UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JOÃO HENRIQUE SENGER

CONCEITOS DE USABILIDADE DE SOFTWARE APLICADOS EM UM PROJETO WEB DE CONTROLE DE SALAS

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

JOÃO HENRIQUE SENGER

CONCEITOS DE USABILIDADE DE SOFTWARE APLICADOS EM UM PROJETO WEB DE CONTROLE DE SALAS

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Msc. Juliano Rodrigo Lamb

MEDIANEIRA
2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

CONCEITOS DE USABILIDADE DE SOFTWARE APLICADOS EM UM PROJETO WEB DE CONTROLE DE SALAS

Por

JOÃO HENRIQUE SENGER

Este Trabalho de Diplomação (TD) foi apresentado às 09:20 h do dia 03 de fevereiro de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Juliano Rodrigo Lamb, M.Sc
UTFPR – Campus Medianeira
(Orientador)

Prof. Ricardo Sobjak, M.Sc
UTFPR – Campus Medianeira
(Convidado)

Prof. Alessandra B.Garbelotti Hoffmann, M.Sc
UTFPR – Campus Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb, M.Sc
UTFPR – Campus Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

SENGER, João Henrique. 68f. CONCEITOS DE USABILIDADE DE SOFTWARE APLICADOS EM UM PROJETO WEB DE CONTROLE DE SALAS. 2015. Trabalho de diplomação (Curso Superior De Tecnologia Em Análise E Desenvolvimento De Sistemas). Universidade Tecnológica Federal do Paraná. Medianeira. 2015.

A usabilidade aplicada no desenvolvimento de *software* proporciona uma melhoria na qualidade de *software* e um melhor conforto para o usuário. Este trabalho tem como objetivo melhorar a usabilidade de um *software web* de controle de salas utilizando a linguagem Java, padrões de projetos, características de IHC (Interface Homem Computador) e usabilidade de *software.* A metodologia de avaliação utilizada é a de inspeção e cognição, em que o avaliador que é o próprio autor do trabalho inspeciona o conteúdo do programa e levanta possíveis problemas na interface. Após a avaliação é apontado em forma de quadros as principais características da interface e sua conformidade com as determinações de interface. Por fim, este trabalho apresenta o desenvolvimento de melhoria da usabilidade aplicado ao software, apoiado na pesquisa realizada.

Palavras-chave: engenharia de *software*, IHC, interface, qualidade de software.

ABSTRACT

SENGER, João Henrique. 68f. Usability evaluation applied to a control rooms of web design. 2015. Completion of course work (Technology Analysis and Systems Development), Federal Technological University of Paraná, Medianeira 2015.

Usability applied in software development provides an improvement in software quality and better comfort for the user. This work aims to improve the usability of a control rooms of web software using the Java language, design patterns, IHC characteristics (Human Interface Computer) and software usability. The evaluation methodology used is the inspection and cognition, where the evaluator who is the author's own work inspects the program content and raises potential problems in the interface. After the evaluation is pointed in tabular form the main features of the interface and its compliance with the interface determinations. Finally, this paper presents the development of improved usability applied to software, based on the research conducted.

Keywords: IHC, interface, software engineering, software quality.

LISTA DE SIGLAS

CC	Ciência Da Computação
CMMI	Capability Maturity Model Integration
EDL	Error In Design Logical
EDR	Error In Data Representation
ES	Engenharia De *Software*
HCI	Ambiguous Or Inconsistent Human/ Computer Interface
ICI	Inconsistent Component Interface
IDS	Intentional Deviation From Specifications
IES	Incomplete Or Erroneous Specifications
IET	Incomplete Or Erroneous Testing
IID	Inaccurate Or Incomplete Documentation
IHC	Interação Homem Computador
ISO/IEC	International Organization For Standardization/International Electrotechnical Commission
MCC	Misinterpretation Of Customer Communication
MIS	Miscellaneous
MPS.br	Melhoria Do Processo Do *Software* Brasileiro
NBR	Normas Brasileiras
NATO	North Atlantic Treaty Organization
PLT	Error In Programming Language Translation Of Design
SEI	Software Engineering Institute
SI	Sistemas De Informação
SQA	Software Quality Assurance
SQuARE	Software Quality Requeriments And Evalution
SWEBOK	Software Engineering Body Of Knowledge
TI	Tecnologia Da Informação
VPS	Violation Of Programming Standards

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Segundo Sommerville (2011), a engenharia de *software* é uma disciplina da área de CC (Ciência da Computação) que ocupa de todos os aspectos da produção de *software*. A meta da ES (Engenharia de *Software*) é ter uma boa relação custo-benefício no desenvolvimento de sistemas.

Pressman (2006), afirma que nenhuma pessoa poderia prever que milhões de programas de computadores tivessem de ser corrigidos, adaptados e aperfeiçoados à medida que o tempo passasse, e que o cumprimento das manutenções destes iriam adquirir mais recursos que todo o trabalho empenhado na criação de novos produtos.

A palavra *software* ainda é muito associada a programas de computadores, todavia não é apenas isso, mas também toda a documentação associada a ele para que este funcione corretamente (Sommerville, 2011).

O desenvolvimento de sistemas tem se tornado um fator dominante nas economias, as empresas trabalham com um grupo de desenvolvedores e não mais sem eles ou com apenas um. Existe um grande impacto para os usuários, pois o tempo para desenvolvimento ainda é grande, os custos são altos, os *softwares* são propícios a erros de execução, é gasto tempo mantendo programas existentes em vez de substituí-los por novos (Pressman, Engenharia de software, 2006).

A qualidade de *software* é uma importante área da engenharia de *software*. Porém, indicações da mesma ainda não são seguidas na íntegra, em que atividades ainda não são documentadas, ou são parcialmente documentadas, o que dificulta o projeto inteiro, não é desenvolvido um plano de projeto, em que pode ser dividido o trabalho entre a equipe e atribuído tempo para cada fase. Atualmente existem várias ferramentas que podem ser utilizadas na melhoria deste processo, bem como métricas que podem ser seguidas (Pressman, Engenharia de Software: Uma Abordagem Profissional, 2011).

Mesmo com o aumento da equipe de desenvolvimento, e um maior investimento financeiro na área de TI (Tecnologia da Informação), os programas de computadores muitas vezes não são desenvolvidos como o esperado.

Conforme relatórios de conferência de 1968 do comitê da NATO (KOSCIANSKI; SOARES, 2007, p. 22), bem como documentação de estudos

realizados na década de 1970 os problemas em volta do desenvolvimento de *software* são os mesmos que os atuais, ou seja:

a) Cronogramas não observados;
b) Projetos abandonados pelo grau de dificuldade;
c) Módulos que distinguem operações do combinado;
d) Programas que não cumprem o esperado;
e) Programas que são descartados pela dificuldade de manuseio;
f) Programas que deixam de atuar sem justificativas.

A utilização de *software* vem crescendo muito, pois o cotidiano das pessoas depende cada vez mais deste *software*.

A qualidade de *software* tem como objetivo aprimorar o processo de desenvolvimento e como consequência, melhorar o produto resultante, avaliar a habilidade de produto visando à emissão de documentos sobre aptidão do *software*, e a sua conformidade referente a um padrão ou norma.

Conforme Inthurn (2001) a ISO/IEC 9126 (NBR 13596) define as características de qualidade de *software* que devem estar presentes em todos os produtos, a ISO/IEC 12119 estabelece os requisitos da qualidade para pacotes de *software* e instruções para testes, já a ISO/IEC 14598 define um processo de avaliação da qualidade de produto de *software*.

Diversos fatores são evolvidos pela usabilidade, às dificuldades de aprendizagem, facilidades de lembrar os caminhos da tarefa, eficiência na utilização, baixa taxa de erros, e ser subjetivamente agradável ao usuário. A usabilidade é imprescindível, se o usuário notar uma grande dificuldade na utilização este irá buscar outra solução no mercado (Nielsen, 1993).

Um dos aspectos mais marcantes em um *software* é a usabilidade*,* pois as interações entre o usuário e o sistema influenciam na impressão sobre a qualidade.

A usabilidade é a qualidade que caracteriza o uso de um software, refere-se a um acordo entre interface, usuário, tarefa e ambiente, afinal um usuário mais experiente pode ter menos dificuldade em utilizar um pouco intuitivo do que um usuário que possua conhecimentos elementares na manipulação de um sistema (CYBIS, 2007).

Neste estudo serão apresentadas ferramentas, e metodologias que possam ser utilizadas no desenvolvimento de aplicativos, para que assim seja atingida a melhoria da usabilidade.

1.1 OBJETIVO GERAL

Aplicar técnicas de usabilidade de software em um estudo experimental para reservas e controle de salas.

1.2 OBJETIVOS ESPECÍFICOS

a) Apresentar características de IHC, usabilidade e métodos de avaliação;
b) Melhorar a usabilidade de um protótipo de *software web* de controle de salas;
c) Aplicar técnicas de melhoria da usabilidade;
d) Apresentar as vantagens do uso de conceitos de IHC e usabilidade, apontando resultados.

1.3 JUSTIFICATIVA

Sistemas confiáveis são exigidos cada vez mais, é determinado que sejam fáceis de manusear, e que atendam todas as necessidades do usuário. Possíveis melhorias no desenvolvimento de *softwares* devem ser apresentadas, apontando o uso da engenharia do *software* com aplicação da qualidade em projetos *web.*

Aplicando o uso da engenharia focando principalmente na usabilidade de *software*, pode-se reduzir problemas como o de desenvolvimento frustrado, ou seja, um *software* que além de não atender as necessidades do usuário é propício a erros.

“Um componente de software deve ser projetado e implementado de modo que possa ser reusado em muitos programas diferentes. Componentes modernos reutilizáveis encapsulam tanto os dados quanto o processamento aplicado aos dados, permitindo ao engenheiro de software criar novas aplicações a partir de partes reusáveis.” (Pressman, Engenharia de software, 2006, p. 6).

Pressman (2006) orienta que os *softwares* sejam desenvolvidos de forma genérica em que posteriormente podem ser adicionadas novas funcionalidades.

O intuito da engenharia de *software* é fornecer uma estrutura para a construção de aplicativos de alta qualidade, sendo que qualquer abordagem de engenharia deve se firmar num compromisso organizacional com a qualidade.

A interface molda a percepção do usuário sobre a qualidade do sistema. Se o usuário considerar que as cores, a disposição de tela, os textos, imagens, ícones e mensagens passadas pelo software são inadequados, obscuros ou desconfortáveis o usuário pode rejeitar funções e conteúdos importantes (Pressman, Engenharia de software, 2006).

Avaliar uma interface já existente tem como objetivo melhorar apresentação de conteúdos, quantificar o nível de satisfação dos usuários e aperfeiçoar a experiência de utilização. Análise e projeto de interface é essencial na interação homem-máquina no contexto atual, já que *Web Apps* não são utilizadas exclusivamente pelos seus desenvolvedores, um bom projeto de interface torna a utilização de um sistema mais fácil de aprender e utilizar gerando assim produtividade por parte dos usuários

1.4 ESTRUTURA DO TRABALHO

O trabalho se divide em cinco capítulos, o primeiro apresenta o contexto do trabalho, define os objetivos do trabalho e a justificativa da escolha do tema.

O segundo capítulo apresenta o embasamento teórico do trabalho, em que são apresentados assuntos a serem utilizado no protótipo do sistema.

O terceiro trata da análise de um protótipo, e mostra as tecnologias utilizadas.

O quarto capítulo apresenta os resultados e discussões do trabalho.

O quinto capítulo visa mostrar as considerações finais do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ENGENHARIA DE SOFTWARE

A ES (Engenharia de Software) se divide em camadas, a Figura 1 apresenta que qualquer abordagem deve se apoiar em um compromisso com qualidade.

Figura 1 - Camadas da Engenharia de Software.
Fonte: Pressman (2006, p. 17).

a) O alicerce da ES é a camada de processos, pois mantém unidas as camadas de tecnologia e permite o desenvolvimento racional e oportuno de software.
b) Os métodos da ES fornecem técnicas de “como fazer” para desenvolver softwares.
c) As ferramentas de ES fornecem apoio automatizado aos processos e métodos de desenvolvimento.
d) A documentação é recomendada para uma ES bem sucedida, esta fornece métodos com técnicas de “como faz”.

Para Pressman (2006) deve existir um processo para um projeto de *software*, e este processo contém pelo menos as seguintes características:

a) Comunicação: Conversa com o cliente e levantamento de requisitos;
b) Planejamento: Define tarefas a serem executadas, riscos, recursos disponíveis, recursos necessários, o produto a ser desenvolvido e o cronograma de trabalho;

c) Modelagem: Modelagem do sistema a ser desenvolvido, esta característica faz com que o cliente e o desenvolvedor entendam melhor os requisitos;
d) Construção: Processo de desenvolvimento e testes;
e) Implantação: Entrega do produto ao cliente.

Com base nas características citadas percebem-se duas atividades da engenharia de *software*: análise e projeto. A análise abrange o levantamento, elaboração e validação de requisitos. O projeto abrange as tarefas do trabalho como o modelo e desenvolvimento.

A SWEBOK (*software engineering body of knowledge*) divide a engenharia do *software* em onze áreas, sendo elas: requisitos, gerência de engenharia, projeto, métodos e ferramentas de engenharia, construção, processo de engenharia, testes, qualidade, manutenção, disciplinas relacionadas e gerência de configuração (KOSCIANSKI; SOARES, 2007). Na Figura 2 são exibidos os níveis de subdivisão da SWEBOK.

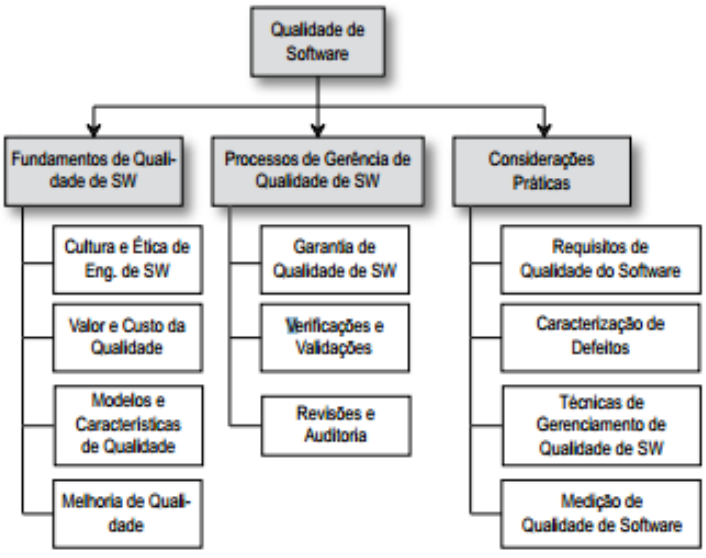

Figura 2 - Tópicos de divisão da qualidade.
Fonte: Koscianski; Soares (2007, p38).

