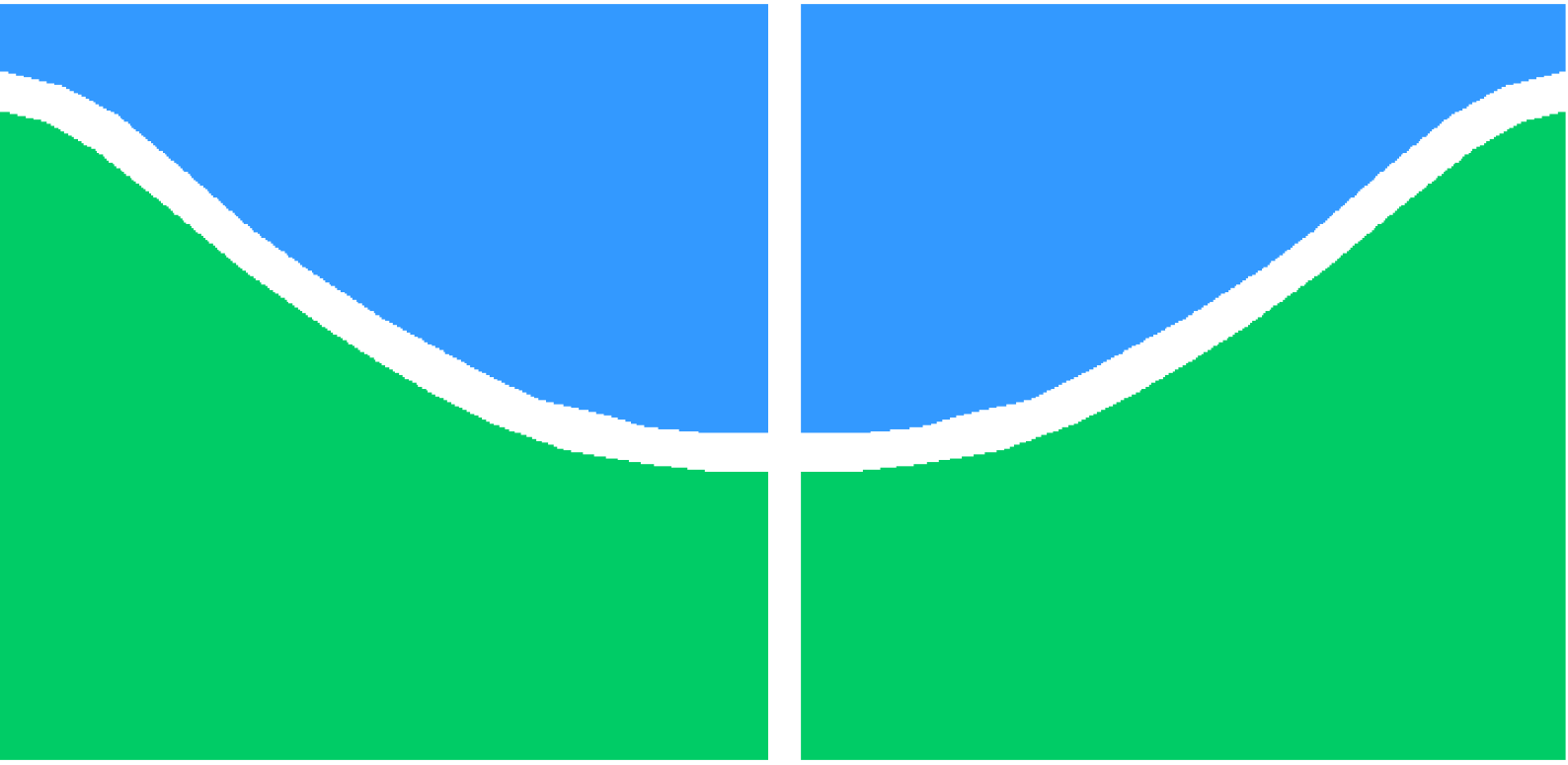

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Aplicando técnicas ágeis de usabilidade em projetos de software livre: um estudo de caso para o projeto Mezuro

