo *dashboard* e o gerenciador de *issues*	Permite gerenciar e executar os casos de teste, além de obter a cobertura de testes
IBM Rational Team Concert	Gerenciamento de projeto	É feito através da criação de times	É feito através da criação de times	É feito utilizando notificações e mensagens	É possível criar *issues* como planos de teste e atualizar o estado de cada uma destas.
Quality Center	Gerenciamento de QA (*Quality Assurance*)	Não mencionado	Por meio de um *dashboard*	Por meio de um *dashboard*	É possível criar planos de teste. Suporta teste de integração e unidade.
Wip-CAFÉ TMS	Gerenciamento e automação de teste	Não menciona	Não menciona	Não menciona	É possível criar especificações de teste

References

Kitchenham, B. (2007). Guidelines for performing systematic reviews in Software Engineering. Technical report, Keele University and University of Durham.

Uso de Mapas Mentais Como Ferramenta Auxiliar ao Desenvolvimento Distribuído de Software

William Simão de Deus[1], José Antônio Gonçalves[2], Alexandre L'Erario[3]

[1-2-3]Universidade Tecnológica Federal do Paraná (UTFPR) Cornélio Procópio – PR – Brasil

williamsimao@outlook.com, {alerario, jgoncalves}@utfpr.edu.br

1. Introdução

Junto com a evolução da globalização relacional de empresas, a globalização do desenvolvimento distribuído de software (DDS) tornou-se uma realidade, aumentado distâncias entre equipes que desenvolvem um mesmo projeto [Redmiles *et al*., 2007]. O DDS implica em benefícios e problemas [Prikladnicki e Audy 2002]. Segundo os autores, há fatores que impulsionam vantagens (custos econômicos, temporais, qualidade e flexibilidade de projetos) e desafios (comunicação, configuração da equipe, estrutura do trabalho e sincronia entre os *stakeholders*).

A fim de minimizar os riscos e aumentar a capacidade em gerir projetos DDS, este artigo apresenta a utilização de mapas mentais como forma de auxilio para seu desenvolvimento. Esta pesquisa abrange a área organizada a partir do estudo de caso de cinco projetos que simulam o ambiente DDS em um curso de graduação. Os resultados identificaram a configuração do ambiente e os fatores de deficiência mapeados durante o desenvolvimento do projeto.

2. Metodologia

O protocolo aplicado nessa pesquisa analisou um experimento acadêmico que ocorreu na Universidade Tecnológica Federal do Paraná – Campus Cornélio Procópio, aplicado aos alunos do segundo e quarto período do curso superior em Análise e Desenvolvimento de Sistemas. A fim de simular um ambiente de DDS formaram-se equipes com diferentes graus de conhecimento entre alunos de períodos diferentes. Foram organizadas oito equipes em cada período composta de três integrantes. Durante o projeto cada equipe se correspondeu unicamente à outra equipe de período diferente. Ambas deveriam trabalhar de forma colaborativa e cooperativa, organizando o desenvolvimento de um software em conjunto, ocasionando obrigatoriamente uma interação entre equipes de períodos diferentes dispersas temporalmente.

O experimento foi alicerçado a partir do mapeamento por equipes do quarto período. Este mapeamento apresentava requisitos, funcionalidades, contexto de aplicação e a configuração do desenvolvimento de um software. O escopo dessa pesquisa verificou dois fatores de configuração. O primeiro é a análise desse mapeamento, visando ás dificuldades de compreensão e a linguagem conceitual apresentada durante um estágio inicial de desenvolvimento. O segundo escopo verificou o desenvolvimento do projeto, analisando a comunicação e a conclusão do mesmo.

A partir da análise de cinco estudos de casos ocorreu uma compilação, agregando apenas um resultado que apresenta fatores chaves encontrados em todas as equipes.

3. Resultados

Os escopos de observação resultantes foram classificados como Documentação e Comunicação, e ocorrem devido à disparidade temporal e conceitual entre integrantes, sendo observado em todas as equipes. A Documentação representa o escopo total do projeto, e foi organizada como: *Requisitos*, *Apresentação* e *Mapeamento*. Nota-se uma dependência nesta composição, caso algum fator apresente deficiência, provavelmente, outros também apresentarão. Estes fatores são apresentados a seguir:

Requisitos: caracterizados por métricas não estabelecidas e especificações incompletas que representaram incongruências entre funcionalidades do projeto. *Apresentação*: compreensão do escopo do projeto, sendo uma identificação entre a construção, desenvolvimento e sua finalização. Os estudos de caso demonstraram que a Documentação se preocupava com a finalidade do projeto, deixando de lado seu desenvolvimento, criando assim lacunas para a compreensão. *Mapeamento*: definido como análise, progressão e finalidade do projeto, auxiliando sua implementação. Os estudos de caso demonstraram que as equipes não traçaram métricas durante o projeto, não se preocupando com a organização temporal entre integrantes.