2.2 ENGENHARIA WEB

A engenharia *web* dispõe de uma estrutura ágil para a produção de *WebApps* de qualidade industrial. As empresas sofisticadas requerem uma adequação constante, pois regras de negócio mudam com o tempo e os *stakeholders* solicitam mudança mesmo quando o tempo para entrega é curto (PRESSMAN; LOWE, 2009).

> Uma equipe ágil é uma equipe ligeira, capaz de responder apropriadamente a mudanças. A mudança é tudo o que é desenvolvido de *software* é. Mudanças no *software* sendo construído, mudanças nos membros da equipe, mudanças por causa de novas tecnologias, mudanças de todos os tipos que podem ser um impacto sobre o produto que eles constroem ou sobre o projeto que cria o produto. O apoio a mudanças deve estar embutido em tudo o que fazemos no *software*, algo que aceitamos porque isso é o "coração" do *software*. Uma equipe ágil reconhece que o *software* é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas (e sua capacidade de colaborar) são essenciais para o sucesso do projeto (JACOBSON (2002 apud PRESSMAN; LOWE, 2009, P. 13)).

Para Jacobson (2002) é necessário que a equipe de desenvolvimento tenha facilidade e agilidade em adaptações.

Segundo Pressman e Lowe (2009), o desenvolvimento *web* envolve uma série de processos, o qual pode ser chamado de arcabouço genérico, segue alguns exemplos:

a) Comunicação: Envolve a cooperação e comunicação de modo acentuado com os interessados no projeto, envolve o levantamento de requisitos e atividades associadas;
b) Planejamento: Articula um plano inicial para o projeto, descreve as etapas que devem suceder-se, as atividades a serem desenvolvidas, levantamento de riscos e recursos necessários, o cronograma a ser seguido, e o produto planejado;
c) Modelagem: Constitui protótipos que ajudam o desenvolvedor e cliente a captar melhor os requisitos do projeto *web*;
d) Construção: Condiz a produção de HTML, XML, Java e códigos semelhantes, contando com testes para disseminar erros no código;

e) Implantação: Entrega do projeto ao cliente, que executa avaliação do projeto.

A eficiência em manutenção e sua facilidade definem a qualidade de *webapps*, a Figura 3 apresenta esse trabalho. Os critérios apresentados na figura são de particularidade do engenheiro web, que deve projetar, construir e manter *webapps* a longo prazo (PRESSMAN; LOWE, 2009).

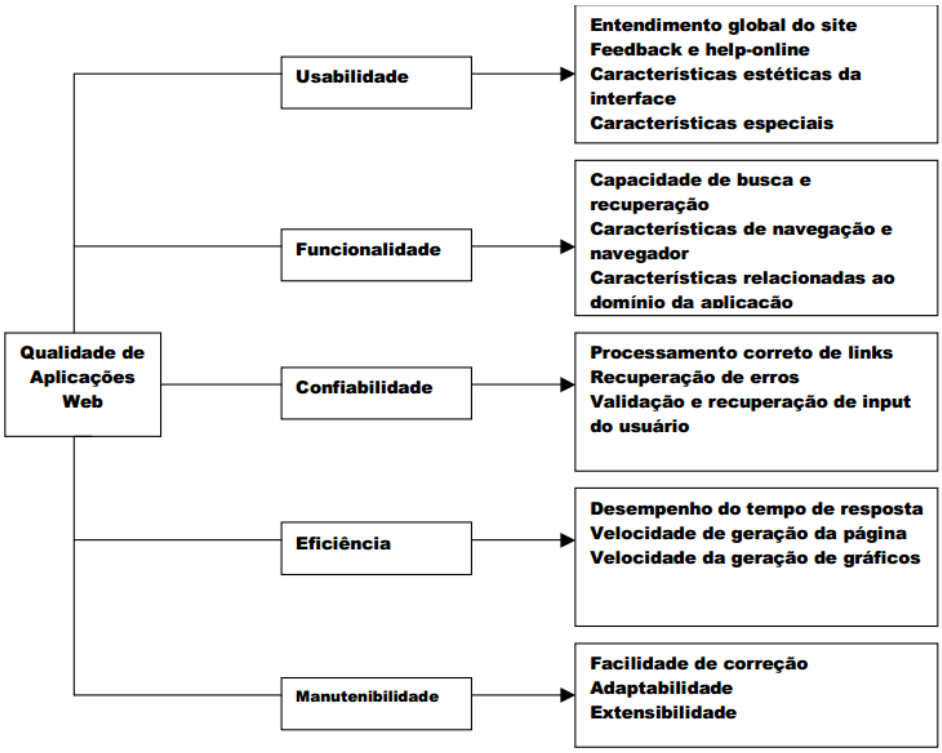

Figura 3 - Árvore de Qualidade *Web.*
Fonte: Adaptado de Pressman e Lowe (2009, p. 176).

Existe uma série de *websites*, que podem se enquadrar nas seguintes categorias:

- Website institucional ou informativo, funcionam como um cartão de visitas online;
- WISS (*Web-Based Information Systens*): que são um sistema baseado em web, como um sistema de gestão comercial.

2.3 DESENVOLVIMENTO DE SOFTWARE

Pressman (2006) afirma que os *software*s estão classificados nos seguintes grupos:

a) *Software* de Sistema: programas que apoiam outros programas como S.O e compiladores;
b) *Software* de Aplicação: Desenvolvido ser executado em um determinado negócio de uma empresa;
c) *Software* Científico e de Engenharia: Algoritmos para processarem números;
d) *Software* Embutido: desenvolvidos para serem executados dentro de um produto específico, como as teclas digitais de uma máquina de lavar;
e) *Software* para Linhas de Produtos: conhecidos como *softwares* de prateleiras (desenvolvido em larga escala e oferecido para qualquer interessado pela internet sem tributação);
f) *Software* para *Web*: aplicações que são executados na internet;
g) *Software* de Inteligência Artificial: softwares para robótica;
h) Computação ubíqua: *softwares* que realizam a computação distribuída;
i) *Software* Aberto: aqueles que oferecem o código fonte.

Para Sommerville (2011) o *software* não é só composto de código fonte, mas sim pelo agrupamento de vários artefatos.

Para desenvolver um sistema que esteja pronto para encarar os problemas observados do século XXI, deve-se compreender alguns fatos, como a mudança nos requisitos vem se tornando mais complexa, pessoas, empresas e órgãos governamentais dependem de sistemas para decisões habilidosas, além de que uma falha no sistema pode ser uma falha catastrófica. Conforme o sistema evolui, a chance é de que aumente o número de usuários e a durabilidade da aplicação (Pressman, Engenharia de Software: Uma Abordagem Profissional, 2011).

2.3.1 Processos de Desenvolvimento de Software

O processo de desenvolvimento pode ser dividido em quatro atividades fundamentais: Especificações; Desenvolvimento; Validação e Evolução, a primeira deve abordar as funcionalidades; operações e restrições. O processo de desenvolvimento o *software* deve atender as especificações. A validação deve assegurar que seja o esperado pelo cliente. A evolução trata do atendimento nas solicitações de modificação feita pelos clientes (Sommerville, 2011).

Para assegurar o desenvolvimento de forma consistente é preciso utilizar boas práticas de ES, com o uso apropriado de um processo de desenvolvimento. O processo de desenvolvimento varia de acordo com o projeto (Pressman, Engenharia de software, 2006).

Pressman (2011), ainda cita cinco atividades no processo de desenvolvimento:

a) Comunicação: comunicar com os *stakeholders*;
b) Planejamento: criar um plano para o projeto com as atividades a serem executadas, prever possíveis riscos e recursos necessários, além de elaborar um cronograma;
c) Modelagem: criar um esboço do sistema, para que se possa entender a aparência desejada;
d) Construção: produzir códigos e testes;
e) Emprego: entregar o sistema ao cliente, o qual deve avaliar e fornecer *feedback*.

“Um processo define quem está fazendo o que, quando e como para alcançar um objetivo” (Jacobson, Booch, e Rumbaugh. ([19--?] apud Pressman, 2006, p. 19)).

Para Oliveira e Neto (2003 apud BELMONTE, SCANDELARI e KOVALESKI, 2004, p. 2), pode-se comparar o processo de desenvolvimento a produção industrial, na qual instrumentos precisam de um cuidado particular, visto que influenciam diretamente na produção, e necessitam táticas que ofereçam o menor número de interrupções possíveis.

Na Figura 4 pode ser verificado um modelo de protótipo sugerido por Pressman (2006), o modelo deve começar da comunicação, até chegar no feedback,

em que deve ser avaliado para aperfeiçoar os requisitos, porém este protótipo não deve ser utilizado como versão final, ele serve apenas para avaliar as verdadeiras necessidades do cliente, e para o desenvolvedor ter uma base do que precisa ser feito.

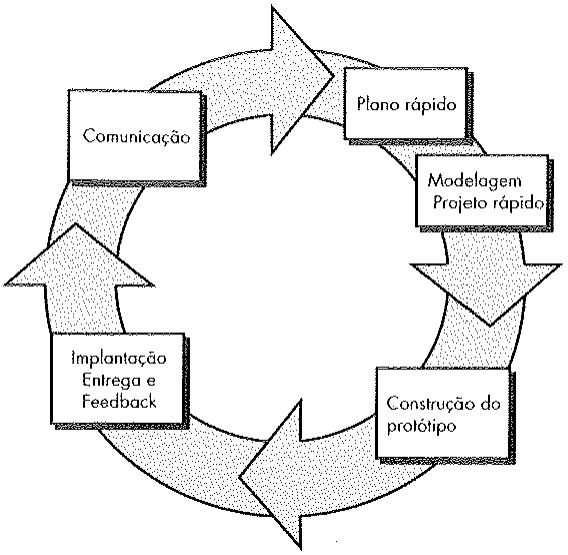

Figura 4 - Modelo de Prototipagem.
Fonte: Pressman (2006, p. 43).

2.3.2 Problemas no Processo Desenvolvimento de Sistemas

Um dos problemas no desenvolvimento de *software* é resolver o que dizer, e não como dizer, o aperfeiçoamento iterativo e interatido dos requisitos com a ajuda de um exemplo rápido é visto como um afronto ao fundamento conceitual de *software*, esse exemplo pode simular interfaces escolhidas e realizar a principais funções no sistema, desenvolver o sistema de forma incremental também pode ser visto como uma forma de enfrentar dificuldades essenciais (PETERS; PEDRYCZ, 2001).

A coleta dos requisitos é um grande desafio, pois geralmente as empresas não dispõem de tempo para uma coleta de informações aprofundada, e consequentemente o *software* leva mais tempo para ser desenvolvido, por não fazer a coleta profunda e não realizar certos procedimentos, tudo acaba sendo feito ao mesmo tempo, o que impossibilita a coleta de métricas e diminui a produtividade. Na maioria das vezes os problemas ocorrem porque faltaram alguns requisitos, ou

foram levantados os requisitos de forma simplificada de forma que não observa certas funcionalidades (Pressman, Engenharia de Software: Uma Abordagem Profissional, 2011).

Conforme Schulmeyer e Mcmanus (1999), muitas vezes o *software* tem uma baixa qualidade, porque recentemente as empresas começaram a praticar e entender os testes e a sua influência, o que também ocorre com a avaliação de qualidade integralmente, do começo ao encerramento do processo de desenvolvimento. Qualidade de *software* e conceitos quantitativos são termos recentes que estão se conceituando, de toda forma um sistema de baixa qualidade é algo difícil de manter.

Estimativas de desenvolvimento são feitas de forma imprecisa, o que incide na entrega do produto, possibilitando atrasos e problemas com o cliente, com a imprecisão do levantamento de requisitos que foi feito às pressas, as estimativas acabam sendo quebradas, na maioria das vezes os projetos são mal dimensionados, prevendo muito ou pouco tempo. Sem um indicativo concreto de produtividade, as chances de estimar a competência de ferramentas, métodos, ou padrões é muito pequena (Pressman, Engenharia de software, 2006).

2.3.3 O Que é Um Software de Qualidade?

Para Crosby (1979 apud Pressman, 2006, p. 577), qualidade não é o que as pessoas conhecem a respeito. O problema é o que elas pensam saber a respeito.

"As pessoas esquecem quão rapidamente você fez um trabalho - mas elas sempre se lembram de quão bem você o fez" (Newton ([19--?] apud PRESSMAN, 2006, p. 579)).

Um sistema pode ter erros e mesmo assim ter qualidade, um erro é algo a ser corrigido, mas não necessariamente faz com que o produto deixe de ter qualidade, alguns *softwares* possuem erros já conhecidos pelos usuários e considerados de baixo destaque, outros erros são apresentados em determinadas condições que representam a minoria dos usuários, logo o gerente de projeto pode optar por não alterar o código, pois modificações neste podem implicar em profundas alterações (KOSCIANSKI; SOARES, 2007).

> No desenvolvimento de *software*, a qualidade abrange os requisitos, as especificações e o projeto do sistema. A Qualidade de conformidade é um assunto concernente, principalmente às implementações. Se a implementação segue o projeto e o sistema resultante satisfaz os requisitos e metas de desempenho, a qualidade de conformidade é alta (Pressman, Engenharia de software, 2006, p. 579).

Conforme Cortes e Chiossi (2001), desenvolver *software* seguindo padrões é algo essencial para quem busca qualidade, além disso, simplicidade na manutenção, reutilização de códigos, e claridade no entendimento do que está desenvolvido ajuda a promover a confiabilidade. A padronização somada à documentação ajuda os autores a entenderem o sistema, o que está relacionado diretamente à qualidade.

2.4 QUALIDADE DE SOFTWARE

Conforme Ferreira e Leite (2003), a qualidade é fundamental indiferente da empresa, e estas precisam de SI descomplicados de ser utilizados.

"A qualidade de um produto corresponde a quanto ele muda o mundo para melhor" (DeMarco (1999 apud PRESSMAN, 2006, p. 579))

> Para adquirir qualidade é preciso empregar corretamente o uso das metodologias pela equipe de desenvolvimento. Mas o que é qualidade? Como medi-las?
> Devem ser estabelecidos critérios que sirvam para avaliar o produto, o autor ainda diz que cada produto pode ter especificações diferentes (KOSCIANSKI; SOARES, 2007, P. 22).