Autor: Renan Costa Filgueiras
Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF
2013

Renan Costa Filgueiras

Aplicando técnicas ágeis de usabilidade em projetos de software livre: um estudo de caso para o projeto Mezuro

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2013

Renan Costa Filgueiras
Aplicando técnicas ágeis de usabilidade em projetos de software livre: um estudo de caso para o projeto Mezuro / Renan Costa Filgueiras. – Brasília, DF, 2013-
35 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. Usabilidade. 2. Software livre. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aplicando técnicas ágeis de usabilidade em projetos de software livre: um estudo de caso para o projeto Mezuro

CDU 02:141:005.6

Renan Costa Filgueiras

Aplicando técnicas ágeis de usabilidade em projetos de software livre: um estudo de caso para o projeto Mezuro

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 19 de julho de 2013:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Prof. Dr. Nilton Correia da Silva
Convidado 1

Profa. Dra. Milene Serrano
Convidado 2

Brasília, DF
2013

Resumo

Palavras-chaves: usabilidade. software livre.

Abstract

Key-words: social networking. open-source. federation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UnB	Universidade de Brasília

Sumário

1 Introdução

1.1 Objetivos

Esta seção apresenta os objetivos gerais e específicos deste TCC.

1.1.1 Objetivos Gerais

1.1.2 Específicos

1.2 Organização do Trabalho

2 Software Livre

2.1 SL

De acordo com Richard Stallmann um programa é um software livre se os usuários possuírem 4 liberdades essenciais:

- A liberdade de executar o programa, para qualquer propósito (liberdade 0).
- A liberdade de estudar como o programa funciona, e adaptá-lo às suas necessidades (liberdade 1). Para tanto, acesso ao código-fonte é um pré-requisito.
- A liberdade de redistribuir cópias de modo que você possa ajudar ao próximo (liberdade 2).
- A liberdade de distribuir cópias de suas versões modificadas a outros (liberdade 3).

Desta forma, você pode dar a toda comunidade a chance de beneficiar de suas mudanças. Para tanto, acesso ao código-fonte é um pré-requisito.

3 Métodos Ágeis

Métodos ágeis (AM) é uma coleção de metodologias baseada na prática para modelagem efetiva de sistemas baseados em software. É uma filosofia onde muitas metodologias se encaixam.

As metodologias ágeis aplicam uma coleção de práticas, guiadas por princípios e valores que podem ser aplicados por profissionais de software no dia a dia ()

3.1 Metodologia Ágil

A expressão "Metodologias Ágeis"tornou-se popular em 2001 quando dezessete especialistas em processos de desenvolvimento de software representando diversos métodos como Scrum (), Extreme Programming (XP) e outros, estabeleceram princípios comuns compartilhados por todos esses métodos. Foi então criada a Aliança Ágil e o estabelecimento do Manifesto Ágil () Os conceitos chave do Manifesto Ágil são: **Indivíduos e interações** ao invés de processos e ferra- mentas. **Software executável** ao invés de documentação. **Colaboração do cliente** ao invés de negociação de con- tratos. **Respostas rápidas a mudanças** ao invés de seguir pla- nos. O Manifesto Ágil não discrimina processos, ferramentas, documentação, negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparado com os indivíduos, interações, software executável, participação do cliente e feedback rápido a mudanças e alterações.

Tory Diba em uma revisão sistemática elenca alguns dos principáis métodos ágeis presente em 2008 e trás uma breve descrição de cada um, o que ajuda a observar caractériscas das camunidades ágeis já que os principais especialistas desses métodos ajudaram a escrever o manifesto ágil.()

3.2 Princípios Ágeis

O manifesto ágil define 12 princípios que devem ser seguidos **??**:

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.

2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.

3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.

4. Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.

5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.

6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.

7. Software funcional é a medida primária de progresso.

8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.

9. Contínua atenção à excelência técnica e bom design, aumenta a agilidade.

10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.

11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.