Outro ponto observado corresponde à comunicação e refere-se à partilha da informação, com a finalidade da interação entre membros da mesma equipe. No experimento realizado, percebeu-se que as equipes direcionavam sua atenção com a entrega final do projeto, deixando de lado a projeção para seu desenvolvimento. A organização e comunicação, neste estágio, apresentaram falhas, devido a componentes de uma mesma equipe que não conseguiam apresentar conclusões ou dúvidas sobre o projeto, criando falha de interação entre a equipe e dificultando o desenvolvimento.

5. Conclusão

Este artigo apresentou a análise dos resultados de diferentes estudos de caso sobre o DDS no meio acadêmico. Com a finalidade de utilizar mapas mentais como uma ferramenta de auxilio ao DDS. Algumas equipes elegeram formas de organização e desenvolvimento, atendendo as especificações do projeto, outras equipes encontraram maiores dificuldades aliando o problema à documentação. Dentre os recursos disponíveis para a realização deste experimento encontravam-se mecanismos de organização e engenharia de software para mapear o progresso. As equipes que o utilizaram apresentaram maior facilidade para seu desenvolvimento e sua compreensão.

A simulação um ambiente DDS dentro da graduação gerou habilidades tecnológicas aprofundando a importância da comunicação, divisão de tarefas e organização. Ao final do estágio pesquisado, as equipes que organizaram o desenvolvimento do software, simulando uma ordenação mental entre membros e tarefas desenvolvidas concluíram seus projetos com maior facilidade e qualidade.

Referências

Prikladinicki, R.; Audy, J. L N. (2002). “Desenvolvimento Distribuído de Software e Processos de Desenvolvimento de Software”. 66 f. Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2002.

Redmiles, D. *et al.* (2007). “Continuous Coordination : A New Paradigm to Support Globally Distributed Software Development Projects”, 1–18.

An Empirical Study about Testing in a Distributed Software Project

Adailton M. Lima , Carla A. Lima Reis, Rodrigo Quites Reis

LABES / PPGEE / Federal University of Pará - UFPA

Belém, Brazil

{adailton, clima, quites}@ufpa.br

***Abstract**: We present the results of an empirical study carried out to identify the context and factors that can affect the productivity of a team of testers in distributed software projects. This information helped us to identify communication delay, knowledge expertise and software quality as the main factors that affected testers productivity.*

1. Introduction

This paper discusses some factors that can affect testing team productivity in DSD projects. Our findings are based on data collected in a testers team that worked in a project distributed over twelve different teams in an international company. The information we collected during this study helped us to identify the communication delay and the domain knowledge as some of the main factors that could have affected the testers productivity.

2. Study Context

The studied project had six different sites and was distributed over USA, Europe and India, and also had more than three thousands of requirements and enrolled nearly one hundred and fifty developers. Such sites adopted Scrum-based management practices to help them to coordinate project releases and activities planning.

The studied context represents a largely distributed project and is different from current studies in literature, like the work of [Martin 2007] that also investigated the issues involved in testing software in DSD, but in this case in a team of a small software company. Additionally, this context is similar to the study of [Grechanik 2010] that investigated testing centers working as isolated sites for software testing with no or little interaction with project developers.

3. Reported Problems and Practices

According to the testers their work was heavily affected by their knowledge expertise and the communication dependencies to the remote teams that they had to interact with. It has been reported that these factors affected their project schedule and the quality of their deliverables.

The main issue related to the knowledge expertise described by the testers is that the centralized testing strategy caused a dependency to the domain of all different software modules. Then, when testing software modules were delivered by different development sites the testers had to change the work context many times.

As the testing activities occur after a development team considers their functionality done, in many cases the communication requests made by the testers were answered after a long time with low priority by the developers. This happened because the targeted developer was already allocated with the development of new features. In this context, the testers reported that the manager of a remote team does not allow direct contact to the developers because he thought it could spend or waste their time.