No Quadro 1 Pressman (2006) aponta um exemplo genérico das possíveis causas dos problemas de *software*, coletados durante um ano, a partir do desenvolvimento até a entrega do produto final.

	Total		Grave		Moderado		Trivial	
Erro	Qtd.	%	Qtd.	%	Qtd.	%	Qtd.	%
IES (*Incomplete or Erroneous Specifications)*	205	22%	34	27%	68	18%	103	24%
MCC (*Misinterpretation Of Customer Communication*)	156	17%	12	9%	68	18%	76	17%
IDS (*Intentional Deviation From Specifications*)	48	5%	1	1%	24	6%	23	5%
VPS (*Violation Of Programming Standards*)	25	3%	0	0%	15	4%	10	2%
EDR *(Error in data representation)*	130	14%	26	20%	68	18%	36	8%
ICI (*Inconsistent Component*	58	6%	9	7%	18	5%	31	7%

Interface)								
EDL *(Error In Design Logical*)	45	5%	14	11%	12	3%	19	4%
IET (*Incomplete Or Erroneous Testing*)	95	10%	12	9%	35	9%	48	31%
IID (*Inaccurate Or Incomplete Documentation*)	36	4%	2	2%	20	5%	14	3%
PLT (*Error In Programming Language Translation Of Design*)	60	6%	15	12%	19	5%	26	6%
HCI (*Ambiguous Or Inconsistent Human/ Computer Interface*)	28	3%	3	2%	17	4%	8	2%
MIS (*Miscellaneous*)	56	6%	0	0%	17	4%	41	9%

Quadro 1 – estatística de SQA (garantia de qualidade de software).
Fonte: adaptado de Pressman (2006, p. 590).

Pode-se notar que IES, MCC, e EDR são as principais causas de erros, para corrigir esses problemas Pressman (2006) indica que para melhora MCC pode ser introduzido praticas simplificadas de coleta de requisitos para melhorar a identificação e o entendimento das necessidades com o cliente. Para aperfeiçoar EDR podem ser utilizadas ferramentas de UML e realizado verificações estreitas no projeto de dados. Conforme esses erros são corrigidos outros sobem ao topo, então deve ser focado em causas vitais, assim quando essas são retificadas outras caem da extremidade da pilha.

Para Crosby (1992 apud Koscianski; Soares, 2007, p. 25) “A qualidade é conformidade aos requisitos”. Isso mostra que os requisitos podem ser utilizados como ponto de referência, e que deve ser traçado metas com margens de precisão.

Segundo Sommerville (2003 apud Koscianski; Soares, 2007, p. 26) “Diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneiras distintas. Os engenheiros de requisitos precisam descobrir todas as possíveis fontes de requisitos e encontrar pontos em comuns e os conflitos”.

Os requisitos de *software* mudam com o tempo, porém o impacto depende da diferença de tempo da introdução, quando as alterações são informadas com antecedência o impacto financeiro é relativamente baixo. Contudo o impacto aumenta no decorrer do tempo (Pressman, Engenharia de software, 2006). O autor afirma que “quanto mais cedo você começar a escrever código, mais vai demorar para acabar” (Pressman, Engenharia de software, 2006, p. 11). Segundo o mesmo de 60% a 80% do esforço é destituído após a primeira entrega.

Para Koscianski e Soares (2007) as principais dificuldades estão no escopo do projeto, e a mudança nos requisitos, o que acaba resultando em uma difícil maneira de prever como o aplicativo deve ser concluído. Uma técnica que pode

ajudar a reduzir este impacto é o desenvolvimento de tecnologias e ferramentas que podem automatizar o processo, diminuindo as tarefas para o desenvolvedor e garantindo uma igualdade nas tarefas executadas.

Os métodos, técnicas e ferramentas da engenharia do *software*, podem ajudar a facilitar o trabalho. Por isso deve-se definir com precisão qual o objetivo a assumir.

2.4.1 Normas ISO/IEC

Um modelo existente é a norma SQquaRE (*Software Quality Requeriments And Evalution*), em que a ISO/IEC 25000 fornece indicações sobre o uso desta nova série de normas que se originou das ISO/IEC 9126 e ISO/IEC 14598 que tratam de qualidade de produto de *software*. A ISO/IEC 9126 e a ISO/IEC 14598 são divididas conforme o Quadro 2.

9126-1	Modelo de qualidade de *software*
9126-2	Métricas externas
9126-3	Métricas internas
9126-4	Métricas para qualidade em uso
14598-1	Visão geral
14598-2	Planejamento de gerenciamento
14598-3	Guia para desenvolvedores
14598-4	Guia para adquirentes
14598-5	Guia para avaliadores
14598-6	Documentação dos módulos

Quadro 2 - ISO/IEC 9126 e ISO/IEC 14598.
Fonte: Adaptado de Koscianski; Soares (2007, p. 27)

A ISO/IEC 9126 proporciona um modelo de qualidade para o produto, enquanto a ISO/IEC14598 cuida da parte gerencial. A ISO/IEC 9126 está definida nas seguintes características:

a) Funcionalidade: Diz respeito ao que o *software* faz, é determinado que seja feito um escopo;

b) Manutenibilidade: Está relacionada com a manutenção de *software*, que deve ser feita de forma prática e fácil;
c) Usabilidade: Refere-se à facilidade na utilização do produto, que é difícil debater, pois depende de cada usuário;
d) Confiabilidade: Eficácia em manter um certo nível de desempenho quando executado em um certo âmbito de uso;
e) Eficiência: Alocação de recursos e tempo de resposta;
f) Portabilidade: Indica a possibilidade em executar o aplicativo em diferentes plataformas;
g) Medições: A medição está definida na ISO/IEC 2502n que se baseia na ISO 15939, em que é definido atributos a serem medidos.

A ISO/IEC 14598 complementa a ISO/IEC 9126, e permite uma avaliação padronizada da qualidade.

Usabilidade é caracterizada pela ISO 9241-11 de forma parecida com a ISO 9126.

A ISO/IEC 15504 tem relação com o modelo CMMI e é uma evolução da ISO/IEC 12207 e possui níveis de capacidade por processo que podem ser verificados no Quadro 3, e pode ser aplicada no melhoramento ou avaliação dos processos.

Nível	Nome	Descrição
0	Incompleto	O processo não é implementado ou falha em atingir seus objetivos.
1	Executado	O processo essencialmente atinge os objetivos, mesmo se de forma pouco planejada ou rigorosa.
2	Gerenciado	O processo é implementado de forma controlada (planejado, monitorado e ajustado); os produtos por ele criados são controlados e mantidos de forma apropriada.
3	Estabelecido	O processo é implementado de forma sistemática e consistente.
4	Previsível	O processo é executado e existe um controle que permite verificar se ele se encontra dentro dos limites estabelecidos para atingir os resultados.
5	Otimizado	O processo é adaptado continuamente para, de uma forma mais eficiente, atingir os objetivos de negócio definidos e projetados.

Quadro 3 - Níveis de capacitação ISO/IEC 15504.
Fonte: Adaptado de Koscianski; Soares (2007, p. 159)

Para avaliar a capacidade de um processo é adotada uma visão mais detalhada, são definidos nove atributos agrupados em níveis de capacidade, que são aplicados a qualquer processo, estes atributos podem ser verificados no Quadro 4.

Nível	Atributo
1	1.1 Execução: O processo atinge os objetivos dele esperados.
2	2.1 Administração do processo: Avalia até que ponto o processo é gerenciado para produzir os produtos de trabalho que satisfazem aos seus objetivos. 2.2 Administração dos produtos obtidos do processo: Avalia até que ponto os produtos de trabalho são documentados, controlados e verificados.
3	3.1 Definição: Avalia as interações com outros processos e competências com base em um processo padronizado 3.2 Implementação: Implementa o que foi identificado no item 3.1, aloca recursos e se preciso, é realizado treinamentos.
4	4.1 Medição: Avalia objetivos e medições, coleta analisa e publica resultados 4.2 Controle: Estabelece limites de variação para o item 4.1
5	5.1 Inovação: Define melhorias, avalia até em que podem ser realizadas mudanças na definição, gerencia e execução do processo. 5.2 Otimização: Verifica a contribuição das mudanças de processos na melhoria contínua, gerencia a implementação de mudanças.

Quadro 4 - Níveis de atributos de processos ISO/IEC 15504.
Fonte: Adaptado de Koscianski; Soares (2007, p 167)

A ISO 8402 define qualidade como a totalidade das características de uma entidade que lhe confere a capacitação de atender as necessidades explicitas e implícitas.

2.4.2 Processos e Métodos para Qualidade

Segundo Cameira (2003), algumas atividades que compõe processos são semelhantes entre si, estas atividades devem gerar componentes capazes de se conectarem facilmente, pelas características de aplicação e padrões de integração. As atividades iguais geram um só componente que será reutilizado em vários processos.

Pressman (2011), afirma que a reusabilidade é uma característica importante de um componente de *software* de alta qualidade, isto é, o *software* deve ser planejado, e implementado de forma que possa ser reutilizado em outros programas. Um componente reutilizável facilita que o engenheiro de *software* crie novos programas a partir destas partes reusáveis.

Cada componente do processo terá relação com o componente do sistema que proporciona o *software* por meio de um processo componentizado.

2.4.3 Ferramentas Para Obtenção de Qualidade

Conforme Sommervile (2003), a tecnologia CASE está amplamente desenvolvida, e existem ferramentas disponibilizadas por diversos fornecedores, o autor divide em três perspectivas estas ferramentas, sendo elas funcionais, de processo e de integração. A maioria das ferramentas CASE oferecem apoio a toda ou quase toda as etapas do processo de desenvolvimento. Fuggetta (1993 apud SOMMEVILLE 2003, p. 54) classifica as ferramentas CASE em três categorias:

a) Ferramentas: apoiam tarefas individuais, compilação de programas, comparação de resultados de testes;
b) *Workbenches:* auxiliam etapas ou atividades do processo, como a especificação e o projeto, geralmente consistem em um grupo de ferramentas com um grau de integração superior ou inferior;
c) Ambientes: dispõem de ajuda a todo ou quase todo o processo de *software*, na maioria das vezes dispõem de distintos *workbenches* que são integrados de alguma forma.

Sommeville (2003) ainda afirma que as ferramentas CASE tem oferecido melhorias na qualidade de *software* e na produtividade, embora esse aumento não seja conforme previsto pelos primeiros defensores da tecnologia, que sugeriam que era possível fazer enormes melhorias com o seu uso. Foi atingida uma melhoria de quarenta por cento em média com o uso desta tecnologia (HUFF (1992, apud SOMMEVILLE, 2003, p. 53)).

A UML é uma linguagem que contribui em diversos sentidos no desenvolvimento de *software*, ela pode ser usada para especificar, visualizar, contribuir, documentar, desenhar, entender, pesquisar, manter e controlar informações sobre um sistema. Pode ser utilizada no desenvolvimento de sistemas estruturados ou orientados a objetos, não elucida um procedimento padrão, entretanto procura ser útil e objetiva em processos de desenvolvimento (RUMBAUGH (1999 apud BELMONTE, SCANDELARI e KOVALESKI, 2004)).

2.5 IHC (INTERAÇÃO HOMEM COMPUTADOR)

O termo IHC surgiu com o intuito de propor “harmonia” na atividade exercida, e vice-versa entre homem e máquina, baseia-se por aprendizagens sobre a humanidade de um lado e computadores do outro, abrangendo a forma que um influencia o outro (Rocha, MALDONADO, & WEBER, 2001).

A IHC baseia-se em três requisitos não-funcionais: acessibilidade, usabilidade e comunicabilidade. A acessibilidade é a capacidade de qualquer usuário conseguir acessar o sistema, sem que a interface lhe imponha algum obstáculo. A Usabilidade refere-se a qualidade que define o uso de software, é uma combinação entre usuário, tarefa, ambiente e interface. A comunicabilidade trata a compreensão pelo usuário das intenções lógicas aplicadas no sistema (CYBIS, 2007).

Para Pressman (2006) a interface em meio dos usuários procede da consciência humana na visão, audição e tato, constantemente, facilitando que o usuário de SI (Sistemas de Informação) adquira a informação, institua-a na memória e à processe, explorando a capacidade da imaginação.

A interação é o processo que trata das ações do usuário sobre a interface de um sistema, e suas interpretações sobre as respostas reveladas pela mesma (SOUZA, 2014). Por meio da interface, os usuários interagem com o sistema, ela também é considerada um ambiente virtual de ações (LEITE, 2001).

Rocha et al. (2001), explica que IHC é um conteúdo que envolve várias áreas, e com isso a contribuição de diversas ciências é fundamental. O efeito disso é uma conduta a ser observada para assegurar uma comunicabilidade, e convívio entre SI e usuários, partindo de interfaces charmosas e efetivas.

Com a gradativa e frequente utilização das tecnologias no planeta, IHC se popularizou, ela foi proferida com base em buscas e aprendizagens enfocando fatores humanos, tendo a intenção de desenvolver interfaces pensando nas imposições de diversos tipos, ambientes, conjuntos, categorias e *staff’s,* de usuários (DIAS, 2003).

A análise das características e exigências dos usuários, junto com suas interpretações, na relação com vários sistemas de informação na área de CC ocasionada com a ES e decorrente da engenharia de usabilidade, é oficio de

estudos da usabilidade (Pressman, Engenharia de Software: Uma Abordagem Profissional, 2011).

2.5.1 Métodos de Avaliação

Existem três métodos de avaliação de IHC: investigação, observação e inspeção. Estes métodos orientam os passos a serem seguidos pelos critérios de avaliação (JUNQUEIRA; SILVA, 2010).

2.5.1.1 Investigação

Este método envolve a coleta de dados por meio de questionários, entrevistas, grupos de foco e pesquisas na literatura, que permitam que o responsável pela avaliação interprete as opiniões e comportamentos do usuário em relação ao sistema avaliado, neste caso, ao site (JUNQUEIRA; SILVA, 2010).

Instrumentos de investigação são questionários que tem por objetivo coletar dados, dentro de uma área de interesse, na inquisição de um grupo dentro de uma população. Independentemente de possuírem o mesmo objetivo, os questionários, diferentemente da entrevista, não articulam interação direta entre questionado e argumentador. (AMARO, PÓVOA e MACEDO, 2004).

O método de investigação, é utilizado para entender a situação atual de interação, afirmar ou esmorecer decisões de design, identificar problemas e coletar ideias de mudanças, baseadas nas perspectivas e interações do usuário com o sistema (MEMÓRIA, 2014).