12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

Os princípios ágeis devem ser vividos pela equipe e é o principal motivador e base para o desenvolvimento ágil. As práticas ágeis adotadas serão de acordo com o método escolhido e ajudarão a obter os objetivos propostos pelo método, mas são nos princípios ágeis isso sempre deve ser fundamentado de acordo com o manisfeto ágil.

Método Ágil	Descrição
Crystal methodologies	Crystal Clear, centra-se na comunicação de pequenas equipes de desenvolvimento de software que não possui ciclo de vida crítico. Seu desenvolvimento tem sete características: entrega frequente, melhoria reflexivo, comunicação osmótica, segurança pessoal, foco, fácil acesso a usuários experientes e os requisitos para o ambiente técnico
Dynamic software development method (DSDM)	Divide projetos em três fases: pré-projeto, o ciclo de vida do projeto e pós projeto. Nove princípios subjacentes ao DSDM: o envolvimento do usuário, capacitando da equipe do projeto, a entrega frequente, atender às necessidades de negócios atuais, desenvolvimento iterativo e incremental, permitir mudanças, o escopo de alto nível a ser fixados antes do início do projeto, testar todo o ciclo de vida e eficiente e eficaz comunicação
Feature-driven development	Combina desenvolvimento model-driven e ágil, com ênfase em modelo inicial de objeto, a divisão do trabalho em funções e design iterativo para cada recurso. Afirma ser adequado para o desenvolvimento de sistemas críticos. Uma iteração de um recurso é composto de duas fases: concepção e desenvolvimento
Lean software development	Uma adaptação de princípios de produção de carne magra e, em particular, o sistema de produção Toyota para desenvolvimento de software. Consiste em sete princípios: eliminar o desperdício, aumentar a aprendizagem, decidir o mais tarde possível, entregar o mais rápido possível, capacitar a equipe, construir integridade, e ver o todo
Scrum	Centra-se na gestão de projetos em situações em que é difícil planejar o futuro, com mecanismos de " controle de processos empíricos ", onde loops de feedback constituem o elemento central. Software é desenvolvido por uma equipa de auto-organização em incrementos (chamado sprints), começando com o planejamento e terminando com uma retrospectiva. Recursos a serem implementadas no sistema estão registrados em um backlog . Em seguida, o proprietário do produto decide quais itens do backlog deve ser desenvolvido da seguinte sprint. Coordenar membros da equipe de seu trabalho em uma reunião diária de stand-up. Um membro da equipe, o scrum master, é responsável pela resolução de problemas que impedem a equipe de trabalho de forma eficaz.
Extreme programming (XP,XP2)	Concentra-se nas melhores práticas para o desenvolvimento. Consiste em doze práticas: o jogo de planejamento, pequenos lançamentos, metáfora, design simples, testes, refatoração, programação em pares, propriedade coletiva, integração contínua, 40 h semana, os clientes no local, e os padrões de codificação. A revista "XP2"consiste nas seguintes "práticas primárias": sentar-se juntos, toda a equipe, trabalho informativo, o trabalho energizado, programação em pares, histórias, ciclo semanal, ciclo trimestral, slack, construção de 10 minutos, integração contínua, testes primeiro que programação e design incremental. Há também 11 "corollary practices".

Tabela 1 – Descrição dos principais métodos ágeis de desenvolvimento

4 Usabilidade

4.1 Terminologias

O termo usabilidade de modo geral pode ser escrito como a facilidade com a qual um equipamento ou programa pode ser usado. Esse termo dentro da computação foi diversas vezes refinado como nas ISO 9126, 12119, 9241, 14598 e, por fim, na ISO 25010, que define como uma medida pela qual um produto pode ser usado por usuários específicos para alcançar metas específicas com eficácia, eficiência e satisfação em um contexto específico de uso ().

