

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-

50 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, 05 de dezembro de 2013:

**Prof. Dr. Paulo Roberto Miranda
Meirelles**
Orientador

Prof. Dr. André Barros de Sales
Convidado 1

Prof. Dr. Maurício Serrano
Convidado 2

Brasília, DF
2013

Resumo

Muitos dos projetos de software livre não possuem práticas para melhoria da usabilidade em seu processo de desenvolvimento, o que pode influenciar em sua adoção. O desenvolvimento de software livre pode ser considerado um método ágil do ponto de vista da engenharia de software. O objetivo deste trabalho foi aplicar tarefas de usabilidades através de técnicas de usabilidade ágeis em comunidades de software livre. Para isso, foi empregado uma revisão sistemática e bibliográfica para levantamento da usabilidade em software livre e um estudo de caso a fim de utilizá-las em um contexto real levantado assim os resultados através de métodos de avaliação.

Palavras-chaves: usabilidade, software livre, técnicas de usabilidade, métodos ágeis.

Abstract

Many free software projects have no practices to improve usability in their development process, which may influences on the adoption of them. The free software production can be considered an agile methodology from the software engineering point of view. The objective of this work was to apply usability tasks through agile usability techniques in open source communities. For this, we used a systematic literature review and survey for usability in open source software and a case study in order to use them in a real context so raised by the results of evaluation methods.

Key-words: usability, free software, usability techniques, agile methods.

Lista de ilustrações

Figura 1 – Ciclo de atividades de usabilidade	28
Figura 2 – Ciclo das tarefas de usabilidade x desenvolvimento	29
Figura 3 – Técnicas de usabilidade baseado no DCU	29
Figura 4 – Interação Mezuro e Kalibro Metrics	42
Figura 5 – Protótipo de tela: Login	45
Figura 6 – Protótipo de tela: Signup	45
Figura 7 – Protótipo de tela: New Repository	46
Figura 8 – Protótipo de tela: New Protótipo	46

Lista de tabelas

Tabela 1 – Conjunto integrador de critérios, princípios, regras e heurísticas de ergonomia	27
Tabela 2 – Mecanismos de usabilidade	30
Tabela 3 – Mapeamento entre os mecanismos de usabilidade e ações	31
Tabela 4 – Práticas de usabilidade no contexto de Software Livre	32
Tabela 5 – Cronograma para atividades do TCC2	48

Lista de abreviaturas e siglas

UnB	Universidade de Brasília
FGA	Faculdade Gama
TCC	Trabalho de Conclusão de Curso
DCU	Design Centrado no Usuário
DSDM	Dynamic software development method
XP	Extreme programming
ISO	International Organization for Standardization
RITE	Rapid Iterative Testing and Evaluation
UX	User eXperience
AGPL	GNU Affero General Public License
LGPL	GNU Lesser General Public License
QualiPSO	Quality Platform for Open Source
MVC	Model View Controller
Rails	Ruby on Rails
SOAP	Simple Object Access Protocol
JSON	JavaScript Object Notation

Sumário

1	Introdução	17
1.1	Objetivos	17
1.1.1	Objetivos Gerais	18
1.1.2	Específicos	18
1.2	Organização do Trabalho	18
2	Software Livre	19
3	Métodos Ágeis	21
3.1	Metodologia Ágil	21
3.2	Princípios Ágeis	22
4	Usabilidade	25
4.1	Terminologias	25
4.2	Técnicas de Usabilidade Ágeis	27
4.3	Usabilidade em Software Livre	31
4.4	Métodos de Avaliação	36
4.4.1	Avaliação heurísticas	36
4.4.2	Inspecões por listas de verificação	37
5	Estudo de Caso: projeto Mezuro	39
5.1	Mezuro	40
5.1.1	Mezuro Plugin	41
5.1.2	Mezuro Standalone	42
6	Considerações Finais	47
6.1	Cronograma	47
	Referências	49

1 Introdução

Ao lado da própria dificuldade em se mensurar objetivamente a usabilidade de um sistema, é comum em programas de software livre que haja pouco incentivo (ou interesse) nesse aspecto, dado que a prioridade do mesmo é a implementação das funcionalidades. Essa cultura leva o desenvolvedor a iniciar o projeto pelo código, deixando o design de interfaces em segundo plano ([THOMAS, 2008](#))

Métodos de usabilidade são utilizados por empresas especialistas em projeto de interação e, nos processos de desenvolvimento de projetos de software fechado, o que garante, em geral, uma melhor usabilidade quando comparados à maioria dos sistemas de software livre.

Esse cenário é um dos fatores que limita a expansão de uso e aceitação de sistemas livres, que com baixa usabilidade, perde-se também em confiança dos usuários. O uso desse tipo de sistema limita-se a usuários mais experientes em computação, o que resulta em perdas para usuários inexperientes. Muitos sistemas de software livre possuem código de qualidade, o que lhes confere algoritmos eficientes e de bom desempenho, pois foram produzidos, na maioria das vezes, por indivíduos motivados a resolver desafios ou problemas no sistema. Santos ([2012](#)) ressalta que a perda de usuários potenciais devido à baixa usabilidade configura, portanto, um desperdício de recursos para a sociedade.

Já Nichols e Twidale ([2006](#)) afirmam que desenvolvedores de software livre não costumam notar a baixa usabilidade de diversos sistemas, por serem geralmente usuários experientes e acostumados a interfaces de baixo nível como a linha de comando, e em contrapartida, empresas que produzem software comercial muitas vezes possuem funcionários especialistas em usabilidade, raro em equipes de desenvolvimento de software livre.

Apesar desse cenário, o desenvolvimento de software livre vem mudando nos últimos anos com a crescente preocupação com a usabilidade, ainda que essa preocupação esteja normalmente limitada a projetos de grande visibilidade, geralmente patrocinados por grandes empresas. Porém, no âmbito das comunidades, a usabilidade raramente é considerada no processo ([NICHOLS; TWIDALE, 2006](#)).

1.1 Objetivos

Esta seção apresenta os objetivos gerais e específicos deste TCC.

1.1.1 Objetivos Gerais

Neste trabalho de conclusão de curso, temos o objetivo de implementar tarefas de usabilidade dentro do ciclo de desenvolvimento de um projeto de software livre, mantendo o responsável pela usabilidade inserido na equipe como implementador e ao final levantar os resultados da aplicação da usabilidade através de métodos de avaliação.