2.5.1.2 Observação

A observação fornece informações baseadas na inspeção da realização de tarefas, detecta incidentes que ocorrem no momento da experiência de uso do

sistema. Pode ocorrer tanto num laboratório quando em um ambiente de sistema em produção (JUNQUEIRA; SILVA, 2010).

A metodologia de observação concede a coleta de dados mais variados, porque alguns problemas de interação não são constatados pelo próprio usuário ou expressados em palavras, e a observação do uso permite ao analista ter uma visão dos problemas e aspectos positivos, que não foram identificados. Os dados desse tipo de avaliação são guardados em forma de anotações do observador (PRATES e BARBOSA, 2014).

2.5.1.3 Inspeção

O método de inspeção permite que o analista (avaliador) examine o sistema de modo que sejam detectados problemas, para isso, o avaliador se comporta com o perfil esperado de um determinado tipo de usuário, previamente especificado na documentação, e testa o sistema, apontando os erros que esse usuário iria identificar (CUNHA, 2012).

A compreensão é parte da inspeção, e ela refere-se a Descoberta (ou reconhecimento) de informações por aprendizado (SUH, 2005). A análise de cognição trata da argumentação do processo de aprendizado por um usuário sob a utilização de um produto, da facilidade de memorização e aprendizado (MEMÓRIA, 2014).

Na avaliação de uma interface por inspeção, para verificar a qualidade cognitiva de um *software* os avaliadores utilizam o sistema de Percurso Cognitivo (Cognitive Walkthrough), este tipo de avaliação pode ser verificado no Quadro 5 (CYBIS, BETIOL e FAUST, 2010).

• O usuário executará a tarefa proposta, sem nenhum tipo de auxílio? • Se ocorrer algum erro na execução de uma tarefa, é possível voltar a tela inicial ou desfazê-la? • O usuário é capaz de entender as respostas vindas do sistema, no término ou inicio de um trabalho?

Quadro 5 - Perguntas para avaliação de interface.
Fonte: Cybis, Betiol e Faust (2010, p.216);

Para analisar a capacidade de entendimento de uma interface por um usuário, observações de imagens ou propostas de tarefas a serem realizadas, são importantes para que o avaliador constate as dificuldades do usuário no controle do sistema (CYBIS, BETIOL e FAUST, 2010).

2.5.2 Usabilidade

A expressão usabilidade começou a ser utilizada ainda na década de oitenta, como um sucessor do termo *user-frindly.* Um dos principais motivos para esta mudança foi a não necessidade para o usuário em ter máquinas amigáveis, mas sim que elas realizem as suas tarefas com sucesso. Até pelo fato que, o programa pode ser amigável para um usuário e não para outro, sendo que as indispensabilidades podem variar entre usuários (DIAS, 2003).

A usabilidade é uma das características mais marcantes de um sistema, porque a interação entre usuário exercem influência sobre a impressão de sua qualidade, sendo assim, problemas relacionados a uso de software devem ser encarados com seriedade pelos programadores, pois um usuário que não compreende bem a interface de um sistema e apresente dificuldades em utiliza-la, tem maior chances de desemprenhar mal sua tarefa, tornar-se frustrado pela sua experiência de uso ou até mesmo abandonar a ferramenta, pela sua dificuldade de operação (KOSCIANSKI e SOARES, 2006).

Pressman (2006) afirma que a usabilidade é um experimento em medir a *user-friendlines,* ao mesmo tempo em que qualidade, que representa a amigabilidade no português, ou seja, qualidade e amigo do usuário. Pressman ainda assegura que caso o programa deixe de ser *user-friendly* repetidamente estará destinado ao insucesso, ainda quando as funções que ele exerce sejam valiosas.

Conforme Cybis (2007), a engenharia de usabilidade promove o “esforço sistemático das empresas e organizações para desenvolver programas de *software* interativo com usabilidade”. Sendo assim ela surde da engenharia do *software*, logo:

> É um rebento da engenharia de sistemas de *hardware*. Ela abrange um conjunto de três elementos fundamentais, métodos, ferramentas e procedimentos, que possibilita ao gerente o processo do desenvolvimento de *software* e oferece ao profissional uma base para a construção de *software* de qualidade produtivamente (Pressman, Engenharia de software, 2006, p. 581).

Cybis (2007) indica que o desenvolvimento de um sistema com usabilidade constitui-se de uma análise cautelosa de vários integrantes do cenário de uso, e da contribuição diligente de usuários na escolha de projetos de interfaces, notado conforme o progresso de configuração de qualidade interna e externa ao sistema.

Nielsen (1993) cita cinco atributos de usabilidade, são eles:

a) Facilidade de aprendizado: simples e fácil de entender, para rapidamente entender o sistema e executar suas tarefas;
b) Eficiência de uso: permitir alta produtividade na execução das tarefas;
c) Facilidade de memorização: fazer com que o usuário consiga realizar as tarefas com facilidade mesmo sem usar o sistema com frequência;
d) Baixa taxa de erros: recuperar erros, caso ocorram e permitir que o usuário execute suas atividades em grandes empecilhos;
e) Satisfação Subjetiva: o usuário deve achar o sistema agradável e se sentir satisfeito.

O mecanismo pelo qual são estabelecidas as informações entre o sistema e o usuário é chamado de interface. Se os fatores humanos tiverem sido levados em conta no desenvolvimento, o convívio entre o usuário e o *software* será harmonioso, já se os fatores humanos tiverem sido ignorados o sistema será visto pelo usuário como “não amigável” (Pressman & Lowe, Web Engineering: A Practitioner's Approach, 2009).

A usabilidade está ligada diretamente a interface e é a tendência do *software* em permitir que o usuário alcance seus objetivos de interação com o sistema. Fornecer fácil aprendizagem, facilitar a utilização eficiente e apresentar poucos erros são aspectos importantes para a compreensão de boa usabilidade para o usuário.

A utilização de padrões de design, de componentes gera uma aparência elegante que para algumas classes de usuários pode ser definitiva para a avaliação da qualidade do sistema, e especialmente para sua usabilidade. O aspecto visual, é um importante fator de usabilidade, pois permite que o usuário utilize suas habilidades de uso de uma aplicação para outra, ou também de uma sessão do site para outra (PAGANINI, 2011).

Quando as informações apresentadas pelo IHC forem incompletas, confusas, incompreensivas a aplicação deixará de atender as necessidades do usuário (Pressman & Lowe, Web Engineering: A Practitioner's Approach, 2009).

Alguns dos fatores que devem ser levados em conta para uma boa interação entre o usuário e o *software* são:

a) Flexibilidade;
b) Janelas;
c) Mensagens de erros;
d) Forma da entrada de dados;
e) Informações emitidas pelo *software*;
f) Alertas;
g) Interatividade.

A usabilidade na *web* inclui um bom uso das cores, tamanhos adequados de fonte, design responsivo, quantidade de informação, qualidade, compatibilidade entre navegadores, convenção de nomenclatura, acessibilidade e internacionalização (SUH, 2005).

A usabilidade, devido sua subjetividade, é difícil de avaliar, o que não significa que a especificação da mesma deva ser desconsiderada. Para um sistema com alta usabilidade é testa-lo por meio de testes de compreensão, em que a maneira com que o usuário compreende os itens da interface é quantificado, sua organização, posicionamento propósitos dos itens e também a realização de testes de tarefas, em que o avaliador solicita a realização de uma tarefa, e avalia a dificuldade do usuário de conseguir executa-la (KRUG, 2008).

2.5.2.1 Inspeção de Usabilidade

A inspeção de usabilidade define-se como um conjunto de métodos baseados em se ter avaliadores inspecionando ou observando aspectos relacionadas a usabilidade de uma interface de usuário (ROCHA; BARANAUSKAS, 2003).

Rocha e Baranauskas (2003) citam que problemas de usabilidade podem ser definidos como aspectos da interface de usuário que podem causar uma usabilidade reduzida ao usuário final do sistema.

Segundo Rocha e Baranauskas (2003), Métodos de inspeção podem ser aplicados em fases iniciais ou finais do projeto, e o resultado é um relatório formal dos problemas indentificados com recomendações para mudanças.

Dente os vários métodos de inspeção, Rocha e Baranauskas (2003) destaca os citados:

- Avaliação Heurística: é feita a inspeção na interface tendo como base uma lista *heurística* de usabilidade.
- Avaliação de *Guidelines:* A interface é analisada no entendimento de verificar se está de acordo como uma lista de *guidelines* de usabilidade.
- Inspeção de Consistência; O avaliador verifica a consistência de uma família de interfaces, quanto à terminologia, cores, *layout*, formatos de entrada e saída e tudo o mais dentro da interface.
- Percurso Cognitivo: o avaliador simula o usuário na interface para executar tarefas típicas. Tarefas mais frequentes são o ponto inicial de análise, entretanto tarefas críticas, como recuperação de erros, também são percorridas.

2.5.2.2 Percurso Cognitivo

Rocha e Baranauskas (2003) cita que durante o percurso cognitivo os revisores avaliam a interface proposta no contexto de uma ou mais tarefas do usuário. A entrada para uma sessão de percursos inclui uma descrição detalhada da interface, o cenário da tarefa, suposições explícitas sobre a população de usuários e o contexto de uso, e também uma sequência de ações que o usuário deverá que fazer para executar corretamente a tarefa. O processo de percurso pode ser dividido em duas fases básicas, fase preparatória e fase de análise, no Quadro 6 será apresentado as características das fases.

Fase Preparatória: • Analistas definem tarefas, sequências de ações para cada tarefa, população de usuário e a interface a ser analisada. 1. Quem serão os usuários do sistema? 2. Qual tarefa (ou tarefas) devem ser analisadas? 3. Qual é a correta sequência de ações para cada tarefa e como pode ser descrita?

4. Como é definida a interface?

Fase de Análise:

- Objetiva contar uma estória verossímil que informe sobre o conhecimento do usuário e objetivos, e sobre o entendimento do processo de solução de problemas que leva o usuário a "adivinhar" a correta solução. Analistas respondem quatro questões:

1. Os usuários farão a ação correta para atingir o resultado desejado?
2. Os usuários perceberão que a ação correta está disponível?
3. Os usuários irão associar a ação correta com o efeito desejado?
   Se a ação correta for executada os usuários perceberão que foi feito um progresso em relação à tarefa desejada?

- Uma estória verossímil de fracasso será contada se alguma das questões acima tiver resposta negativa

Quadro 6 - Processo de Percurso Cognitivo.
Fonte: Rocha; Baranauskas (2003, P. 186).

Rocha e Baranauskas (2003) explica que antes da definição das entradas para o percurso na fase preparatória quatro aspectos do Quadro 7 devem estar plenamente acordados:

- Quem são os usuários do sistema?

Pode ser uma descrição simples, como “pessoas que usam LINUX”, ou “Usuários de Windows que trabalha com Microsoft Office”.

- Qual a tarefa será analisada?

A análise deve ser limitada a uma razoável, mas representativa, coleção de tarefas. Algumas tarefas devem ser escolhidas como exemplo a partir da funcionalidade central da aplicação.

- Qual a correta sequência de ações para cada tarefa e como é descrita?

Para cada tarefa, deve haver uma descrição de como se espera que o usuário veja a tarefa antes de aprender sobre a interface. Essas ações podem incluir uma mensagem como “Autenticar no Sistema”.

- Qual a interface definida?

Precisa descrever *prompts* que procedem cada ação requerida para completar as tarefas que estão sendo analisadas, como por exemplo, o tempo de resposta e cores.

Quadro 7 - Definição Das Entradas Para o Percurso Cognitivo.
Fonte: Rocha; Baranauskas (2003, P. 188).

A fase de análise cognitiva consiste em examinar cada ação do caminho da solução e tentar contar uma estória verossímil de como o usuário iria escolher aquela ação. As características críticas da interface, são aquelas que proveem uma ligação entre a descrição do usuário para a tarefa e a ação correta, e aquelas que proveem *feedback* indicando o efeito da ação do usuário (ROCHA; BARANAUSKAS 2003).

Rocha e Baranauskas (2003) afirmam que os analistas ao contarem suas estórias devem responder as quatro questões citadas no Quadro 8.

- Os usuários farão a ação correta para atingir o resultado desejado?

Suponha que em uma determinada aplicação antes de mandar imprimir um documento é preciso selecionar determinada impressora. O usuário irá saber que tem que fazer isso antes de executar a tarefa de impressão?

- Os usuários perceberão que a ação correta está disponível?

Se a ação estiver disponível no menu e for facilmente identificada não há problema.

- Os usuários irão associar a ação correta com resultado desejado?

Se existe um item menu claro e facilmente encontrado informando “selecionar impressora” não há problemas.

- Se a ação correta for executada os usuários perceberão que foi feito um progresso em relação à tarefa desejada?

Se após a seleção o usuário tiver um feedback informando “impressora X selecionada” não há problemas.

Quadro 8 – Questões a Responder no Percurso Cognitivo.
Fonte: Rocha; Baranauskas (2003, P. 189).

2.5.3 Teste de Interação e Usabilidade

O objetivo dos testes é avaliar a qualidade das interações, de modo que problemas sejam detectados, assim como seus impactos. Os testes de interação envolvem o uso do sistema por usuários reais ou representativos, na realização de tarefas em um contexto real ou simulado. Em ambos os casos, existe a presença de fatores como a expressão do usuário, especificação do local de teste, resultados esperados e limitações (CYBIS, BETIOL e FAUST, 2010).

Os testes, geralmente, são realizados em dois tipos de ambiente, em um laboratório de testes ou no próprio ambiente de trabalho, se isso for apropriado ao contexto do *software* (CYBIS, BETIOL e FAUST, 2010). A especificação em um teste é o momento em que o usuário expõe seus pensamentos referentes às ações propostas nos testes, essas ações podem ocorrer durante ou após os testes de interação.

Os resultados dos testes podem ser qualitativos ou quantitativos, bem como confirmar comportamentos esperados ou também apontar comportamentos diferentes da expectativa.

3 MATERIAL E MÉTODOS

Neste capítulo são apresentados os materiais e métodos utilizados para o desenvolvimento da aplicação experimental.

3.1 FERRAMENTAS

O protótipo *web* de controle de salas foi desenvolvido por Agnes e Bazilio (2013), como trabalho final da disciplina de Padrões de Projetos. Este trabalho foi utilizado para aplicar o estudo experimental de usabilidade. As seguintes ferramentas e metodologias foram utilizadas para o seu desenvolvimento:

- Linguagem Java;
- Framework JSF;
- Framework JPA;
- Sistema Operacional Windows Seven;
- Astah Community;
- Sistema de Gerenciamento de Banco de Dados MySQL;
- NetBeans 7.4;
- Servidor Web GlassFish 4.0.