A usabilidade não é uma qualidade intrínseca de um sistema, é dependente de um acordo entre as características de sua interface e as características de seus usuários na busca de determinados objetivos e situação de uso (). Por esse motivo uma interface que pode ser considerada satisfatória para determinado grupo de usuários pode ser inviabilizada por outros, como usuários experientes *versus* novatos, além de uma percepção diferente dependendo do ambiente onde esse sistema se encontra, um computador lento *versus* computador rápido. Dessa forma, podemos definir que a usabilidade é um acordo entre interface, usuário, tarefa e ambiente. Baseado nesta definição é que pautaremos nossas discussões e estudos de caso neste trabalho.

A necessidade de se garantir que sistemas e dispositivos estejam adaptados à maneira como o usuário pensa, comporta-se e trabalha, entra o conceito de ergonomia. Tal conceito surgiu logo após a II Guerra Mundial, como consequência do trabalho interdisciplinar realizado por diversos profissionais, tais como engenheiros, fisiologistas e psicólogos, durante a guerra ().

Há algumas definições formais para o termo "ergonomia", de acordo com a *Ergonomics Society*, a Associação Brasileira de Ergonomia e a *International Ergonomics Association*. Para este trabalho, adotamos a definição da Associação Brasileira de Ergonomia, que a conceitua como o estudo das interações das pessoas com a tecnologia, a organização e o ambiente, objetivando intervenções e projetos que visem melhorar, de forma integrada e não-dissociada, a segurança, o conforto, o bem-estar e a eficácia das atividades humanas ().

Neste contexto, a questão que norteia este trabalho é como pode-se avaliar, entender, verificar, observar a interface de uma aplicação em determinado contexto ou sistema? Para respondermos essa questão, mapeamos as definições de alguns especialistas em usabilidade e ergonomia, que estabeleceram critérios, regras e princípios para nortear essa necessidade.

- Jakob Nielsen, em seu livro *Usability Engineering* , propõe um conjunto de dez heurísticas de usabilidade ():
  - Viabilidade do estado do sistema;
  - Mapeamento entre o sistema e o mundo realizada;
  - Liberdade e controle ao usuário;
  - Consistência e padrões;
  - Prevenção de erros;
  - Reconhecer em vez de relembrar;
  - Flexibilidade e eficiência de uso;
  - Design estético e minimalista;
  - Suporte para o usuário reconhecer, diagnosticar e recuperar erros;
  - Ajuda e documentação.
- Ben Shneiderman, em seu *livro Designing The User Interface*, propõe, o que ele denominou de "oito regras de ouro":
  - Perseguir a consistência;
  - Fornecer atalhos;
  - Fornecer feedback informativos;
  - Marcar o final dos diálogos;
  - Fornecer prevenção e manipulação simples de erros;
  - Permitir o cancelamento das ações;
  - Fornecer controle e iniciativa ao usuário;
  - Reduzir a carga de memória de trabalho.
- Christian Bastien e Dominique Scapin definiram 8 critérios ergonômicos:
  - Condução;
  - Carga de trabalho;
  - Controle;
  - Adaptabilidade;
  - Gestão de erros;
  - Coerência;
  - Significado dos códigos;

– Denominações e Compatibilidade.

Portanto, baseado nas heurísticas e critérios listados acima, Walter Cybis, no livro Ergonomia e Usabilidade, propôs uma tabela que relaciona todas essas definições, conforme aprensentado na Tabela 2 ().

Condução	Qualidade da ajuda e da documentação Adequação ao aprendizado Apresentação do estado do sistema Convite Agrupamento e distinção por localização Agrupamento e distinção por formato Feedback imediato
Carga de trabalho	Legibilidade Brevidade das entradas individuais Concisão das apresentações individuais Ações mínimas Densidade informacional Design minimalista e estético
Controle	Ações explícitas Controle do usuário
Adaptabilidade	Flexibilidade Personalização Consideração da experiência do usuário
Gestão de erros	Proteção de erros Tolerância aos erros Qualidade das mensagens de erro Correção de erros
Coerência	Homogeneidade interna a uma aplicação Homogeneidade externa a plataforma
Significado dos códigos e denominações	Interface clara
Compatibilidade	Compatibilidade com o usuário Compatibilidade com a tarefa dos usuários Compatibilidade com a cultura dos usuários