1.1.2 Específicos

Os objetivos específicos desse trabalho são:

1. Identificar as técnicas de usabilidade ágeis que possam ser utilizadas na comunidade de software livre.
2. Detalhar e definir uma técnica de usabilidade a ser aplicada.
3. Definir interação no projeto com a equipe de desenvolvimento.
4. Implementar práticas de usabilidade.
5. Avaliar e apresentar resultados da usabilidade do projeto para a equipe.

1.2 Organização do Trabalho

No Capítulo 2 encontra-se uma definição de software livre, suas liberdades e principais características; no Capítulo 3 uma breve descrição resumida dos seus princípios, práticas, principais métodos e sobre o manifesto ágil; no Capítulo 4 uma revisão bibliográfica sobre usabilidade, seus aspectos sobre a visão dos principais autores e terminologias associadas, após foi realizado uma revisão sistemática para levantar técnicas de usabilidade ágeis que possam ser aplicadas em comunidades de software livre e escolhida uma para detalhamento e uso no trabalho proposto, além da apresentação dos métodos de avaliação que serão utilizados para avaliar os resultados da aplicação da usabilidade; no Capítulo 5 é apresentado o estudo de caso, a ferramenta Mezuro que se encontra em um segundo ciclo de desenvolvimento voltado para uma plataforma standalone que possibilita a implementação das tarefas de usabilidade.

2 Software Livre

O software, em um sistema computacional, é o componente que contém o conhecimento relacionado aos problemas a que a computação se aplica, contendo diversos aspectos que ultrapassam questões técnicas ([MEIRELLES, 2013](#)), por exemplo:

- O processo de desenvolvimento do software;
- Os mecanismos econômicos (gerenciais, competitivos, sociais, cognitivos etc.) que regem esse desenvolvimento e seu uso;
- O relacionamento entre desenvolvedores, fornecedores e usuários do software;
- Os aspectos éticos e legais relacionados ao software

O que define e diferencia o software livre de software proprietário vai do entendimento desses quatro pontos dentro do que é conhecido como *ecossistema do software livre* ([MEIRELLES, 2013](#)). O princípio básico desse ecossistema refere à liberdade dos usuários de executar, copiar, distribuir, estudar, mudar e melhorar o software. Estas estão definidas nas quatro liberdades para os usuários do software:

- A liberdade de executar o programa, para qualquer propósito (liberdade no. 0);
- A liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades (liberdade no. 1). Acesso ao código-fonte é um pré-requisito para esta liberdade;
- A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade no. 2);
- A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie (liberdade no. 3). Acesso ao código-fonte é um pré-requisito para esta liberdade.

Um programa é software livre se os usuários têm todas essas liberdades. Assim, você deve ser livre para redistribuir cópias, seja com ou sem modificações, seja de graça ou cobrando uma taxa pela distribuição, para qualquer um em qualquer lugar.

Você deve também ter a liberdade de fazer modificações e usá-las privativamente no seu trabalho ou lazer, sem nem mesmo mencionar que elas existem. Se você publicar as modificações, você não deve ser obrigado a avisar a ninguém em particular, ou de qualquer forma particular.

A liberdade de executar o programa significa a liberdade para qualquer tipo de pessoa física ou jurídica utilizar o software em qualquer tipo de sistema computacional, para qualquer tipo de trabalho em geral e finalidade, sem que seja necessário comunicar sobre isso com o desenvolvedor ou a qualquer outra entidade em especial.

A liberdade de redistribuir cópias deve incluir formas binárias ou executáveis do programa, assim como o código-fonte, tanto para as versões modificadas.

A liberdade de acesso ao código-fonte do programa. Portanto, acesso ao código-fonte é uma condição necessária ao software livre (SYSTEM, 2013).

Essa liberdade dentro do software livre é dita por Richard Stallman como a principal vantagem. Software não-livre é ruim porque nega sua liberdade. Então, perguntar sobre as vantagens práticas do software livre é como perguntar sobre as vantagens práticas de não estar algemado (STALLMAN; GAY, 2009).

O GNU quis dar aos utilizadores a liberdade, não só para ser popular. Contudo precisava usar termos de distribuição que impediriam software livre de ser transformado em software proprietário. O método usado foi chamado de copyleft. Copyleft é um método geral para fazer um software livre e exige que todas as versões modificadas e estendidas do programa sejam também. Ele utiliza lei de direitos autorais, mas veio para servir como oposto de sua finalidade usual: ao invés de um meio de privatizar o software, torna-se um meio de manter o software livre (STALLMAN; GAY, 2009)

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 (SCHWABER; BEEDLE, 2002), 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 (ÁGIL, 2013) Os conceitos chave do Manifesto Ágil são: **Indivíduos e interações** ao invés de processos e ferramentas. **Software executável** ao invés de documentação. **Colaboração do cliente** ao invés de negociação de contratos. **Respostas rápidas a mudanças** ao invés de seguir planos. 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 Dyba (2008) em uma revisão sistemática elenca alguns dos principais métodos ágeis presente em 2008 e trás uma breve descrição de cada um, o que ajuda a observar características das comunidades ágeis já que os principais especialistas desses métodos ajudaram a escrever o manifesto ágil.

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 planeamento 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) concentra-se nas melhores práticas para o desenvolvimento. Consiste em doze práticas: o jogo de planeamento, 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".

3.2 Princípios Ágeis

O manifesto ágil define 12 princípios que devem ser seguidos ([ÁGIL, 2013](#))

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 adequam 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 manifesto ágil.

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 ([SYSTEMS...](#), 2010).

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 ([CYBIS](#); [BETIOL](#); [FAUST](#), 2010).

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 ([LIDA](#), 2005).

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 ([ERGONOMIA](#), 2013).

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 (NIELSEN, 1994):
 - 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” (SHNEIDERMAN; BEN, 2003)
 - 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 (BASTIEN; SCAPIN et al., 1993)
 - 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 apresentado na Tabela 1 (CYBIS; BETIOL; FAUST, 2010).

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 1 – Conjunto integrador de critérios, princípios, regras e heurísticas de ergonomia

4.2 Técnicas de Usabilidade Ágeis

Técnicas de usabilidade e desenvolvimento ágil têm muito em comum, principalmente o fato de que, muitas vezes, estão envolvidos no desenvolvimento do mesmo software. Apesar disso, tem havido pouca investigação ou discussão sobre a forma como os dois processos trabalham em conjunto e os resultados dessa parceria. (FERREIRA; NOBLE; BIDDLE, 2007).