3.2 ARQUITETURA

Das sessões 3.2.1 até 3.2.5 será apresentada a visão geral, os diagramas de caso de uso, diagramas de sequencia, e diagramas de classe referente ao protótipo de software desenvolvido.

3.2.1 Visão Geral

O objetivo do sistema e gerenciar as reservas de salas, bem como os usuários, tendo o controle dos níveis de prioridade dentro da estrutura hierárquica do sistema, permitindo que determinado usuário somente execute o que seu nível lhe permita. A reserva somente poderá ser feita por um usuário previamente cadastrado no sistema. O Responsável Geral é o único usuário que poderá inserir e excluir outros usuários. O cadastro de salas e feito pelo Responsável Geral e os Responsáveis por cada setor terão a permissão para modificar as salas. Para que uma reserva seja feita a sala desejada deverá estar disponível na data determinada pelo requerente (usuário). O usuário poderá inserir e consultar qualquer reserva, e excluir somente a sua reserva. Havendo algum problema com relação à reserva de uma determinada sala, o Responsável Setor poderá excluir as de seu setor.

Quadro 9 - Visão Geral
Fonte: AGNES; BAZILIO (2013).

3.2.2 Requisitos do Sistema

A partir do Quadro 10 até o Quadro 23 podem ser verificados os requisitos do *software*.

RF. 1 Cadastrar Usuário	Evidente (X)			
Descrição: Somente o responsável geral poderá cadastrar os usuários no sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
1.1 obter dados para cadastro	Nome do usuário, e-mail e senha.	Interface	( )	(x)
1.2 Cadastrar dados	Verificar a existência do usuário no banco de dados.	Segurança	( )	(x)

Quadro 10 - RF Cadastrar usuários
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 2 Excluir cadastro de Usuário	Evidente (X)			
Descrição: Somente o responsável geral poderá excluir os usuários no sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
2.1 Obter dados	Identificação do usuário	Interface	( )	(x)
2.2 Excluir Sala	Verificar se o usuário é cadastrado no sistema.	Segurança	( )	(x)

Quadro 11 - RF Excluir Cadastro de Usuário
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 3 Consultar Usuário	Evidente (X)			
Descrição: Todos os usuários poderão efetuar a consulta no sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
3.1 Obter dados	Identificação do usuário	Interface	( )	(x)
3.2 Excluir Sala	Verificar se o usuário é cadastrado no sistema.	Segurança	( )	(x)

Quadro 12 - RF Consultar Usuário
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 4 Cadastro de Salas	Evidente (X)			
Descrição: Somente o responsável geral poderá cadastrar salas no sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
4.1 Obter dados	Descrição da sala e seus equipamentos	Interface	( )	(x)

Quadro 13 - RF Cadastro de Salas
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 5 Modificar Cadastro de Salas	Evidente (X)			
Descrição: Somente o responsável geral e o responsável de setor poderão modificar salas no sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
5.1 Obter dados	Identificação da sala, campo desejado e novo dado	Interface	( )	(x)

Quadro 14 - RF Modificar Cadastro de Sala
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 6 Excluir Sala	Evidente (X)			
Descrição: Somente o responsável geral poderá excluir as salas do sistema				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
6.1 Obter dados	Identificação da sala	Interface	( )	(x)
6.2 Excluir Sala	Verificar se a sala é cadastrada no sistema	Segurança	( )	(x)

Quadro 15 - RF Excluir Sala
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 7 Consultar Salas	Evidente (X)			
Descrição: Todos os usuários poderão efetuar a consulta				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
7.1 Obter dados	Identificação da sala	Interface	( )	(x)
7.2 Consultar Sala	Verificar se a sala é cadastrada no sistema	Segurança	( )	(x)

Quadro 16 - RF Consultar Salas
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 8 Cadastrar Reserva Salas	Evidente (X)			
Descrição: Somente o professor poderá reservar uma sala				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
8.1 Obter dados	Identificação da sala, identificação do usuário, data, hora de inicio e hora de fim	Interface	( )	(x)
8.2 Cadastrar Reserva	Verificar se a sala é cadastrada no sistema e se a mesma já possui uma reserva na data	Segurança	( )	(x)

Quadro 17 - RF Cadastrar Reserva de Salas
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 9 Excluir Reserva de Sala	Evidente (X)			
Descrição: O Responsável Geral poderá excluir qualquer reserva, o responsável do setor poderá excluir reserva do seu setor, o professor poderá excluir suas reservas				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
9.1 Obter dados	Identificação da reserva e identificação do usuário	Interface	( )	(x)
9.2 Excluir Reserva	Verificar se a reserva é cadastrada no sistema	Segurança	( )	(x)

Quadro 18 - RF Excluir Reserva de Sala
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 10 Consultar Reserva de Salas	Evidente (X)			
Descrição: Todos os usuários poderão efetuar a consulta				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
10.1 Obter dados	Identificação da reserva de sala	Interface	( )	(x)
10.2 Consultar Sala	Verificar se a reserva é cadastrada no sistema	Segurança	( )	(x)

Quadro 19 - RF Consultar Reserva de Salas
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 11 Cadastrar Equipamentos de Salas	Evidente (X)			
Descrição: Somente o responsável geral poderá cadastrar equipamentos em qualquer sala, o responsável de setor poderá cadastar equipamentos nas salas de seu setor.				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
11.1 Obter dados	Identificação do equipamento, identificação da sala, tipo.	Interface	( )	(x)
11.2 Cadastrar Reserva	Verificar se o identificador do equipamento já não foi cadastrado no sistema.	Segurança	( )	(x)

Quadro 20 - RF Cadastrar Equipamentos de Salas
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 12 Consultar Equipamentos	Evidente (X)			
Descrição: Todos os usuários poderão efetuar a consulta				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
12.1 Obter dados	Identificação dos equipamentoses	Interface	( )	(x)
12.2 Consultar Equipamentos	Verificar se o equipamento é cadastrada no sistema	Segurança	( )	(x)

Quadro 21 - RF Consultar Equipamentos
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 13 Modificar Equipamentos	Evidente (X)			
Descrição: O responsável geral poderá modificar qualquer equipamento				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
13.1 Obter dados	Identificação do equipamento, campo desejado e novo dado	Interface	( )	(x)

Quadro 22 - RF Modificar Equipamentos
Fonte: Adaptado de AGNES e BAZILIO (2013)

RF. 14 Excluir Equipamentos de Sala	Evidente (X)			
Descrição: O Responsável Geral poderá excluir qualquer equipamento.				
Requisitos não Funcionais				
Nome:	Restrição	Categoria	Desejável	Permanente
14.1 Obter dados	Identificaçãodo equipamento	Interface	( )	(x)
14.2 Excluir Reserva	Verificar se o equipamento é cadastrada no sistema	Segurança	( )	(x)

Quadro 23 - RF Excluir Equipamentos de Sala
Fonte: Adaptado de AGNES e BAZILIO (2013)

3.2.3 Caso De Uso

A Figura 5 representa o diagrama de caso de uso do protótipo de software.

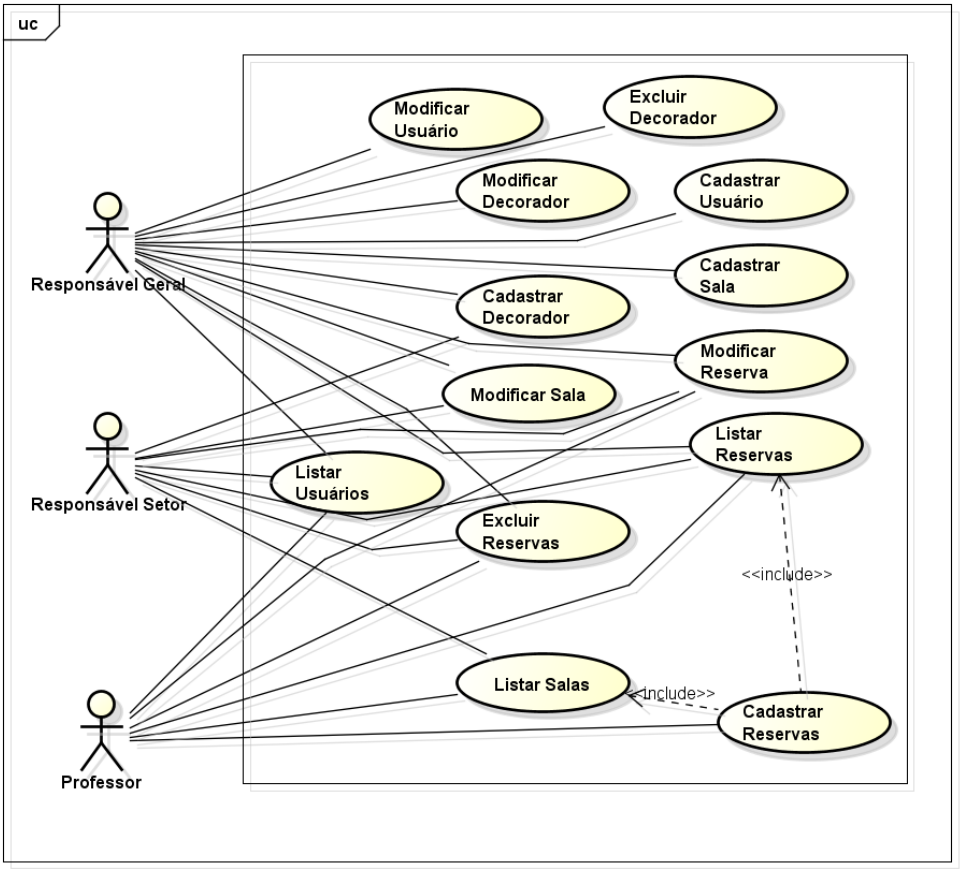

Figura 5 - Diagrama de Caso de Uso.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Nos Quadro 24 até Quadro 36 são apresentadas as descrições dos casos de uso referente à Figura 5.

Caso de Uso: Cadastrar Sala
Atores: Responsável Geral, Responsável Setor
Interessados: Setor
Pré-concições: Devem estar documentadas as características da sala, bem como seus componentes internos.
Pós-condições: A sala foi cadastrada.
Fluxo Principal: 1. O responsável do setor informa dados e características da sala. 2. [IN] O responsável geral registra a sala.
Tratamento de exceções: 1. O responsável não tem as informações da sala.

Quadro 24 - UC Cadastrar Sala.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Modificar Sala
Atores: Responsável Geral, Responsável Setor
Interessados: Setores
Pré-concições: A sala deve estar previamente cadastrada no sistema.
Pós-condições: Dados da sala alterados e salvos no banco de dados.
Fluxo Principal: 1. O responsável do setor ou responsável geral informa dados da sala a ser modificada. 2. [IN] O responsável geral ou responsável do setor altera dados da sala.
Tratamento de exceções: 1. A sala não está cadastrada no sistema.

Quadro 25 - UC Modificar Sala.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Listar Salas
Atores: Responsável Setor, Professor
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema.
Pós-condições: Lista de salas.
Fluxo Principal: 1. [IN] Selecionar buca de salas. 2. [OUT] Lista de salas.
Tratamento de exceções:

Quadro 26 - UC Listar Salas.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Cadastrar Usuário
Atores: Responsável Geral
Interessados: Setores
Pré-concições: O usuário deve ser um funcionário da Universidade dentro destes cargos: Responsável Geral, Responsável Setor, e Professor.
Pós-condições: Usuário cadastrado.
Fluxo Principal: 1. O responsável geral informa dados do usuário. 2. [IN] O responsável geral registra o usuáiro sala.
Tratamento de exceções: 1. O responsável não tem as informações do usuário.

Quadro 27 - UC Cadastrar Usuário.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Modificar Usuário
Atores: Responsável Geral
Interessados: Setores
Pré-concições: O usuário deve estar previamente cadastrada no sistema.
Pós-condições: Dados do usuário alterados e salvos no banco de dados.
Fluxo Principal: 1. O responsável geral informa dados do usuário a ser modificada. 2. [IN] O responsável geral altera dados do usuário.
Tratamento de exceções: 1. O usuário não está cadastrada no sistema.

Quadro 28 - UC Modificar Usuário.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Listar Usuários
Atores: Responsável Geral, Responsável Setor, Professor
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema.
Pós-condições: Lista de pessoas.
Fluxo Principal: 1. [IN] Selecionar buca de pessoas. 2. [OUT] Lista de pessoas.
Tratamento de exceções:

Quadro 29 - UC Listar Pessoas.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Cadastrar Reservas
Atores: Responsável Geral, Responsável Setor, Professor
Interessados: Usuários
Pré-concições: A sala deve estar previamente cadastrada no sistema, o usuário que solicita a reserva deve estar cadastrado no sistema, e a sala a ser reservada não pode estar em outra reserva.
Pós-condições: Reserva Registrada.
Fluxo Principal: 1. [IN] O professor informa os dados da sala a ser reservada. 2. [OUT] Reserva efetuada
Tratamento de exceções: 1. A sala não está cadastrada no sistema. 2. O usuário solicitante não está cadastrado no sistema. 3. A sala solicitada já esta reservada.

Quadro 30 - UC Cadastrar Reservas.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Modificar Reservas
Atores: Responsável Geral, Responsável Setor, Professor
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema, O usuário deve informar os dados da reserva
Pós-condições: Reserva modificada.
Fluxo Principal: 1. O responsável do setor informa os dados da reserva a ser modificada. 2. [IN] O responsável do setor informa altera dados da reserva.
Tratamento de exceções: 1. A reserva não está cadastrada no sistema.

Quadro 31 - UC Modificar Reservas.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Listar Reservas
Atores: Responsável Geral, Responsável Setor, Professor
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema.
Pós-condições: Lista de reservas.
Fluxo Principal: 1. [IN] Selecionar buca de reservas. 2. [OUT] Lista de reservas.
Tratamento de exceções:

Quadro 32 - UC Listar Reservas.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Excluir Reserva
Atores: Responsável Geral, Responsável Setor, Professor
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema, O usuário deve informar os dados da reserva
Pós-condições: Reserva liberada.
Fluxo Principal: 1. [IN] Dados da reserva. 2. [OUT] Reserva liberada.
Tratamento de exceçoes: 1. A reserva não está cadastrada no sistema.

Quadro 33 - UC Excluir Reserva.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Cadastrar Equipamento
Atores: Responsável Geral, Responsável Setor,
Interessados: Usuários
Pré-concições: O usuário que solicita a reserva deve estar logado no sistema
Pós-condições: Equipamento Registrado.
Fluxo Principal: 1. [IN] O responsável geral ou do setor informa os dados do equipamento a ser cadastrado.
Tratamento de exceçoes: 1. O responsável não tem as informações do equipamento.