Also related to knowledge expertise and poor documentation, when they had broken features, they faced difficulty to contact the right person to clear their doubts. Even when they found the right person, he could be busy for a remote meeting or even for chatting. In this case, a tester reported that he simply used to postpone the solution of the problem until he gets a vacancy in the agenda of the developer, causing in many tasks a delay time. As a practice to solve the communication difficulties, the testers also reported that had a local developer that they used to have informal conversations to get some tips to help them to guess a way to test it while they waited for a formal answer from the remote developers.

The problems and practices reported in this study are not new, but corroborates with the little research we can find about testing in a DSD context, like the work of [Grechanik 2010] and [Mathrani 2013].

4. Conclusions

This paper presented the results of a study carried out to identify the context and factors that can affect the productivity of a team of testers in a distributed software project. From our results, it is clear that, corroborating with other studies in the context of DSD projects, the productivity of the testers was negatively impacted by knowledge, communication and quality of the products delivered by the developers.

As a limitation of this work data analysis, the project metrics present on managerial spreadsheets, such as story points, could not be compared between different teams because they mean a perception from a group about project size and estimations. Thus, the results discussed in this study are based only in the perception and experience of project members during the software development life cycle.

We also have a conflict between point of views in the project: as project manager and software architect believed that the domain knowledge was not a big issue on this project, the testers reported that it has impacted negatively their productivity. The reason was the absence of required documentation to fully understand the system functionalities. The testers also reported that they needed to understand the whole system domain to be able to successfully execute system integration testing. Poor software cohesion and high coupling between software modules could cause this problem.

5. References

Grechanik, M.; Jones, J. A.; Orso, A.; Van der Hoek, A. “Bridging gaps between developers and testers in globally-distributed software development”. In Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10). ACM, New York, NY, USA, 149-154.

Martin, D.; Rooksby, J.; Rouncefield, M.; Sommerville, I. “'Good' Organisational Reasons for 'Bad' Software Testing: An Ethnographic Study of Testing in a Small Software Company”. In Proceedings of the 29th international conference on Software Engineering (ICSE '07). IEEE Computer Society, Washington, DC, USA, 602-611.

Mathrani, A.; Mathrani, S.; “Test strategies in distributed software development environments”. In: Computers in Industry, Volume 64, Issue 1, January 2013, Pages 1-9.

Proposta de cenários genéricos para definição da comunicação em sítios de Desenvolvimento Distribuído de Software

José Antônio Gonçalves[1], Leonardo Sanches dos Santos[2], Alexandre L'Erario[3]

[1-2-3]Programa de Pós-graduação em Informática - Universidade Tecnológica Federal do Paraná - Cornélio Procópio – PR – Brasil
{jgoncalves, alerario}@utfpr.edu.br, leo.utfpr@gmail.com

1. Introdução

O Desenvolvimento Distribuído de Software é definido e caracterizado principalmente pela colaboração e cooperação entre departamentos de organizações, e pela criação de grupos de pessoas que trabalham em um projeto comum, porém, localizados de forma geográfica e temporalmente dispersos [Prikladnicki et al 2004, Huzita et al 2007].

Segundo Martins (2010), a comunicação é essencial a quaisquer tipos de projetos, independentemente de suas complexidades. Nestes faz-se necessário um fluxo constante de informações sobre o andamento e repasse das atividades entre os *stakeholders*. Afirma também que a comunicação é de extrema importância no desenvolvimento de um projeto, por meio dela as informações geradas são distribuídas aos envolvidos e armazenadas de forma adequada e consistente.

O objetivo deste projeto foi mapear a comunicação que ocorreu em ambientes de DDS, com o intuito de definir a relação entre sua quantidade e qualidade, para então criar um conjunto de cenários para definição da comunicação em sítios de DDS, o estudo de caso foi o método escolhido para tal.

Para isto foram utilizadas duas pesquisas precursoras a esta, sendo, uma delas um experimento controlado para o qual foi criado e configurado um cenário de maneira a se comportar como um ambiente de DDS. Neste foram desenvolvidos cinco projetos de software envolvendo dez grupos de pessoas que se interagiam de forma colaborativa, como descreve L´Erario (2009), de maneira a reforçar este ambiente de DDS.