Para tanto realizou-se uma revisão sistemática. A revisão sistemática é uma forma de síntese das informações disponíveis em dado momento, sobre um problema específico, de forma objetiva e reproduzível, por meio de método científico. Ela tem como princípios gerais a exaustão na busca dos estudos analisados, a seleção justificada dos estudos por critérios de inclusão e exclusão explícitos e a avaliação da qualidade metodológica, bem como a quantificação do efeito dos tratamentos por meio de técnicas estatísticas. (LIMAA; SOARES; BACALTCHUK, 2000)

O Objetivo da revisão sistemática era levantar as técnicas de usabilidade em projetos ágeis aplicadas no desenvolvimento de software livre. A busca foi realizada nas bibliotecas digitais Google Academic e Springer Link que possuem máquinas de busca com bom funcionamento e abrangência, além das bases de teses e dissertações da UnB e USP. Como resultado do trabalho levantou-se uma lista de técnicas e selecionadas aquelas que poderiam ser mais facilmente aplicadas em projetos de software livre.

Santos e Kon (2009) propõe a aplicação do próprio processo de DCU (Design Centrado no Usuário) para levantamento dos usuários e do contexto de uso da metodologia. Os usuários do processo são membros de equipes de software livre, que desejam inserir práticas de usabilidade no processo de desenvolvimento. O contexto de uso são sistemas distribuídos geograficamente, com pessoas trabalhando colaborativamente. Os métodos podem ser aplicados tanto para o início de novos projetos, que tenham como usuários, pessoas não acostumadas a sistemas livres ou mesmo para a melhoria de sistemas existentes, com respeito à usabilidade.

Já Lee, Judge e McCrickard (2011) defendem a aplicação de definição de histórias de usuário de usabilidade dirigidas pela decisão do responsável por design realizando a prototipação dessas histórias e depois sendo validadas através de testes de usabilidade conforme Figura 1.

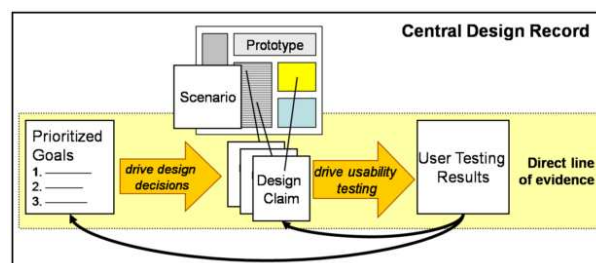


Figura 1 – Ciclo de atividades de usabilidade

As atividades de usabilidade seguem o modelo de ciclo de vida das outras atividades dentro do desenvolvimento ágil com a diferença que o ciclo de usabilidade tem sua iteração primeiro que as outras atividades de desenvolvimento conforme Figura 2:

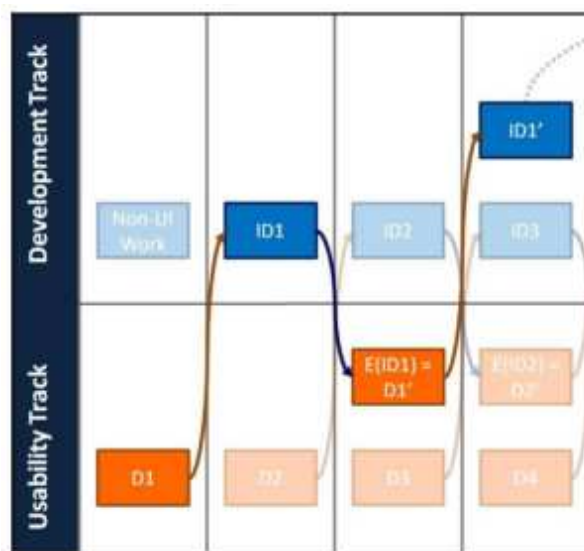


Figura 2 – Ciclo das tarefas de usabilidade x desenvolvimento

Santos (2012) em uma nova pesquisa aplicada propôs a aplicação das técnicas de usabilidade de acordo com as fases do DCU, conforme a descrição das técnicas de usabilidade das comunidades de métodos ágeis e de software livre. Para a fase do DCU Criar soluções de design, não foram identificadas propostas de adaptações, sendo portanto, descritas propostas para as fases: Identificar necessidades para design centrado em humano, Especificar contexto de uso, Especificar requisitos e Avaliar Designs conforme Figura 3.

Práticas Propostas	
Fases do 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 para desenvolvedores de software livre

Figura 3 – Técnicas de usabilidade baseado no DCU

Moreno e Yagüe (2012) trás em seu trabalho a inserção da usabilidade por meio da criação de User Stories de usabilidade que denominam de *Usability Stories*. Tendo que especificar os recursos funcionais de usabilidade que serão utilizados no projeto, o artigo trás um tabela que é necessária para definir as características dessa funcionalidade.

Status do Sistema	Para informar os usuários sobre a situação interna do sistema
Aviso	Para informar os usuários de qualquer ação com consequências importantes
Longa ação Feedback	Para informar aos usuários que o sistema está processando uma ação que vai demorar algum tempo para completar
Desfazer global	Para desfazer as ações do sistema em vários níveis
Abortar Operação	Para cancelar a execução de uma ação ou toda a aplicação
Abortar comando	Para cancelar a execução de uma tarefa em andamento
Voltar	Para voltar a um estado particular em uma sequência de execução de comando
Entrada de texto estruturada	Para ajudar a prevenir o usuário de cometer erros de entrada de dados
Execução Passo-a-Passo	Para ajudar os usuários a realizar tarefas que requerem diferentes passos com a entrada do usuário e tal entrada correta
Preferências	Para gravar as opções de cada usuário para usar as funções do sistema
Favoritos	Para gravar determinados locais de interesse para o usuário
Ajuda Multinível	Para fornecer diferentes níveis de ajuda para usuários diferentes

Tabela 2 – Mecanismos de usabilidade

Associado a essas características é definido três maneiras pelas quais a incorporação de usabilidade influencia as histórias de usuário.