Quadro 34 - UC Cadastrar Equipamento.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Modificar Equipamento
Atores: Responsável Geral
Interessados: Usuários
Pré-concições: O equipamento deve estar previamente cadastrado no sistema, o usuário que solicita a reserva deve estar logado no sistema
Pós-condições: Equipamento Modificado.
Fluxo Principal: 1. [IN] Dados do equipamento. 2. [IN] Novos dados do equipamento. 3. [OUT] Equipamento modificado.
Tratamento de exceçoes: 1. O equipamento não está cadastrado.

Quadro 35 - UC Modificar Equipamento.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Caso de Uso: Excluir Equipamento
Atores: Responsável Geral
Interessados: Usuários
Pré-concições: O usuário deve estar logada no sistema, O usuário deve informar os dados do equipamento
Pós-condições: Equipamento Excluído.
Fluxo Principal: 1. [IN] Dados do equipamento. 2. [OUT] Equipamento excluído.
Tratamento de exceçoes: 1. O equipamento não está cadastrada no sistema.

Quadro 36 - UC Excluir Equipamento.
Fonte: Adaptado de AGNES e BAZILIO (2013).

3.2.4 Diagrama de Atividades

A Figura 6 representa o diagrama de atividade referente ao protótipo de software desenvolvido, o diagrama apresenta a seguinte funcionalidade:

1) Usuário informa *login* e senha;
2) Sistema valida o *login*;
3) Sistema verifica categoria do usuário;
4) Sistema busca lista de reserva para o usuário (se for o caso);
5) Sistema mostra lista de reservas (se for o caso);
6) Sistema apresenta menu;
7) Usuário seleciona ação;
8) Sistema valida permissão através do *design pattern proxy*;
9) Sistema trata dados referentes à ação selecionada ou retorna página de erro.

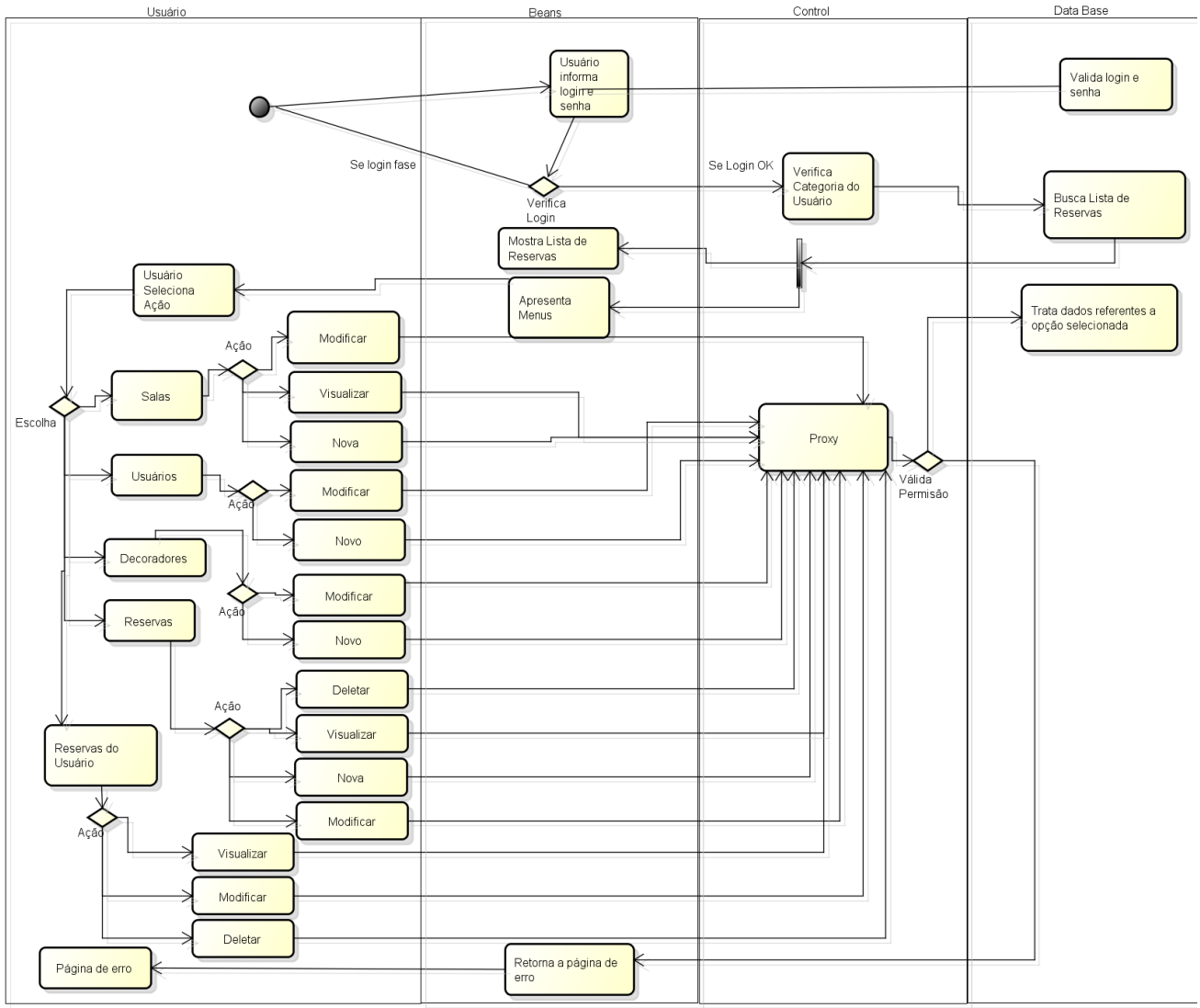

Figura 6 - Diagrama de Atividade
Fonte: Adaptado de AGNES e BAZILIO (2013)

3.2.5 Diagrama de Classes

A Figura 7 expõe o diagrama de classes do protótipo desenvolvido, são apresentadas as classes referentes ao gerenciamento do sistema, a estrutura de classes referente às salas do sistema, e exibido as classes referentes a reservas no sistema e estrutura de pessoas.

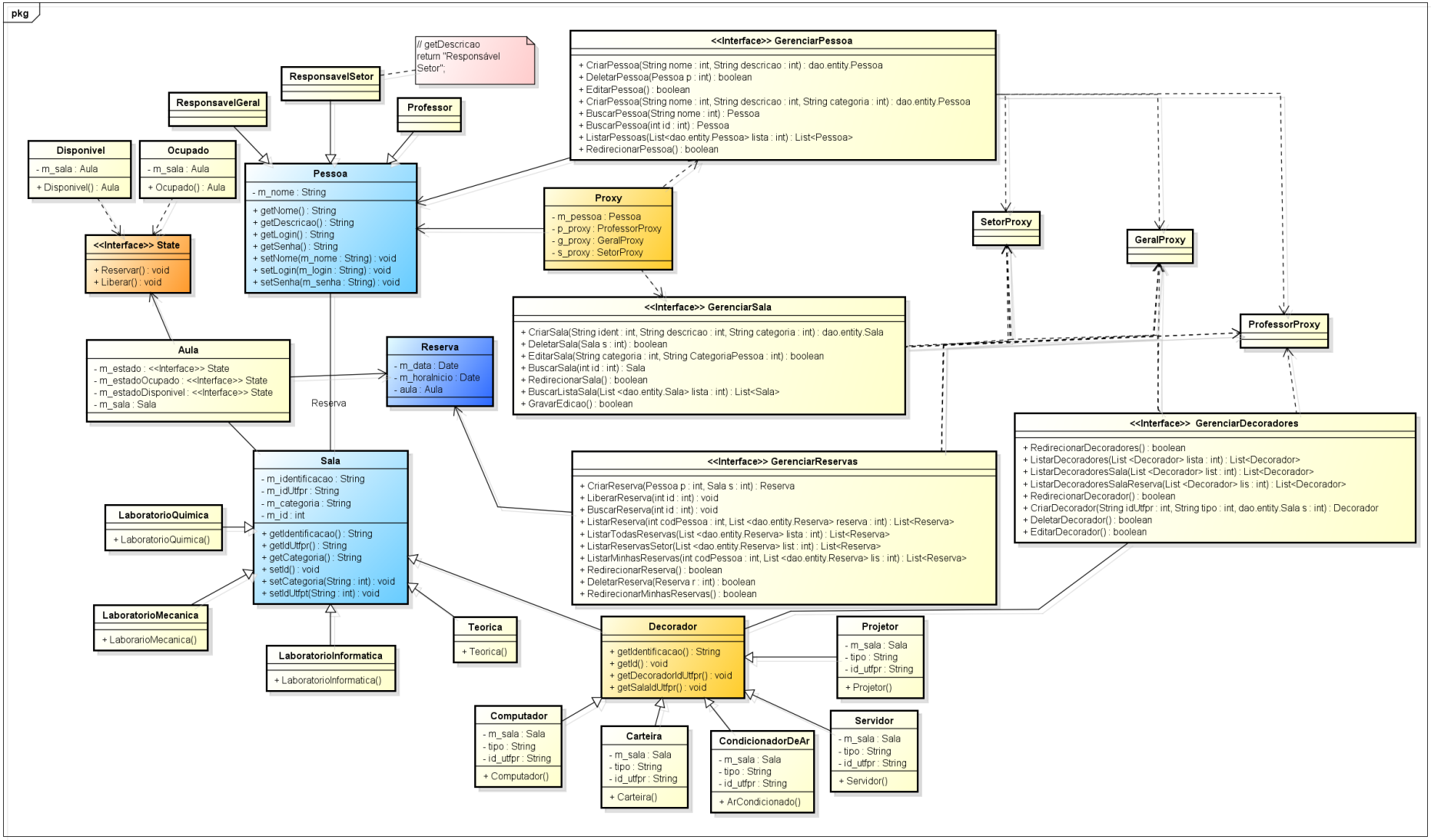

Figura 7 - Diagrama de Classes
Fonte: Adaptado de AGNES e BAZILIO (2013)

3.3 CRITÉRIOS DE AVALIÇÃO

Para realizar a avaliação de usabilidade e IHC, foram utilizadas metodologias de inspeção e cognição.

Utilizando a metodologia de inspeção, em que o avaliador aplica um conjunto de questões específicas a si mesmo, assumindo o perfil de usuário. O objetivo desta metodologia é avaliar a interface do sistema, em busca de aspectos que possam aumentar as chances de erros, diminuir entendimentos e dificuldades de navegação e acesso (JUNQUEIRA; SILVA, 2010).

Para avaliar por inspeção o autor assume a posição de usuário do sistema, como avaliador.

A partir do Quadro 37 até o Quadro 42 serão apresentadas estórias de sucesso e estórias de fracasso referente a inspeção de usabilidade utilizando percurso cognitivo.

Um chefe de sala experiente em Windows inicia uma tarefa dando um clique no botão ”logar” Defesa da Credibilidade • O usuário clica no botão “logar” da aplicação porque ele sabe que é preciso estar autenticado no sistema para utiliza-lo. • Usuário conhece por experiência que é preciso estar autenticado no sistema. • Usuário sabe por experiência que o clique é a ação a ser usada. • Mudanças na tela ou nos menus sinalizam o início da aplicação

Quadro 37 - Estória de sucesso ao clicar no botão logar.
Fonte: Autoria Própria.

Um professor experiente em Windows abre uma tela do sistema para fazer uma reserva de sala Defesa da credibilidade? • Usuário está tentando reservar uma sala. • Usuário clica em Salas pois o título é claramente relacionado com o que o usuário quer fazer • Usuário sabe que irá listar as salas, por experiência. • Usuário sabe que tudo está indo bem quando da leitura das opções do menu

Quadro 38 - Estória de sucesso ao clicar no botão sala para realizar uma reserva.
Fonte: Autoria Própria.

Com esses exemplos podem ser resumidos os quatro pontos que os analistas consideraram em cada passo e suas características mais relevantes.

1. Usuários irão conhecer qual resultado querem alcançar:
    - Porque é parte da tarefa original, ou
    - Porque eles tem experiência no uso do sistema, ou
    - Porque o sistema diz a eles o que devem fazer.
2. Usuários irão saber que uma ação está disponível:
    - Por experiência, ou
    - Observando algum dispositivo, ou
    - Observando a representação de uma ação.
3. Usuários irão saber qual ação é adequada para o resultado que estão tentando obter:
    - Por experiência, ou
    - Porque a interface provê um *prompt* ou rótulo que conecta a ação ao que ele está tentando fazer, ou
    - Porque as outras ações não parecem corretas.
4. Usuários irão saber que as coisas estão indo bem depois da ação:
    - Por experiência, ou
    - Por reconhecer a conexão entre a resposta do sistema e o que está tentando fazer.

Estórias de sucesso requerem sucesso em todos os quatro critérios de análise, enquanto estórias de fracasso tipicamente falham em apenas um dos quatro critérios. Seguem quadros relatando estórias de fracasso.

Um professor experiente em Windows tenta clicar no botão Salas sem estar autenticado no sistema. Estória de fracasso: O usuário irá acionar o botão Salas, pois é o botão que lista as salas para reserva, o usuário não obtém o resultado esperado, pois não está autenticado no sistema.

Quadro 39 - Estória de fracasso ao clicar no botão salas sem estar autenticado.
Fonte: Autoria Própria.

Um professor experiente em Windows tenta clicar no botão adicionar no menu salas. Estória de Fracasso: O Usuário professor não tem permissão para adicionar salas, e é retornado uma mensagem que ele não tem permissão para executar tal ação;

Quadro 40 - Estória de fracasso ao clicar no botão adicionar no menu salas, usuário sem permissão.
Fonte: Autoria Própria

Um Chefe de sala experiente em Windows deseja adicionar equipamentos para acrescentar nas salas Estória de Fracasso: O usuário não encontra a opção, pois o botão se chama decoradores e nenhuma descrição aparece ao posicionar o cursor do mouse sobre este.

Quadro 41 - Estória de fracasso, não é possível identificar que o botão decoradores é o botão responsável pela adição de equipamentos.
Fonte: Autoria Própria.

Um responsável geral experiente em Windows ao clicar em reservas deseja cancelar a reserva de uma sala. Estória de Fracasso: O usuário não irá encontrar o botão cancelar, pois o cancelamento de salas pelo responsável geral não foi tratado pelo sistema.

Quadro 42 - Estória de fracasso, botão excluir reserva não aparece para o responsável geral, sistema não trata esse tipo de situação.
Fonte: Autoria Própria.