A outra pesquisa trata-se de um estudo de caso aplicado em uma empresa real, atuante no mercado de desenvolvimento de software e que exerce suas atividades em regime de DDS. Neste caso, por meio de um acompanhamento sistemático do fluxo de comunicação, fez-se um levantamento quantitativo da comunicação ocorrida durante o andamento dos projetos.

A metodologia adotada neste projeto foi a de estudo de caso, pois segundo Piaget (1977), que afirma que a essência de um estudo de caso, ou seja, a tendência central entre todos os tipos de estudo de caso é tentar iluminar uma decisão: por que elas foram tomadas, como foram implementadas e com quais resultados. Para este projeto os itens "porque elas foram tomadas" e "como foram implementadas" são esclarecidos por meio das duas outras pesquisas integrantes do projeto e apresentadas neste artigo. O item "com quais resultados" é respondido pelo conjunto de cenário ao final.

A pergunta que deflagrou o trabalho deste artigo foi: "*É possível criar um conjunto de cenários genéricos para definição da comunicação em DDS?"*. A hipótese levantada a esta pergunta foi a de que *"é possível, se considerar os levantamentos das informações quantitativas e qualitativas desta comunicação"*.

2. Conclusão

Por meio do estudo dos casos, sobre as duas pesquisas apresentadas, atingiu-se a meta da proposta deste trabalho, ou seja, a criação de cenários genéricos para definição da comunicação em ambientes de DDS.

Concluiu-se que o conjunto das variáveis existentes no contexto de um projeto de software é determinante para a definição do perfil do cenário. Por meio da análise deste perfil se pode definir a classificação da comunicação.

A classificação da comunicação poderia ser eficiente ou ineficiente, e se esta situação era ou não esperado. Esta expectativa também é gerada pelas variáveis do ambiente como, por exemplo, o grau de conhecimento em detrimento das quantidades de iterações entre os atores.

Sendo assim, firmou-se a hipótese deste artigo, que conclui ser possível criar um conjunto de cenários genéricos para definição da comunicação em DDS, desde que se faça um levantamento qualitativo e quantitativo desta comunicação.

3. Referências

Carmel, E. (1999). "Global software teams: collaborating across borders and time zones". Prentice Hall PTR, Upper Saddle River, NJ, USA.

Huzita, E. H., Tait, T. F., Colanzi, T. E., and Quinaia, M. A. (2007). "Um ambiente de desenvolvimento distribuído de software-disen. In Workshop de Desenvolvimento Distribuído de Software", pages 31–38.

L'Erario, A. (2009). "M3DS: um modelo de dinâmica de desenvolvimento distribuído de software". Teste de doutorado. Universidade de São Paulo – USP.

Martins, J. C. C. (2010). "Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML" (5ª edição). Brasport.

Piaget, J., Schramm, Y., and Lenardon, E. (1977). "O julgamento moral na criança". Editora Mestre Jou.

Prikladnicki, R., Lopes, L., Audy, J. L. N., and Evaristo, R. (2004). "Desenvolvimento distribuído de software: um modelo de classificação dos níveis de dispersão dos stakeholders". In I Brazilian Symposium on Information Systems (SBSI 04).

Travassos, G. H., Gurov, D., and Amaral, E. (2002). Introdução à engenharia de software experimental. UFRJ.

Wohlin, C., Runeson, P., H¨ost, M., Ohlsson, M. C., Regnell, B., and Wessl´en, A. (2012). "Experimentation in software engineering". Springer.

Yin, R. K. (2002). "Case study research: Design and methods", volume 5. SAGE Publications, Incorporated.

Desafios na Sistematização das Evidências Empíricas em Desenvolvimento Distribuído de Software

Antônio Rafael da Rosa Techio, Rafael Prikladnicki

Pontifícia Universidade Católica do Rio Grande do Sul – Porto Alegre – RS – Brasil

antonio.techio@acad.pucrs.br, rafael.prikladnicki@pucrs.br

Abstract. *Distributed Software Development (DSD) is a recent research area and has been growing over the past few years. However due to the large amount of terms involved in this scenario, concepts are replete with misleading usage, making the process of search for empirical evidence a complex task. In this paper we present and discuss the challenges in the systematization of the empirical evidences in DSD and the results of an analysis of three studies that have been published aiming at proposing solutions for the identified challenges.*