1. A adição de novas histórias para representar requisitos diretamente derivados usabilidade.
2. Adição ou modificação de tarefas em histórias de usuários existentes. Isto significa que algumas ações decorrentes de limitações de usabilidade devem ser realizadas por um usuário existente na história. Esta tarefa pode ser tão simples ou detalhados, conforme necessário.
3. Adição ou modificação de critérios de aceitação. Estes critérios de aceitação aparecem porque a funcionalidade da história de usuário precisa incluir algumas ações específicas para modificar o ambiente operacional.

Com isso os autores realizaram um mapeamento entre os mecanismos de usabilidade e as ações a serem tomadas finalizando a proposta de como a usabilidade deve ser abordada por equipes ágeis.

	Nova Tarefa	Modificar Tarefa	Novo Critério de Aceitação	Modificar Critério de Aceitação	Nova História de Usabilidade	Nova História de Usuário
Status do Sistema		X	X	X	X	
Aviso	X		X	X	X	
Longa ação		X	X			X
Abortar Operação		X	X			
Abortar comando	X	X	X			
Voltar	X	X	X			
Entrada de texto estruturada		X	X			
Execução Passo-a-Passo	X		X	X		
Preferências						X
Favoritos		X	X		X	
Ajuda Multinível		X	X		X	

Tabela 3 – Mapeamento entre os mecanismos de usabilidade e ações

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. (SANTOS, 2012)

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 4.

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 4 – Práticas de usabilidade no contexto de Software Livre

As práticas são descritas por ela 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, levantamento 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 Avaliação

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 têm 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 classificar 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 especialis-

tas em ergonomia, com base em sua experiência e competência no assunto (CYBIS; BETIOL; FAUST, 2010).

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: (NIELSEN, 1995):

- 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 apresentação 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 (CYBIS; BETIOL; FAUST, 2010).

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 avaliação 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 Mezero 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: projeto Mezuro

O trabalho de pesquisa desenvolvido durante o capítulo 4, onde se levantou técnicas de usabilidade ágeis que poderiam ser aplicadas na comunidade de software livre conforme seção 4.2 e depois a seleção de uma dessas técnicas a fim de realizar sua descrição e detalhamento na seção 4.3 tinha como objetivo definir a forma que seria integrado a usabilidade em uma equipe de desenvolvimento em um contexto de software livre. Detalhar essa forma de trabalho se faz necessário, pois esses conceitos e métodos serão apresentados e aplicados em uma ferramenta real que será utilizada como estudo de caso.

YIN (2009) diz que "o estudo de caso é uma inquirição empírica que investiga um fenômeno contemporâneo dentro de um contexto da vida real, quando a fronteira entre o fenômeno e o contexto não é claramente evidente e onde múltiplas fontes de evidência são utilizadas". Validar a pesquisa através de sua aplicação trás vantagens como:

- Produção de artefatos descritivos suficientemente rico para permitir reinterpretações subsequentes.
- Relacionamento da teoria com a prática
- Melhor percepção através de exemplos específicos, acontecimentos ou experiências

Segundo YIN (2009), para um estudo de caso deve se definir:

- Uma visão geral do projeto do estudo de caso
- Os procedimentos que serão aplicados, abordados
- As questões do estudo de caso que o investigador deve ter em mente, os locais, as fontes de informação, os formulários para o registro dos dados e as potenciais fontes de informação para cada questão
- Um guia para o relatório do Estudo do Caso

A definição dessa estrutura se dá de forma automática devido ao modelo adotado de abordagem da usabilidade dentro do projeto facilitando a aplicação do método.

A ferramenta definida para o estudo de caso foi o software livre Mezuro, uma plataforma de monitoramento de código-fonte. A escolha desta ferramenta se deve ao fato que, ao ser concebida, um trabalho similar foi realizado pela pesquisadora Ana Paula Oliveira dos Santos, porém a ferramenta se tratava de um plugin de outra plataforma e tornou-se necessário uma migração para uma ferramenta standalone. Com essa troca

muito foram as mudanças e os aspectos de usabilidade passaram a necessitar de um novo planejamento. Outro fator importante foi a facilidade de estar inserido junto a equipe de desenvolvimento, ficando assim mais próximo do projeto.

5.1 Mezuro

A plataforma Mezuro foi concebida com o intuito de comparar projetos e métricas técnicas, ensinar como usar métricas através de configurações, analisar o código, evitar dívidas técnicas e divulgar o uso de métricas de código.

O contexto da plataforma é descrito de forma resumida pelos seu desenvolvedores da seguinte forma:

Para desenvolvedores de software

que precisam de um melhor entendimento sobre o seu próprio código

o Mezuro

é uma rede social de nicho

que fornece interpretação métrica colaborativa.

Ao contrário CodeClimate

nosso projeto atende várias linguagens, tem configurações abertas para avaliações de código e tudo isso de graça

De forma técnica o Mezuro é uma plataforma para analisar e compreender as métricas, bem como, aplicar os conceitos de código limpo, medição de software e atratividade do software livre. Ele é um software livre sob licença AGPL ¹. Mezuro usa Kalibro Metrics , que é um serviço web que pode se conectar a diferentes tipos de repositórios de código-fonte , baixar projetos de software e executar muitos coletor métrica integrada e ferramentas de calculadora automaticamente. Kalibro é licenciado LGPL ² (MEIRELLES et al., 2012). É uma aplicação Web para que os líderes e desenvolvedores de projetos de software livre possam monitorar características de código-fonte, através de métricas de acoplamento, coesão, tamanho, encapsulamento, entre outras. Isso proporciona um acompanhamento do quanto o software está crescendo e se tornando mais complexo em relação a ele mesmo e à média dos projetos avaliados pelo Mezuro (SANTOS, 2012) .

A plataforma Mezuro foi desenvolvida no contexto do projeto QualiPSO (*Trust and Quality in Open Source Systems*). O projeto integrado Qualipso (Quality Platform for

¹ GNU Affero General Public License (AGPL): <gnu.org/licenses/agpl.html>

² GNU Lesser General Public License (LGPL):<gnu.org/licenses/lgpl.html>

Open Source) busca definir e implementar tecnologias, procedimentos, leis e políticas com o objetivo de potencializar as práticas de desenvolvimento de software livre, tornando-as confiáveis, reconhecidas e estabelecidas na indústria. Para viabilizar o projeto e a sustentação do software livre como uma solução confiável para a indústria, criou-se um consórcio formado por colaboradores de diferentes origens: França, Itália, Brasil, Espanha, China, Alemanha e Escócia ([QUALIPSO, 2013](#)) .