Tabela 2 – Conjunto integrador de critérios, princípios, regras e heurísticas de ergonomia

4.2 Práticas de Usabilidade Ágeis

4.3 Usabilidade em Software Livre

Para aplicação em ambientes distribuídos, abertos e colaborativos, como em comunidades de software livre, implica-se uma adaptação em métodos de usabilidade. Isso ocorre porque em comunidades de desenvolvimento de software livre, não se pode garantir

a existência de um indivíduo especialista em usabilidade ou de uma equipe dedicada a essas atividades. A distância física de membros que contribuem com o projeto também dificulta a utilização de métodos de usabilidade que dependem de comunicação face a face. Mesmo assim, a usabilidade deve ainda ser considerada, afinal ela é muito importante para a criação de um sistema de qualidade, em qualquer ambiente.

A usabilidade deve fazer parte do desenvolvimento, não apenas em busca de produtos usáveis, mas também de práticas usáveis e de modo a envolver todos os membros de uma equipe considerando o contexto em que as práticas serão aplicadas. Dessa forma, não se tem uma equipe de acordo com os valores de métodos ágeis e de métodos de usabilidade se apenas alguns se preocupam em atender as necessidades dos usuários típicos e clientes, pois todos os membros precisam entender a importância de atendê-las. () Ana Paula Oliveira dos Santos, em sua dissertação de mestrado, propôs práticas de usabilidade para a comunidade de software livre através da associação das fases do DCU(Design Centrado no Usuário) com as técnicas de usabilidade da comunidade ágil conforme a tabela 3.

Práticas de usabilidade para Software Livre	
Fases DCU	**Práticas de Usabilidade**
Identificar necessidades para design centrado em humano	Equipe-Núcleo como Donos do Produto Caminhos Completos Especialista-generalista
Especificar contexto de uso	Pouco design antecipado e distribuído
Especificar Requisitos	Definir metas de usabilidade automáticas
Avaliar designs	RITE (Rapid Iterative Testing and Evaluation) para desenvolvedores de software livre

Tabela 3 – Práticas de usabilidade no contexto de Software Livre

As práticas são descritas apresentando um contexto, um problema e uma solução visando a adequação a comunidades de software livre.

Identificar necessidades para design centrado em humano

Equipe-Núcleo como Donos do Produto

Contexto: Equipe de desenvolvimento de software livre composta por desenvolvedores e que não possui especialistas em usabilidade ou UX como membros. Contudo, a equipe-núcleo do projeto percebe a necessidade de compreender melhor os requisitos de negócios e de usabilidade, levando em consideração a visão de clientes e usuários típicos. Problema: Integrar requisitos de negócio com requisitos de usabilidade em um ambiente que não possui especialistas em usabilidade. Principais forças envolvidas:

- Força 1: Necessidade de levantamento de requisitos de negócios com clientes e requisitos de usabilidade com usuários típicos, de modo a integrá-los para o desenvolvimento do sistema.

- Força 2: Não existe garantia de que especialistas em usabilidade ou UX participarão voluntariamente do projeto e/ou não é possível contratá-los. Também não é possível garantir que desenvolvedores voluntários queiram participar dessas atividades.

Solução: Uma adaptação da prática Especialistas em UX como Donos do Produto, da comunidade de métodos ágeis, na qual a equipe-núcleo de um projeto de software livre assumiria o papel de Proprietários do Produto, que levam em consideração a usabilidade do sistema. Dessa forma, podem controlar as contribuições para o projeto, com a visão das necessidades de usuários típicos e clientes.

Caminhos Completos

Contexto: Equipe de desenvolvimento de software livre composta por desenvolvedores e que não possui especialistas em usabilidade ou UX como membros. Contudo, a equipe-núcleo do projeto precisa empregar práticas de usabilidade durante o desenvolvimento do sistema. Problema: Realizar práticas de usabilidade em projetos de software livre que não possuem especialistas em usabilidade ou UX. Principais forças envolvidas:

- Força 1: Necessidade de realizar práticas de usabilidade para pesquisa de usuários, levanta- mento de requisitos e metas de usabilidade, definição de design e avaliações com usuários e clientes.
- Força 2: Não existe garantia de que especialistas em usabilidade ou UX participarão voluntariamente do projeto e/ou não é possível contratá-los.
- Força 3: Desenvolvimento distribuído e participação esporádica de membros.