Resumo. *O Desenvolvimento Distribuído de Software (DDS) é recente como área de pesquisa e vem crescendo ao longo dos últimos anos. No entanto devido a grande quantidade de termos envolvidos nesse cenário, muitas vezes um conceito é utilizado de forma equivocada, tornando o processo de busca por evidências empíricas uma tarefa complexa. Neste artigo são apresentados e discutidos os desafios na sistematização das evidências empíricas em DDS e o resultado de uma análise sobre três estudos publicados com o objetivo de propor soluções para estes desafios.*

1. Introdução

O objetivo deste artigo é iniciar uma discussão sobre as dificuldades envolvidas na sistematização das evidências empíricas em DDS. A terminologia usada para o DDS não é padronizada. Os cenários de DDS são diversos e o que se aplica em um contexto talvez não se aplique diretamente em outro. O DDS ainda é uma área imatura, tornando a sistematização das evidências empíricas uma tarefa complexa. Na próxima seção, apresenta-se uma análise sobre os desafios e soluções encontrados em três estudos que apresentam uma discussão neste sentido.

2. Desafios na Sistematização das Evidências Empíricas em DDS

Sistematizar evidências empíricas em DDS não é uma tarefa trivial e possui alguns desafios, conforme relatado em revisões sistemáticas recentes da área. A seguir apresentam-se os principais desafios e soluções encontradas em três estudos: Prikladnicki & Audy (2010), Smite et al. (2010) e Smite et al. (2012).

Os principais desafios relatados nos estudos estão relacionados com a terminologia da área de DDS, a descrição de contexto dos estudos empíricos e o processo de revisão sistemática da literatura. Muitas vezes o contexto de um estudo empírico em DDS não é descrito de forma clara, o que faz com que seja difícil para

pesquisadores e profissionais identificarem casos que podem ser de interesse para eles. A terminologia aplicada ao DDS não é padronizada, uma vez que existem diversos estudos que utilizam os mesmos termos de forma diferente.

No estudo Prikladnicki e Audy, 2010, os autores utilizaram os termos mais populares relacionados com o campo como uma solução para a falta de padronização da terminologia em DDS. Como resposta ao problema de informação de contexto, os autores criaram uma taxonomia com base em estudos anteriores para organizar uma determinada área e classificar os estudos na taxonomia. Por fim, para revisão sistemática, os autores utilizaram *string* de busca com os termos mais populares da área, incluindo suas principais variações e recomendaram envolver mais de um pesquisador no processo de revisão.

No estudo Smite et al., 2010, os autores também utilizaram os termos mais populares relacionados com o campo. Para melhorar a forma como o contexto de um estudo é descrito, os autores criaram um esquema de classificação como um guia para estruturação e coleta de dados. Em fim, para melhorar o processo de revisão sistemática, utilizou-se *string* de busca aplicada no texto completo do artigo e o recomendou-se envolver mais de um pesquisador no processo revisão.

No estudo Smite et al., 2012, aplicou-se uma *survey* com especialistas da área para criar uma terminologia padrão e um glossário comum de termos. Em relação à informação de contexto, os autores criaram uma taxonomia para classificação hierárquica da área, ajudando, portanto, os pesquisadores a classificarem seus próprios estudos e estudos relacionados. Para revisão sistemática, os autores aplicaram um processo manual e sistemático de revisão dos artigos utilizando as principais conferências e edições especiais de diferentes revistas.

3. Conclusão

A terminologia aplicada ao DDS não é padronizada e muitas vezes o contexto de um estudo empírico não é descrito de forma clara, gerando, portanto, uma oportunidade para propor uma sistematização das evidências empíricas nesta área.

Por fim, terminologias padronizadas e taxonomias podem ajudar no amadurecimento de uma área e, consequentemente, podem ser utilizadas ou adaptadas em outros domínios ou subáreas de pesquisa em Engenharia de Software.

Referências Bibliográficas

Prikladnicki R., Audy J. L. N. (2010) “Process Models in the Practice of distributed Software Development: A Systematic Review of the Literature”, Information and Software Technology 52(8):779–791.

Smite D., Wohlin C., Feldt R., Gorschek T. (2010) “Empirical Evidence in Global Software Engineering: A Systematic Review”, Empirical Software Engineering Journal 15(1):91–118.

Smite, D., Wohlin, C., Galvina, Z., Prikladnicki, R. (2012) “An Empirically Based Terminology and Taxonomy for Global Software Engineering”, Empirical Software Engineering: An International Journal, DOI: 10.1007/s10664-012-9217-9.