Com esses exemplos pode ser observado que as falhas são tipicamente associadas com um dos quatros critérios de avaliação, sugestões gerais como corrigi-las também podem ser organizadas da mesma forma. Seguem os critérios:

1. Os usuários farão a ação correta para atingir o resultado desejado?

Se a interface falha nesse ponto, o usuário não está fazendo a coisa certa, existem três opções para tentar corrigir:

- A ação precisa ser eliminada, ou deve ser combinada com outra ação mais adequada.
- Um *prompt* deve ser dado ao usuário informando sobre qual ação deve ser feita.
- Alguma outra parte da tarefa precisa ser mudada de forma que o usuário entenda a necessidade da ação, talvez porque passe a ser consistente com alguma parte da sequência de ações.

2. Os usuários perceberão que a ação correta está disponível?

Se o usuário tem os objetivos corretos mas não sabe que a ação está disponível na interface a solução é associar a ação a um controle mais óbvio.

3. Os usuários irão associar a ação correta com o efeito desejado?

Para corrigir essa falha é preciso conhecer os usuários e a forma como descrevem a tarefa. Com essa informação é possível desenvolver rótulos e

descrição para ações que incluam palavras que os usuários frequentemente usam para descrever suas tarefas.

4. Se a ação correta for executada os usuários perceberão que foi feito um progresso em relação à tarefa desejada?

Claramente, na maioria das situações qualquer *feedback* é melhor que nenhum, um *feedback* que indica o que aconteceu é melhor do que um que indica alguma coisa que aconteceu. Respostas são sempre mais efetivas quando usam termos relacionados à descrição do usuário para a tarefa.

4 RESULTADOS E DISCUSSÕES

Uma interface simples e fácil de entender auxilia o entendimento dos usuários, para melhorar a interface do protótipo foi modificado o campo de *login,* colocando uma condição que exige a autenticação do usuário no sistema para exebir os botões de navegação do sistema. A Figura 8 exibe como era a tela de *login* antes da alteração.

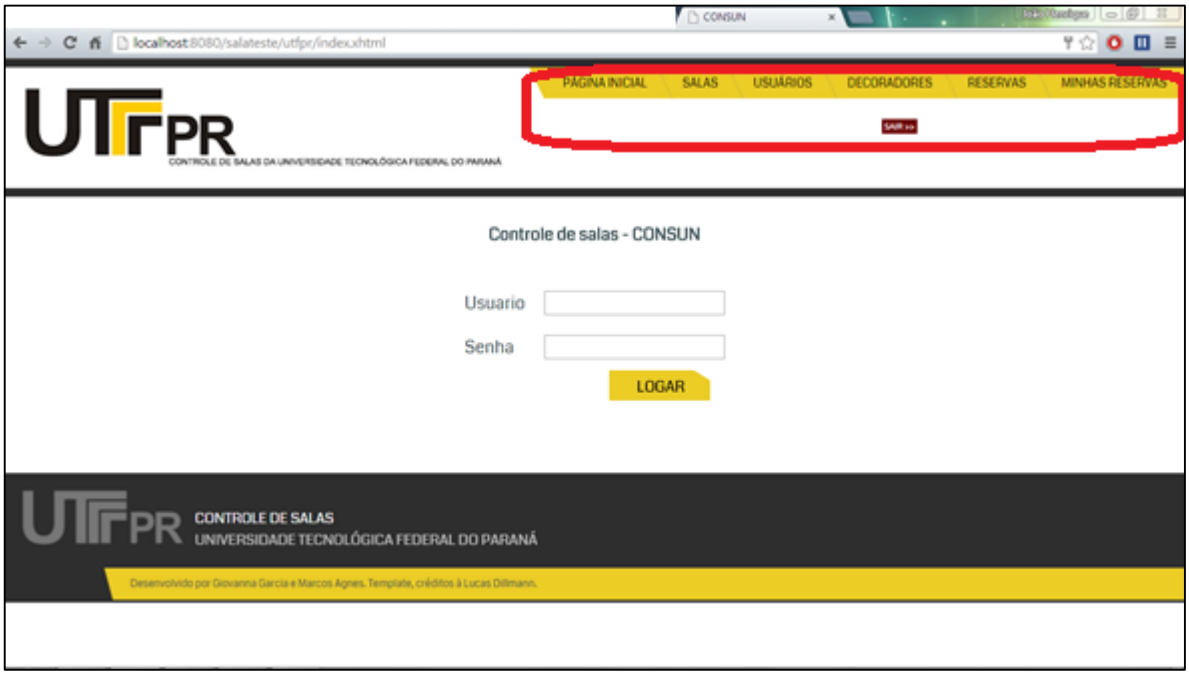

Figura 8 - Tela de Login - Antes da alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Para efetuar as alterações do *login,* citadas no parágrafo anterior foi editado o arquivo `template.xhtml`, editando a linha que continha o *form* dos botões adicionando uma condição de *rendered*, desta forma o arquivo passou de `<h:form>` para `<h:form rendered="#{usuarioControlador.verificalogado==true}">` e também foi editado o *CommandLink* o Quadro 43 apresenta o código antes da alteração e o Quadro 44 apresenta o código após a alteração. Desta forma o sistema exige que o usuário esteja autenticado para exibir os botões de navegação.

```
<h:commandLink styleClass="sair"
               action="#{usuarioControlador.deslogar}" value="Sair" />
```

Quadro 43 - CommandLink Sair antes da alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

```
<h:commandLink styleClass="sair"
               action="#{usuarioControlador.deslogar}" value="Sair"
               rendered="#{usuarioControlador.verificalogado==true}"/>
```

Quadro 44 - CommandLink Sair após alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Na Figura 9 pode ser visualizado como ficou a tela inicial após as alterações citadas nos dois parágrafos anteriores.

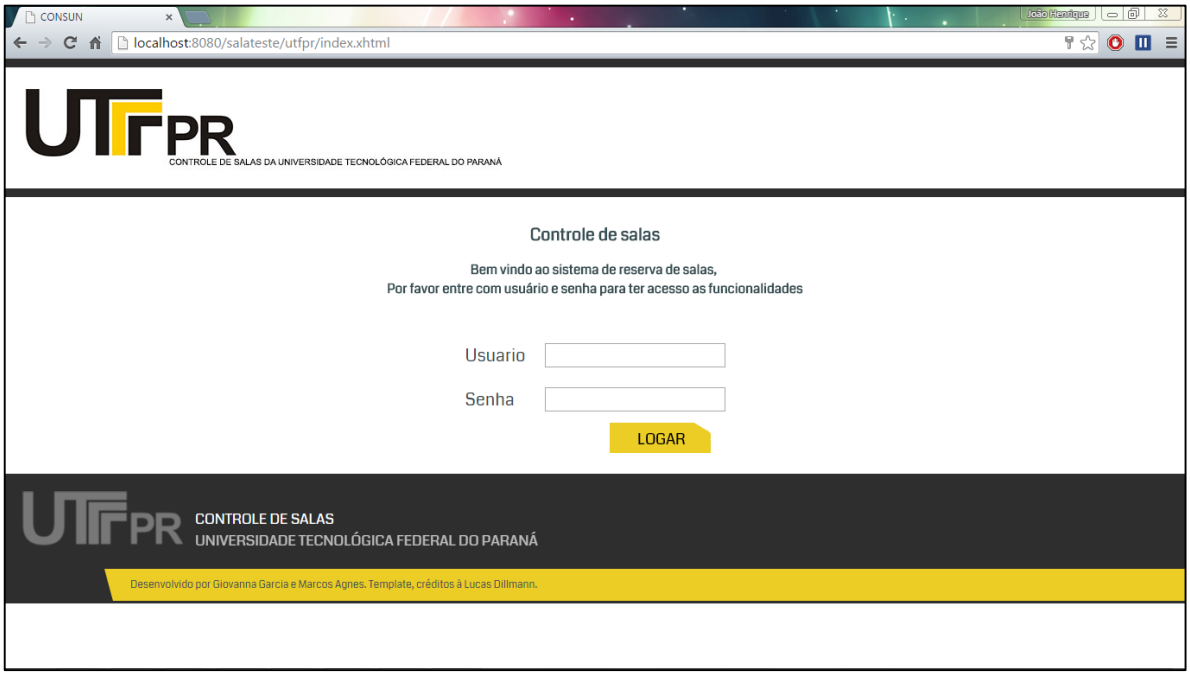

Figura 9 - Tela de Login - Após alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Como o sistema não tinha nenhuma validação para o estado do objeto, foi desenvolvido um método para executar o processo. no Quadro 45 é apresentado o código fonte do método utilizado para fazer a validação que foi apresentada na Figura 9.

```
public boolean verificaLogado() {

    Object o = ((HttpSession) FacesContext.getCurrentInstance().
            getExternalContext().getSession(true)).getAttribute("proxy");

    if (o == null) {
        return false;
    } else {
        return true;
    }
}
```

Quadro 45 - Método verificalogado utilizado para verificar se o usuário está autenticado no sistema.
Fonte: Adaptado de AGNES e BAZILIO (2013).

A importância da interface do usuário aponta que deve ser minimizada a memória do usuário, desta forma foi editado alguns itens impondo validação conforme a categoria do usuário. Se o usuário não tem permissão à determinada função ou botão do sistema não tem porque ele ter acesso a tal, para isso foi editado o arquivo `salas.xhtml`. A regra já definida no sistema aponta que podem existir até três tipos de usuários: 1 – professor, 2 – responsável do setor, 3 – responsável geral. O Professor só pode realizar reserva de salas, porém como mostra a Figura 10 o botão Adicionar está aparecendo para um professor que é o José neste caso, para corrigir isso será feito uma validação no qual só o responsável geral possa ver este botão.

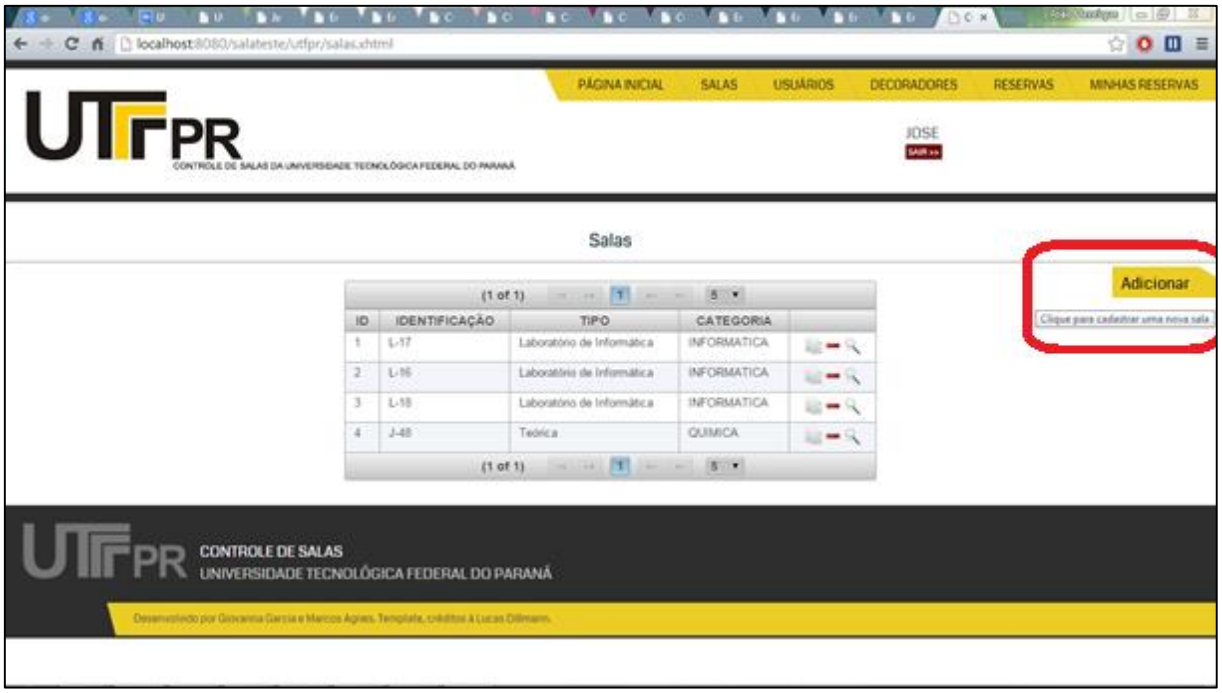

Figura 10 – Tela de Salas.
Fonte: AGNES e BAZILIO (2013).

Para validar o botão foi acrescentado o *rendered* no *commandbutton* do arquivo `salas.xhtml`, passando o arquivo do Quadro 46 à ficar como o arquivo do Quadro 47. Para essa alteração ainda foi necessário adicionar o método `retornacategoria` na classe `UsuarioControlador`, tal método está expresso no Quadro 48.

```
<h:commandButton value="Nova Sala" action="#{redirecionador.novaSala}"
                 image="res/css/img/add.png"
                 title="Clique para cadastrar uma nova sala" />
```

Quadro 46 - *CommandButton* antes da alteração
Fonte: Adaptado de AGNES e BAZILIO (2013)

```
<h:commandButton value="Nova Sala" action="#{redirecionador.novaSala}"
                 image="res/css/img/add.png"
                 title="Clique para cadastrar uma nova sala"
                 rendered="#{usuarioControlador.retornarcategoria()==3}"/>
```

Quadro 47 - *CommandButton* após alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

```
public int  retornarCategoria() {
    Proxy p = (Proxy) ((HttpSession) FacesContext
              .getCurrentInstance().getExternalContext()
              .getSession(true)).getAttribute("proxy");
     return p.getUsuario().getTipo();
}
```

Quadro 48 - Método retorna categoria
Fonte: Autoria Própria

Na Figura 11 pode ser verificado como ficou a tela de salas após a aplicação do método de validação por usuário.