Solução: Em vez de caminhos paralelos entre equipe de desenvolvimento e de UX, como ocorre na prática Caminhos paralelos da comunidade de métodos ágeis, a equipe de desenvolvimento executa o ciclo completo de DCU para um conjunto específico de funcionalidades, utilizando-se de Pouco design antecipado ou Pouco design antecipado e distribuído para coletar informações. A prática pode ser executada apenas pela equipe-núcleo do projeto ou mesmo com a participação dos demais contribuidores que desejem participar.

Especialista-generalista

Contexto: Equipes de desenvolvimento de software livre, que não possuem especialistas em usabilidade ou UX como membros do time, compostas por desenvolvedores que desejam desenvolver sistemas com melhor usabilidade para usuários típicos. Problema: Ausência de especialistas em usabilidade ou UX na equipe de desenvolvimento do projeto. Principais forças envolvidas:

- Força 1: Não existe garantia de que especialistas em usabilidade ou UX participarão voluntariamente do projeto e/ou não é possível contratá-los.

- Força 2: Desenvolvimento distribuído e participação esporádica de membros.

Solução: Os desenvolvedores da equipe-núcleo do projeto aplicam práticas de usabilidade para entender quem são os usuários típicos do sistema, quais são as suas necessidades e em que contexto o sistema seria utilizado, de modo a incluir essas considerações nos requisitos da aplicação. As pesquisas têm baixa granularidade, ou seja, realiza-se apenas o necessário para o entendimento das funcionalidades da próxima iteração. Os requisitos podem ser definidos por meio da escrita de cartões de histórias de usuários, que são validados com o cliente conforme ocorre em comunidades de métodos ágeis. A documentação detalhada dos requisitos pode ser encontrada nos testes de aceitação, que podem ser acessados por qualquer desenvolvedor do sistema, conforme a prática Testes de aceitação de comunidades de métodos ágeis, o que mantém um relatório atualizado das funcionalidades do sistema que atendem ao comportamento esperado. As metas de usabilidade do sistema também podem ser descritas por meio da proposta de prática Definir metas de usabilidade automáticas.

Especificar contexto de uso

Pouco design antecipado e distribuído

Contexto: Equipe de desenvolvimento de software livre composta por desenvolvedores e que não possui especialistas em usabilidade ou UX como membros. Membros da equipe-núcleo do projeto e contribuidores encontram-se distribuídos em diversas localidades. Contudo, existe a necessidade de realização de pesquisas presenciais com usuários típicos para melhor compreensão do contexto de uso do sistema. Problema: Utilizar práticas de usabilidade, em ambiente de desenvolvimento de software livre, para especificar contexto de uso de um sistema, onde membros da equipe estão dispersos em vários locais diferentes. Principais forças envolvidas:

- Força 1: Distância física entre membros de uma comunidade de desenvolvimento de software livre.
- Força 2: Necessidade de realização de pesquisas de usabilidade para definição de perfil de usuários típicos e o contexto de uso do sistema.
- Força 3: Possibilitar a participação de voluntários de diversas culturas.

Solução: Equipe-núcleo do projeto é responsável por definir quais são as práticas de usabilidade a serem utilizadas para especificação do contexto de uso de um sistema e também por realizar as práticas presenciais na sua cidade. Membros da equipe, que se encontram dispersos em locais distintos, poderiam aplicar a mesma prática em sua localidade, de modo a obter feedback de usuários com culturas diferentes; por exemplo, replicando testes, sessões de grupos focais ou entrevistas presenciais, em sua região ou país. Desse modo,

possibilita-se a obtenção da percepção cultural de vários locais distintos, de modo a explorar o contexto de projetos abertos, no qual podem existir desenvolvedores, usuários, membros da equipe-núcleo e contribuidores em diversas localidades.