Para o desenvolvimento do Mezuro realizou-se um estudo de plataformas correlatas a fim de melhor situar as resoluções já existentes ([MEIRELLES, 2013](#)). As ferramentas estudadas foram:

FLOSSMetrics (Free /Libre Open Source Software Metrics) é um projeto que utiliza metodologias e ferramentas existentes para fornecer um grande banco de dados com informações sobre o desenvolvimento de software livre

Ohloh é uma plataforma que oferece um conjunto de serviços web e um sistema web para comunidade de software livre, que visa prover uma visão geral de evolução dos projetos de software livre em desenvolvimento

Qualloss (Quality in Open Source Software) é uma metodologia para automatizar a medição da qualidade de projetos de software livre, usando ferramentas para analisar o código-fonte e as informações dos repositórios dos projetos

SQO-OSS (Software Quality Assessment of Open Source Software) fornece um conjunto de ferramentas para análise e avaliação comparativa de projetos de software livre

QSOS é uma metodologia baseada em quatro etapas: definição de referência utilizada; avaliação de software; qualificação dos usuários em contexto específico; seleção e comparação de software

FOSSology (Advancing open source analysis and development) é um projeto que fornece um banco de dados gratuito com informações sobre licenças de software livre

HackyStat é um ambiente para visualização, análise e interpretação do processo de desenvolvimento de software e dados do produto de software

5.1.1 Mezuro Plugin

A primeira versão da ferramenta foi desenvolvida sendo uma instância da plataforma de rede social Noosfero 6 com o plugin Mezuro ativado. Noosfero é um software livre (plataforma web) para redes sociais que possui as funcionalidades de Blog, e-Portfolios, RSS, discussão temática e agenda de eventos em um único sistema. Ele foi desenvolvido

em linguagem de programação Ruby usando o framework Rails. Ruby on Rails é baseado em arquitetura Model View Controller (MVC). Plugins Rails (como Mezuro plugin) é uma extensão do quadro principal, assim, a arquitetura Mezuro também foi baseada em MVC, seguindo Rails e a arquitetura do Noosfero (MEIRELLES et al., 2012).

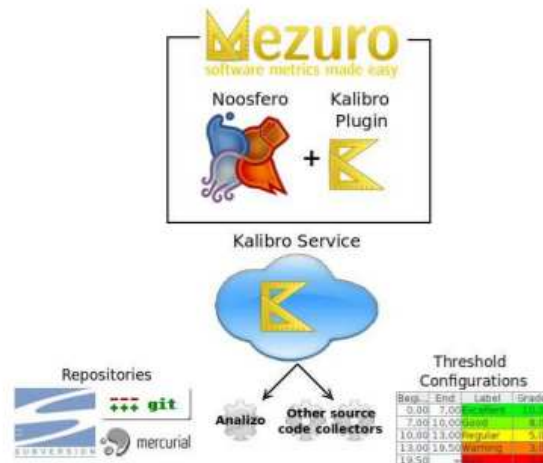


Figura 4 – Interação Mezuro e Kalibro Metrics

A utilização de plugins permitem que os desenvolvedores possam resolver vários problemas sem interagir com a base de código inteira. O Mezuro foi o primeiro plugin do Noosfero, sua estrutura foi construída sobre um paradigma baseado em eventos. O Noosfero dispara um evento em um determinado ponto e todos os plugins interessados nesse evento são capazes de agir. O Mezuro se comunica com o Kalibro Métricas Web Service, como mostrado na Figura 4, via solicitações SOAP, que prove o acesso aos seguintes end-points do Kalibro Services: Kalibro, Project, ProjectResult, ModuleResult, BaseTool, Configuration, e MetricConfiguration. Do ponto de vista conceitual, cada end-point Kalibro é um serviço do Mezuro que funciona como um orquestrador dos sete serviços, uma vez que cada um trabalha de forma independente e tem um conjunto diferente de funcionalidades. O Mezuro chama cada end-point (serviço) de acordo com suas ações (pedidos) para o Kalibro. Em suma, o desempenho e a escalabilidade do Mezuro dependem desta "orquestração". (MEIRELLES et al., 2012)

5.1.2 Mezuro Standalone

O mezuro standalone surgiu da necessidade de portar o mezuro plugin para uma plataforma independente do Noosfero. Isso ocorreu devido à limitações ocasionadas pelo Noosfero. Uma dessas limitações deve-se ao fato de seu desenvolvimento ser baseado em uma versão do antiga do rails com pouco suporte pela comunidade e limitado as

funções dessa versão. O mezuro standalone é baseado no framework rails 4.0, utiliza-se Bootstrap para o desenvolvimento de interface (front-end) e permanece com o protocolo de comunicação SOAP entre o mezuro e o kalibro conforme descrito na seção anterior, porém estuda-se uma possível mudança para JSON. O mezuro standalone é um novo ciclo de desenvolvimento da plataforma Mezuro. Este encontra-se em fase de criação o que permite aplicar a usabilidade dentro do ciclo de vida do projeto juntamente com a equipe de desenvolvimento.

Seguindo as técnicas de usabilidade adotadas, descrita na seção 4.3, as tarefas de usabilidade a serem realizadas no projeto Mezuro serão:

- Coletar requisitos de tarefas e necessidades dos usuários
- Criar protótipos de tela
- Testar protótipos com avaliações heurísticas
- Testar a usabilidade de protótipos com usuários locais e remotos (manipulação de variáveis independentes)
- Coletar métricas (variáveis dependentes)
- Apresentar resultados das análises dos testes (tracking)
- Definição de histórias de usabilidade baseadas nos requisitos fornecidos pelos clientes e no resultado dos testes de us
- Implementação das histórias definidas

Estes artefatos gerados levam em consideração os seguintes aspectos de usabilidade (INFORMÁTICA, 2013):

Presteza verifica se o sistema informa e conduz o usuário durante a interação.

Agrupamento por localização Verifica se a distribuição espacial dos itens traduz as relações entre as informações.

Agrupamento por formato Verifica os formatos dos itens como meio de transmitir associações e diferenças.

Feedback Avalia a qualidade do feedback imediato às ações do usuário.

Legibilidade Verifica a legibilidade das informações apresentadas nas telas do sistema.

Concisão Verifica o tamanho dos códigos e termos apresentados e introduzidos no sistema.