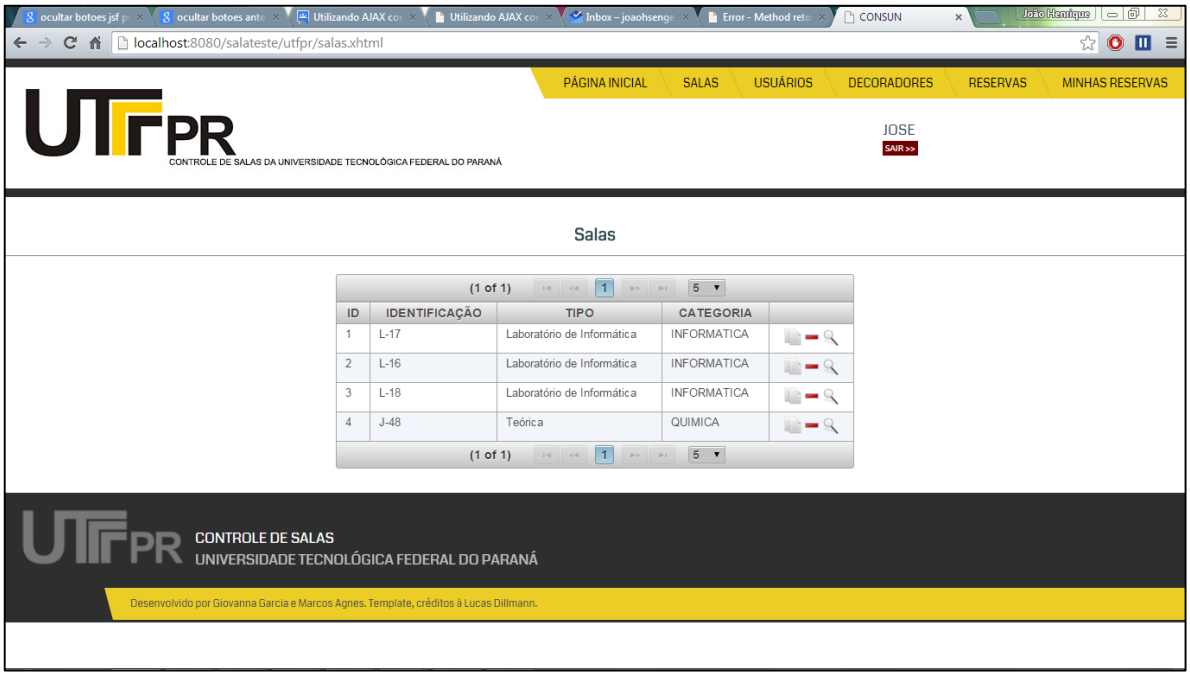

Figura 11 – Tela de Salas após alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Para garantir a usabilidade o sistema deve oferecer facilidade de aprendizado, para isso pode ser ofertado componentes visuais no sistema e mais informações, sendo claro e objetivo. Na página inicial do sistema de controle de salas existia um botão chamado decoradores, como pode ser visualizado na Figura 12, porém não é possível identificar o que o mesmo faz. Por isso esse nome foi modificado para Equipamentos como poder ser visualizado na Figura 13.

Figura 12 - Botão decorador antes da alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Para melhorar a compreensão do usuário foi editado o arquivo `template.xhtml` e alterado o nome de Decoradores para Equipamentos, além de adicionar uma informação no botão que é exibida quando o usuário leva o mouse em cima deste, o Quadro 49 mostra como era o código antes da alteração.

```
<li><h:commandLink value="Decoradores" action="#{redirecionador.verDecoradores}"/></li>
```

Quadro 49 - CommandLink decoradores antes da alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

No Quadro 50 pode ser verificado como foi adicionado a informação citada anteriormente.

```
<li><h:commandLink value="Equipamentos" action="#{redirecionador.verDecoradores}"
                   title="Clique para Adicionar Um Equipamento na Sala.
                   Ex: Projetor; Computador; Carteira"/></li>
```

Quadro 50 - CommandLink decoradores após alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

A Figura 13 mostra como ficou a informação do botão ao passar em cima deste com o ponteiro do mouse, após efetuar a alteração.

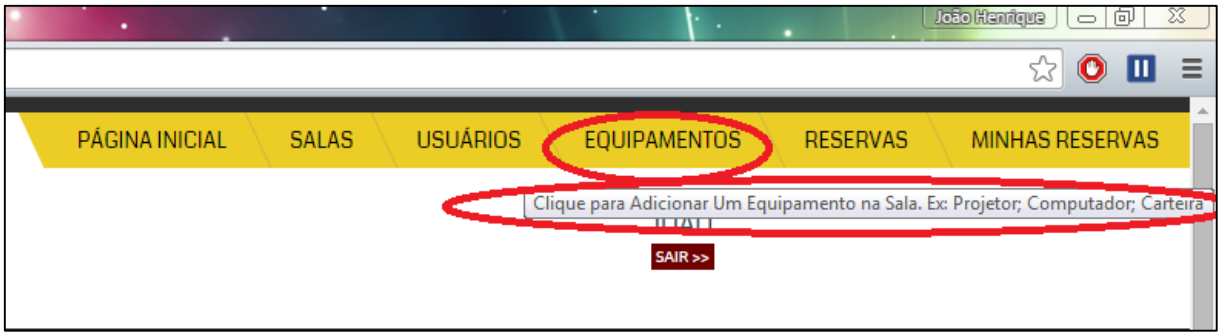

Figura 13 – Alteração Botão Decoradores para Equipamentos.
Fonte: Adaptado de AGNES e BAZILIO (2013).

A interação do sistema e a flexibilidade são essenciais para garantida da usabilidade, parte disso pode ser considerado o cancelamento de reservas pelo responsável geral, que até então não era possível no sistema, como pode ser visualizado na Figura 14 o responsável geral só tinha acesso a lista de reservas.

Figura 14 - Listagem de Reservas Antes da Alteração.
Fonte: Adaptado de AGNES e BAZILIO (2013).

Para que o cancelamento fosse possível foi adicionado a `Column` exibida no Quadro 51 no arquivo `template.xhtml`,esta `Column` exibe o botão cancelar junto a lista de reservas..

```
<p:column>
        <h:commandButton action="#{reservaControlador.deletarReserva(item)}"
                         image="res/css/img/list-remove.png"/>
</p:column>
```

Quadro 51 - Column Deletar Reserva.
Fonte: Autoria Própria.

O Quadro 52 exibe o método deletarReserva que foi desenvolvido para que o responsável geral tivesse permissões de deletar reservas, como já foi mostrado no quadro de requisitos essa exclusão deve ser possível, pois caso um professor deixe de fazer parte do grupo docente e tenha alguma reserva, pode ser necessário a exclusão ou cancelamento desta.

```
public void deletarReserva(src.modelo.concreto.Reserva r) {
    p = getProxy();
    if (p.deletarReserva(r)) {
        if (IdAula.dataAula(new Date(), r.getM_aula().getM_id(), r.getData())) {
            Reserva rDao = new Reserva();
            rDao.setAula(IdAula.converterInt(r.getM_aula().getM_id()));
            rDao.setCod(r.getCodigo());
            rDao.setData(r.getData());
            Pessoa p = new Pessoa();
            p.setCod(r.getPessoa().getId());
            p.setDataCadastro(r.getPessoa().getM_inicio());
            p.setNome(r.getPessoa().getM_nome());
            p.setTipo(TipoPessoa.gambi2(r.getPessoa().getDescricao()));
            rDao.setPessoaCod(p);
            dao.entity.Sala sa = new dao.entity.Sala();
            sa.setCod(r.getSala().getId());
            sa.setIdSala(r.getSala().getIdentificacao());
            sa.setIdUtfpr(r.getSala().getIdUtfpr());
            sa.setCategoria(r.getSala().getCategoria());

            switch (r.getSala().getIdentificacao()) {
                case "Laboratório de Informática":
                    sa.setTipo("LAB_INFORMATICA");
                    break;
                case "Laboratório de Química":
                    sa.setTipo("LAB_QUIMICA");
                    break;
                case "Teórica":
                    sa.setTipo("TEORICA");
                    break;
                case "Laboratório de Mecânica":
                    sa.setTipo("LAB_MECANICA");
                    break;
            }
            rDao.setSalaCod(sa);
            ejbFacadeReserva.remove(rDao);
        } else {
            new Redirecionador().erro("Esta aula não pode mais ser removida.");
        }

    } else {
        switch (p.getUsuario().getTipo()) {
            case 1:
            case 2:
                new Redirecionador().erro("Reserva: " + r.getCodigo() + " erro ao deletar");
                break;
            case 3:
                new Redirecionador().erro("Você não tem permissão para deletar resservas.");
                break;
        }
    }
}
```

Quadro 52 - Método deletarReserva.
Fonte: Autoria Própria.

A Figura 15 exibe como ficou a listagem de reservas após efetuar a alteração.

Figura 15 - Listagem de Reservas Após Alteração
Fonte: Adaptado de AGNES e BAZILIO (2013)

Pode-se verificar que com a alteração é permitido que o administrator remova uma reserva a partir da tela de reservas, desta forma a interface fica mais simples e fácil de trabalhar.

5 CONSIDERAÇÕES FINAIS

Este capítulo tem como objetivo apresentar as considerações finais e os trabalhos futuros deste projeto.

5.1 CONCLUSÃO

As metodologias de inspeção e cognição de IHC ajudaram a compreender em que podiam ser aplicadas melhorias da interface do *software*.

Por meio da avaliação foi identificado que o sistema de controle de salas possuía algumas dificuldades de navegação conforme a categoria do usuário, pois não havia tratamentos de usuários no menu de navegação, desta forma botões que o usuário não tem acesso eram exibidos, ocasionando erros de navegação.

O *software* necessitava de alguns ajustes na sua IHC, pois a interface é compreendida e manipulada pelos usuários, porém estava acarretando em dificuldades de navegação, permitindo acessos a telas as quais o usuário não deveria ter.

Foram aplicados ajustes na interface do sistema tornando-a mais simples e fácil de entender, validações a menus e botões também foram desenvolvidas de modo que fosse minimizada a memória do usuário e aumentado à facilidade de aprendizado, melhorando a compreensão, interação e flexibilidade do sistema, facilitando o uso do mesmo.

Após as alterações não foram identificados grandes problemas relacionados à usabilidade do sistema pelo avaliador, o sistema não exige uma grande quantidade de cliques do usuário e tem uma interface simples e fácil de compreender.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Para trabalhos futuros, é relevante a implantação do recurso de *web design* responsivo, com o recurso de *Media Queries* do CSS (*Cascading Style Sheets*) versão 3, que responde às solicitações *web*, de acordo com o dispositivo.

Pode ser avaliada a acessibilidade do sistema, de modo a verificar se a interface não impede ou atrapalhe usuários com algum tipo de limitação durante a realização da tarefa desejada.

Estudar outros métodos de avaliação de IHC e compará-los com os métodos utilizados.

REFERÊNCIAS BIBLIOGRÁFICAS

AGNES, M; BAZILIO, G. Documento de Requisitos: Controle de Salas. Trabalho Final da Disciplina de Padrões de Projetos do curso de Tecnologia em Análise e Desenvolvimento de Sistemas. **UTFPR**, Medianeira, 2013.

AMARO, A.; PÓVOA, A.; MACEDO, L. A Arte de Fazer Questionários. **UNISC**, 2004. Disponivel em: <http://www.unisc.br/portal/upload/com_arquivo/a_arte_de_fazer_questionario.pdf>. Acesso em: 09 Dezembro 2014.

BELMONTE, D. L.; SCANDELARI, L.; KOVALESKI, J. L. Novo ponto de vista na criação de sistemas com aplicação de modelos. **SIMPEP**, Bauru, 10 Novembro 2004.

CAMEIRA, R. E. A. componentização de Processos e de Sistemas: Impactos Metodológicos na Implantação de Sistemas Orientados por Processo. **ENEGEP**, Ouro Preto, Outubro 2003. Disponivel em: <http://www.abepro.org.br/biblioteca/ENEGEP2003_TR0903_0735.pdf>. Acesso em: 22 Novembro 2014.

CORTES, M. L.; CHIOSSI, T. S. **Modelos De Qualidade De Software**. [S.l.]: UNICAMP, 2001.

CYBIS, W. **Ergonomia e Usabilidade**. SÃO PAULO: NOVATEC, 2007.

CYBIS, W.; BETIOL, A. H.; FAUST, R. **Ergonomia e Usabilidade**. São Paulo: Novatec, 2010.

DIAS, C. **Usabilidade Na Web**. RIO DE JANEIRO: ALTA BOOKS, 2003.

FERREIRA, S. B. L.; LEITE, J. C. S. D. P. Avaliação da usabilidade em sistemas de informação: o caso do Sistema Submarino. **Scielo**, Curitiba, Junho 2003. Disponivel em: <http://www.scielo.br/scielo.php?pid=S1415-65552003000200007&script=sci_arttext>. Acesso em: 20 Outubro 2014.

INTHURN, C. **Qualidade e teste de software**. Florianópolis: Visual Books, 2001.

ISO. **ISO/IEC14598:1998**. Disponivel em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=24906>. Acesso em: 18 Setembro 2013.

JUNQUEIRA, S. D.; SILVA, B. S. D. **Interação Humano-Computador**. São Paulo: [s.n.], 2010.

KOSCIANSKI, A.; SOARES, M. D. S. **Qualidade de software**. [S.l.]: Novatec, 2007.

KRUG, S. **Não me faça pensar**. Rio de Janeiro: Alta Books, 2008.

MEMÓRIA, F. **Design para a internet:** projetando a experiência perfeita. São Paulo: Elsevier, 2014.

NIELSEN, J. **Usability Engineering**. San Diego: Academic Press, 1993.

PAGANINI, T. Usabilidade de interfaces para dispositivos móveis – parte I**. Tableless**, 2011. Disponivel em: <http://tableless.com.br/usabilidade-de-interfaces-para-dispositivos-moveis-parte1>. Acesso em: 12 Novembro 2014.

PETERS, J. F.; PEDRYCZ, W. **Engenharia de Software, Teoria e Prática**. [S.l.]: Campus, 2001.

PRATES, R. O.; BARBOSA, S. D. J. Avaliação de Interfaces de Usuário – Conceitos e. **URISAN**. Disponivel em: <http://www.urisan.tche.br/~paludo/material/IHM/Material/avaliacao.pdf>. Acesso em: set. 2014.

PRESSMAN, R. S. **Engenharia de software**. 6. ed. [S.l.]: McGraw-Hill, 2006.

PRESSMAN, R. S. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. Porto Alegre: MCGraw-Hill, 2011.

PRESSMAN, R. S.; LOWE, D. **Web Engineering:** A Practitioner's Approach. 1. ed. New York: McGraw-Hill, 2009.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, C. K. **Qualidade dos produtos de software:** teoria e prática. São Paulo: Prentice Hall, 2001.

ROCHA, H. V. D.; BARANAUSKAS, M. C. C. **Design E Avaliação De Interfaces Humono-Computador**. Campinas: UNICAMP, 2003.

SCHULMEYER, G. G.; MCMANUS, J. I. **Handbook of Software Quality Assurance**. 3. ed. [S.l.]: Prentice HalL, 1999.

SOMMERVILLE, I. **Software Engineering**. 9. ed. Boston: Addison-Wesley, 2011.

SUH, W. **Web Engineering - Principles and Techniques**. Londom: IGP, 2005.