Especificar requisitos

Definir metas de usabilidade automáticas

Contexto: Desenvolvimento aberto, distribuído e colaborativo, onde desenvolvedores podem entrar e sair do projeto durante o processo de desenvolvimento. Também não existe uma equipe de usabilidade trabalhando em conjunto com a equipe de desenvolvimento. Problema: Definir metas de usabilidade de modo que todos os desenvolvedores que contribuam com um projeto aberto possam conhecer as metas definidas. Principais forças envolvidas:

- Força 1: Necessidade de definição de metas de usabilidade que atendam às necessidades de usuários típicos.
- Força 2: Possibilitar que todos os desenvolvedores tenham contato diário com as metas de usabilidade definidas
- Força 3: Desenvolvimento distribuído e participação esporádica de membros.
- Força 4: Manter documentação atualizada das metas de usabilidade tratadas pelo sistema.

Solução: Escrita de testes de aceitação automáticos baseados em Behaviour Driven Development (BDD) para definição de metas de usabilidade. Para o contexto de desenvolvimento livre, seria mais eficiente escrever as metas de usabilidade diretamente no ambiente de desenvolvimento do que em documentos separados, que correm o risco de não serem lidos. Sendo assim, conforme grupos de funcionalidades são selecionados para desenvolvimento, descreve-se as metas de usabilidade que precisam ser cumpridas para essas funcionalidades. Membros da equipe-núcleo do projeto podem escrever testes de aceitação automáticos, envolvendo usuários típicos e/ou clientes, o que possibilita documentar o comportamento esperado para a funcionalidade, e também gerar um relatório do funcionamento do sistema, exibindo quais funcionalidades e quais cenários são implementados de acordo com as necessidades dos usuários reais.

Avaliar designs

RITE (Rapid Iterative Testing and Evaluation) para desenvolvedores de software livre

Contexto: Equipes de desenvolvimento de software livre que não possuem especialistas em usabilidade ou UX como membros do time, mas que tem a necessidade de realizar

testes de usabilidade com usuários típicos do sistema de modo a desenvolver sistemas com melhor usabilidade. Problema: Possibilitar a identificação e correção de problemas de usabilidade no menor tempo possível durante o desenvolvimento de software livre. Principais forças envolvidas:

- Força 1: Diminuir a distância entre a identificação e a correção de problemas de usabilidade encontrados em testes com usuários.
- Força 2: Não existe garantia de que especialistas em usabilidade ou UX participarão voluntariamente do projeto e/ou não é possível contratá-los.

Solução: O método RITE pode ser aplicado por membros da equipe-núcleo do projeto, não sendo necessário utilizar laboratórios de usabilidade com a aplicação de testes formais. Os desenvolvedores da equipe-núcleo podem observar os usuários utilizando um pequeno conjunto de funcionalidades do sistema e solicitar que falem em voz alta o que estão pensando, enquanto o utilizam (Protocolo Pensando em voz alta). Não seria necessária a criação de relatórios e análises de vídeo dos testes, pois os desenvolvedores que estarão envolvidos na correção dos problemas encontrados podem participar do teste como moderadores ou observadores, de modo que possam obter o conhecimento das melhorias necessárias que precisam ser implementadas. Para documentar problemas referentes a um comportamento esperado do sistema, os testes de aceitação automáticos, utilizados pela comunidade de métodos ágeis, podem servir como forma de documentação, como também, para verificar se o sistema está realizando a tarefa do modo que se espera. Nesse caso, o relatório de teste de usabilidade seria substituído por testes de aceitação automáticos. A criação dos testes de aceitação, nesse caso, seria feita pelos próprios desenvolvedores que participaram do teste e conhecem o problema a ser resolvido. Um breve brainstorm após a sessão de teste serviria para consolidar as impressões dos membros da equipe envolvidos, possibilitando definir como os problemas serão corrigidos. A correção dos problemas encontrados seria realizada na sequência da realização do teste. Dessa forma, os testes de aceitação serviriam para registrar como corrigir um problema de usabilidade, detectado no teste com usuários típicos, para um determinado cenário de uso do sistema.