Ações Mínimas Verifica a extensão dos diálogos estabelecidos para a realização dos objetivos do usuário.

Densidade Informacional Avalia a densidade informacional das telas apresentadas pelo sistema.

Ações Explícitas Verifica se é o usuário quem comanda explicitamente as ações do sistema.

Controle do Usuário Avalia as possibilidades do usuário controlar o encadeamento e a realização das ações.

Flexibilidade Verifica se o sistema permite personalizar as apresentações e os diálogos.

Experiência do Usuário Avalia se usuários com diferentes níveis de experiência têm iguais possibilidades de obter sucesso em seus objetivos.

Proteção contra erros Verifica se o sistema oferece as oportunidades para o usuário prevenir eventuais erros.

Mensagens de erro Avalia a qualidade das mensagens de erro enviadas aos usuários em dificuldades.

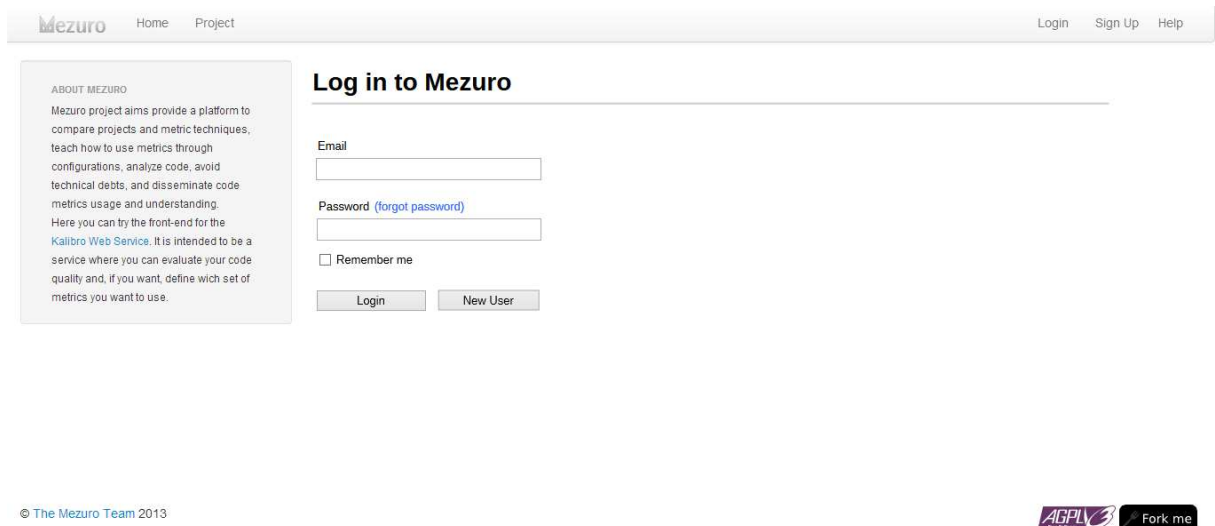
Correção de erros Verifica as facilidades oferecidas para que o usuário possa corrigir os erros cometidos.

Consistência Avalia se é mantida uma coerência no projeto de códigos, telas e diálogos com o usuário.

Significados Avalia se os códigos e denominações são claros e significativos para os usuários do sistema.

Compatibilidade Verifica a compatibilidade do sistema com as expectativas e necessidades do usuário em sua tarefa.

Desses artefatos um deles são os protótipos de telas que visam instruir a equipe de desenvolvimento. Nesses protótipos são inserido as considerações da equipe, requisitos do projeto, experiência do responsável pela usabilidade e alguns dos aspectos apresentados. Os protótipos desenvolvidos e apresentados para a equipe de uma primeira feature foram:



The image shows a web page prototype for the 'Mezuro' application. At the top is a navigation bar with the 'Mezuro' logo, 'Home', and 'Project' links on the left, and 'Login', 'Sign Up', and 'Help' links on the right. The main content area is divided into two sections. On the left, under the heading 'ABOUT MEZURO', there is a paragraph describing the project's goal: to provide a platform for comparing projects and metric techniques, teaching how to use metrics through configurations, analyzing code, avoiding technical debts, and disseminating code metrics usage and understanding. It also mentions a front-end for the 'Kalibro Web Service' intended to be a service for evaluating code quality and defining a set of metrics. On the right, the 'Log in to Mezuro' section contains a form with fields for 'Email' and 'Password' (with a 'forgot password' link). Below the password field is a 'Remember me' checkbox. At the bottom of the form are two buttons: 'Login' and 'New User'. The footer includes a copyright notice '© The Mezuro Team 2013' on the left and a logo with the text 'AGPLv3' and 'Fork me' on the right.

Mezuro Home Project Login Sign Up Help

Log in to Mezuro

ABOUT MEZURO

Mezuro project aims provide a platform to compare projects and metric techniques, teach how to use metrics through configurations, analyze code, avoid technical debts, and disseminate code metrics usage and understanding. Here you can try the front-end for the [Kalibro Web Service](#). It is intended to be a service where you can evaluate your code quality and, if you want, define wich set of metrics you want to use.

Email

Password ([forgot password](#))