Assumindo as práticas descritas ao estudo de caso Mezuro, o ciclo de vida e aplicação destas na comunidade de software livre será seguido dentro do projeto, junto com a equipe de desenvolvimento, visando suprir as deficiências e problemas citados no capítulo 1.

4.4 Métodos de Avalição

Uma vez que conceituamos as práticas de usabilidade ágeis e o que encontramos sobre usabilidade em projetos de software livre, nesta seção detalharemos os métodos

escolhidos para a avaliação da ergonomia das interfaces. Os métodos de avaliação trás um diagnóstico através de verificações e inspeções de aspectos ergonômicos das interfaces que possam ser um problema ao usuário durante sua interação com o sistema. Através desse diagnóstico é possível priorizar e classicar os problemas encontrados de acordo com o método escolhido.

4.4.1 Avaliação heurísticas

Uma avaliação heurística representa um julgamento de valor sobre as qualidades ergonômicas das Interfaces Humano-Computador. Essa avaliação é realizada por especialistas em ergonomia, com base em sua experiência e competência no assunto ().

Para utilização de uma avaliação heurística serão definidos os graus de severidade. A severidade do problema de usabilidade é uma combinação de três fatores:

- A frequência com que ocorre o problema : é comum ou raro?
- O impacto do problema caso ocorra : Será que vai ser fácil ou difícil para os usuários a superar ?
- A persistência do problema: É um problema que com o tempo os usuários possam superar , uma vez que sabe sobre ele ou os usuários repetidamente serão incomodados pelo problema ?

A escala de classificação seguinte de 0 a 4 pode ser usada para avaliar a severidade dos problemas de usabilidade: ():

- 0 = Não há consenso quanto a ser um problema de usabilidade
- 1 = Problema cosmético
- 2 = Problema menor
- 3 = Problema importante de usabilidade
- 4 = Catástrofe de usabilidade

A presentação dos resultados seguirá um modelo simples similar ao que é utilizado em desenvolvimento ágil para documentação de defeitos, elencando o problema, a possível solução e o grau de severidade.

4.4.2 Inspeções por listas de verificação

As inspeções de ergonomia por meio de listas de verificação permitem que profissionais, não necessariamente especialistas em ergonomia, identifiquem problemas menores e repetitivos das interfaces. Nesse tipo de técnica, ao contrário das avaliações heurísticas, são mais as qualidades explicativas da ferramenta e menos os conhecimentos implícitos dos avaliadores que determinam as possibilidades para a avaliação (). Através das inspeções de ergonomia será possível suprir um deficit ocasionado pela falta de experiência do avaliador dentro de determinados contextos do sistema que este não esteja familiarizado. A ISO 9241 fornece listas de verificação de ergonomia bem definidas, porém será utilizado as listas do laboratório LabIUtil do projeto ErgoList [1], que fornece um serviço na Internet para aplicar aplicarmos uma avaliação simplificada e objetiva (*check-list*) e obtermos os resultados imediatamente. Com a aplicação da lista pode obter vantagens como obter conhecimentos ergonômicos, reduzir a subjetividade normalmente associada a processos de avaliação e sistematizar as avaliações se tratando de abrangência de componentes a inspecionar.

Com a aplicação dos métodos de avalição será obtido os relatórios com os diagnósticos da verificação e inspeção dos aspectos ergonômicos. Estes possibilitarão a classificação (de acordo com cada método) e a priorização dos problemas encontrados, fator fundamental para sustentar as práticas de usabilidade adotadas para a implementação no estudo de caso Mezuro descritas na seção 4.3. Ter o controle dos problemas levantados e seus impactos na aplicação, irá auxiliar na tomada de decisão da equipe e deixará todos cientes do andamento da usabilidade no projeto e sua importância dentro do desenvolvimento.

[1] http:labiutil.inf.ufsc.br/ergolist/check.htm

5 Estudo de Caso

5.1 Mezuro Plugin

5.2 Mezuro Standalone