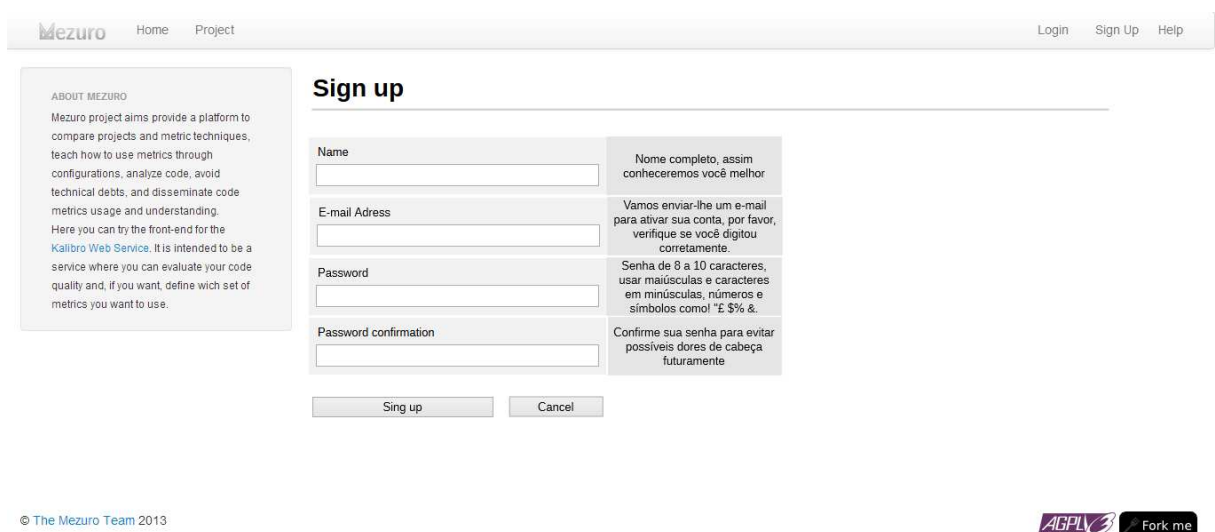
☐ Remember me

Login New User

© The Mezuro Team 2013

AGPLv3 Fork me

Figura 5 – Protótipo de tela: Login



The image shows a web page prototype for the 'Mezuro' application, specifically the 'Sign up' page. The layout is similar to the login page, with a navigation bar at the top. The main content area features an 'ABOUT MEZURO' section on the left and a 'Sign up' section on the right. The 'Sign up' section contains a form with four fields: 'Name', 'E-mail Address', 'Password', and 'Password confirmation'. Each field has a corresponding instruction: 'Nome completo, assim conheceremos você melhor' for Name; 'Vamos enviar-lhe um e-mail para ativar sua conta, por favor, verifique se você digitou corretamente.' for E-mail Address; 'Senha de 8 a 10 caracteres, usar maiúsculas e caracteres em minúsculas, números e símbolos como! *£ \$% &.' for Password; and 'Confirme sua senha para evitar possíveis dores de cabeça futuramente' for Password confirmation. Below the form are two buttons: 'Sing up' and 'Cancel'. The footer is identical to the login page, with a copyright notice and a logo.

Mezuro Home Project Login Sign Up Help

Sign up

ABOUT MEZURO

Mezuro project aims provide a platform to compare projects and metric techniques, teach how to use metrics through configurations, analyze code, avoid technical debts, and disseminate code metrics usage and understanding. Here you can try the front-end for the [Kalibro Web Service](#). It is intended to be a service where you can evaluate your code quality and, if you want, define wich set of metrics you want to use.

Name

E-mail Address

Password

Password confirmation

Sing up Cancel

© The Mezuro Team 2013

AGPLv3 Fork me

Figura 6 – Protótipo de tela: Signup

Mezuro

HomeProject

LoginSign UpHelp

ABOUT MEZURO

Mezuro project aims provide a platform to compare projects and metric techniques, teach how to use metrics through configurations, analyze code, avoid technical debts, and disseminate code metrics usage and understanding. Here you can try the front-end for the [Kalibro Web Service](#). It is intended to be a service where you can evaluate your code quality and, if you want, define wich set of metrics you want to use.

New Repository

Name	Padrão para nome do repositório
Type	Tipo do repositório do projeto
Address	Coloque aqui onde se encontra seu repositório. URL
Configuration	Configuração

SaveBack

© The Mezuro Team 2013


 Fork me

Figura 7 – Protótipo de tela: New Repository

Mezuro

HomeProject

LoginSign UpHelp

ABOUT MEZURO

Mezuro project aims provide a platform to compare projects and metric techniques, teach how to use metrics through configurations, analyze code, avoid technical debts, and disseminate code metrics usage and understanding. Here you can try the front-end for the [Kalibro Web Service](#). It is intended to be a service where you can evaluate your code quality and, if you want, define wich set of metrics you want to use.

New Project

Name	Formato e padrão para nome do projeto
Description	Descreva brevemente o projeto de preferência seguindo a estrutura: Finalidade, Funcionalidades, Licença

SaveCancel

© The Mezuro Team 2013


 Fork me

Figura 8 – Protótipo de tela: New Protótipo

6 Considerações Finais

A evolução do Mezuro não foi motivada por um motivo isolado. Um conjunto de fatores influenciaram e convenceram a equipe que o melhor para o futuro do projeto Mezuro seria um conjunto de modificações em sua estrutura. Os desenvolvedores estavam restritos aos ultrapassados recursos do Rails 2 e do Ruby 1.8, a qual não recebia mais suporte de seus desenvolvedores.

A equipe almejava por liberdade na tomada de decisões, como por exemplo atualizar o Mezuro para as novas versões do Rails e do Ruby, aproveitando suas melhorias e novos recursos. Porém, estavam subordinados as decisões e andamento do projeto Noosfero. A equipe se deu conta que os recursos relacionados a redes sociais fornecidos não eram necessários para uma ferramenta de monitoramento de código-fonte. Além disso, a manutenibilidade e inserção de novos desenvolvedores ao projeto eram tarefas difíceis, já que o Mezuro crescia bastante e se tornava cada vez mais uma aplicação dentro de outra aplicação, ao invés de um plugin com funcionalidades bem específicas.

Levando em conta todos esses fatores, a equipe do Mezuro decidiu retirá-lo do Noosfero, transformando-a em uma aplicação independente, além de formatar uma comunidade de software livre para atrair novos desenvolvedores. Para consolidar esses fatores, que levaram a evolução do Mezuro, foi elaborado um questionário (encontrado no Apêndice ??) direcionado a equipe que o mantém. Dos fatores que motivaram a evolução, os que mais foram citados pela equipe foram aqueles que limitam sua liberdade ou poder de decisão, que é o fato do Mezuro estar contido dentro do Noosfero, tendo seu desenvolvimento limitado.

A contribuição do principal autor deste trabalho, com a funcionalidade de “Manter Repositórios”, é um resultado dessa nova comunidade de software livre formada a partir da decisão de tornar a plataforma Mezuro uma aplicação independente. É importante destacar que, embora o Mezuro esteja evoluindo para uma aplicação distinta do Noosfero, haverá uma integração, ainda a ser discutida pela comunidade de desenvolvedores, entre essas duas ferramentas futuramente.

6.1 Cronograma

Para o curto e médio prazo, no escopo deste trabalho, há atividades planejadas para o futuro do Mezuro. Até agora, o Mezuro evoluiu em sua arquitetura, migrando de plugin para aplicação independente com o Rails 4. Na segunda fase deste trabalho, vamos colaborar com novas funcionalidades, que também demandará uma etapa de pesquisa

complementar ao que fizemos até agora. As atividades planejadas são:

1. Complementação das pesquisas sobre software livre e evolução de software;
2. Pesquisa sobre visualização de software, formas visuais das métricas de código-fonte;
3. Colaboração com finalização da migração das funcionalidades do Mezuro Plugin para o novo Mezuro;
4. Incorporar formas visuais das métricas ao Mezuro (visualizações gráficas);
5. Implementar suporte a novas linguagens (e.g Ruby);
6. Escrita do TCC.

Atividade	Dez 2013	Jan 2014	Fev 2014	Mar 2014	Abr 2014	Mai 2014	Jun 2014
1	•	•					
2		•	•	•			
3	•	•	•				
4				•	•	•	
5						•	•
6	•	•	•	•	•	•	•

Tabela 5 – Cronograma para atividades do TCC2

É importante enfatizar que, após as leituras preliminares e uma avaliação inicial do Mezuro Plugin para a definição do tema e passos para este trabalho, ao final de agosto de 2013, este trabalho começou a ser desenvolvido em setembro de 2013, desde o aprendizado das tecnologias, além dos estudos teóricos para a elaboração deste texto. Em 2 meses, conseguimos colaborar efetivamente com o Mezuro ao desenvolver toda a parte de “gestão de repositório” de um projeto cadastrado no Mezuro, ou seja, um item central entre as funcionalidades do mesmo. Avaliamos que, durante o TCC 1, já percorremos a curva de aprendizado necessária para contribuir com o projeto de Mezuro, para, num primeiro momento, finalizarmos a migração para uma aplicação independente e, posteriormente, inserirmos novas funcionalidades, em especial no contexto de visualização de software, como uma das partes da área de evolução de software que será pesquisada durante o TCC 2 (7 meses, conforme o cronograma apresentado na Tabela 5).

Referências

- BASTIEN, J. C.; SCAPIN, D. L. et al. Ergonomic criteria for the evaluation of human-computer interfaces. 1993. Citado na página 26.
- CYBIS, W.; BETIOL, A. H.; FAUST, R. *Ergonomia e usabilidade*. [S.l.: s.n.], 2010. Citado 3 vezes nas páginas 25, 27 e 37.
- DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. *Information and software technology*, Elsevier, v. 50, n. 9, p. 833–859, 2008. Citado na página 21.
- ERGONOMIA, A. B. de. *O que é ergonomia @ONLINE*. 2013. Disponível em: <<http://www.abergo.org.br/>>. Citado na página 25.
- FERREIRA, J.; NOBLE, J.; BIDDLE, R. Up-front interaction design in agile development. In: *Agile Processes in Software Engineering and Extreme Programming*. [S.l.]: Springer, 2007. p. 9–16. Citado na página 27.
- INFORMÁTICA, L. de Utilizabilidade da. *ErgoList - Checklist @ONLINE*. 2013. Disponível em: <<http://www.labiutil.inf.ufsc.br/ergolist/check.htm>>. Citado na página 43.
- LEE, J. C.; JUDGE, T. K.; MCCRICKARD, D. S. Evaluating extreme scenario-based design in a distributed agile team. In: ACM. *PART 1———Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*. [S.l.], 2011. p. 863–877. Citado na página 28.
- LIDA, I. *Ergonomia: projeto e produção*. 2ª edição. [S.l.: s.n.], 2005. Citado na página 25.
- LIMAA, M. S. de; SOARESB, B. G.; BACALTCHUKC, J. Psiquiatria baseada em evidências. *Rev Bras Psiquiatr*, SciELO Brasil, v. 22, n. 3, p. 142–6, 2000. Citado na página 28.
- MEIRELLES, P. et al. Mezuero: A source code tracking platform. *CBsoft 2012*, FLOSS Competence Center – Universidade de São Paulo, 2012. Citado 2 vezes nas páginas 40 e 42.
- MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 2 vezes nas páginas 19 e 41.
- MORENO, A. M.; YAGÜE, A. Agile user stories enriched with usability. In: *Agile Processes in Software Engineering and Extreme Programming*. [S.l.]: Springer, 2012. p. 168–176. Citado na página 29.
- NICHOLS, D. M.; TWIDALE, M. B. Usability processes in open source projects. *Software Process: Improvement and Practice*, Wiley Online Library, v. 11, n. 2, p. 149–162, 2006. Citado na página 17.

- NIELSEN, J. *Usability engineering*. [S.l.]: Academic Press Limited, 1994. Citado na página 26.
- NIELSEN, J. Severity ratings for usability problems. *Papers and Essays*, 1995. Citado na página 37.
- QUALIPSO. *O Projeto Qualipso @ONLINE*. 2013. Disponível em: <<http://qualipso.icmc.usp.br>>. Citado na página 41.
- SANTOS, A. P. *Aplicação de práticas de usabilidade ágil em software livre*. 2012. Citado 4 vezes nas páginas 17, 29, 31 e 40.
- SANTOS, A. P.; KON, F. Adaptação de metodologias de usabilidade para o contexto de desenvolvimento de software livre. 2009. Citado na página 28.
- SCHWABER, K.; BEEDLE, M. *Agile software development with Scrum*. [S.l.]: Prentice Hall Upper Saddle River, 2002. Citado na página 21.
- SHNEIDERMAN, B.; BEN, S. *Designing The User Interface: Strategies for Effective Human-Computer Interaction, 4/e (New Edition)*. [S.l.]: Pearson Education India, 2003. Citado na página 26.
- STALLMAN, R. M.; GAY, J. *Free software, free society: Selected essays of Richard M. Stallman*. [S.l.]: CreateSpace, 2009. Citado na página 20.
- SYSTEM, G. O. *O que é software livre? @ONLINE*. 2013. Disponível em: <<http://www.gnu.org/philosophy/free-sw.en.html>>. Citado na página 20.
- ISO 25010, SYSTEMS and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. 2010. Citado na página 25.
- THOMAS, M. P. Why free software has poor usability, and how to improve it. *Computing & Internet, Usability*, 2008. Citado na página 17.
- YIN, R. K. *Case study research: Design and methods*. [S.l.]: Sage, 2009. Citado na página 39.
- ÁGIL, M. *Manifesto para o desenvolvimento ágil de software @ONLINE*. 2013. Disponível em: <<http://manifestoagil.com.br/>>. Citado 2 vezes nas páginas 21 e 22.