UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
CURSO BACHARELADO EM SISTEMAS DE INFORMAÇÃO

PROPOSTA DE INTERFACE PARA A FERRAMENTA SGP – SISTEMA DE GERENCIAMENTO DE PROJETOS

DENIS COSTA PAIVA

PICOS-PI

2016

DENIS COSTA PAIVA

PROPOSTA DE INTERFACE PARA A FERRAMENTA SGP – SISTEMA DE GERENCIAMENTO DE PROJETOS

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação, Campus Senador Helvídio Nunes de Barros, da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharelado.

Orientador: Prof. Esp. Dennis Sávio Martins da Silva

PICOS-PI

2016

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

P149p Paiva, Denis Costa.

Proposta de interface para a ferramenta SGP – Sistema de Gerenciamento de Projetos / Denis Costa Paiva.– 2016.

CD-ROM : il.; 4 ¾ pol. (64 f.)

Monografia (Curso Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí, Picos, 2016.

Orientador(A): Prof. Esp. Dennis Sávio Martins da Silva

1. Sistema de Gerenciamento de Projetos-Interface. 2.Guia de Estilo. 3.*Design*-Interação. I. Título.

CDD 005.1

DENIS COSTA PAIVA

Monografia aprovado como exigência parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Data de Aprovação

Picos-PI, 15 de julho de 2016

Dennis Sávio Martins da Silva

Prof. Esp. Dennis Sávio Martins da Silva
Orientador

Alcilene Dalília de Sousa

Profª. Me. Alcilene Dalília de Sousa
Membro

Leonardo Pereira de Sousa

Prof. Esp. Leonardo Pereira de Sousa
Membro

Trabalho dedicado para minha família, especialmente a minha mãe Rejane Paiva. A minha namorada Roseângela Belo, pelo grande incentivo na minha vida. Dedico também a todos os meus amigos que me acompanharam nessa longa jornada.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por depositar forças inimagináveis que permitiram a realização deste trabalho.

A minha mãe, Rejane Paiva, pelo carinho, amor, afeto, paciência, ensinamentos, conselhos e apoio que me ajudou a continuar a seguir em frente para alcançar meus ideais. Sem a senhora na minha vida, não teria forças suficiente para continuar a progredir na vida.

A minha falecida tia, Iracélia Paiva, que sempre sonhou em me vê formado. Uma tristeza enorme de não poder compartilhar desse momento, mas sei que, de uma forma inexplicável, está aqui presente.

A minha namorada, Roseângela Belo, por compartilhar os momentos mais incríveis da minha vida. A única pessoa capaz de me fazer sorrir nos momentos mais tensos e frustrantes.

Ao meu orientador, professor e amigo Dennis Sávio, pela orientação, ensinamento e dedicação em compartilhar um pouco do seu conhecimento. A todos os professores do curso de Sistemas que contribuíram para a minha formação acadêmica.

Aos meus amigos da Universidade Federal do Piauí - UFPI, estes que estavam ao meu lado na maior parte do tempo, sempre nos divertindo, rindo, estudando ou aflitos pelas provas e trabalhos. Não conseguiria chegar até aqui sem vocês.

A todos que contribuíram para este trabalho, um muitíssimo obrigado!

“Um bom *design* é o mínimo de *design* possível”

Dieter Rams

RESUMO

Muitos sistemas interativos confrontam-se com problemas gerados por decisões inconsistentes na produção de suas interfaces. Boa parte disso deve-se ao fato dos desenvolvedores não utilizarem um material que referencie os princípios, as recomendações e as características que uma interface deveria ter para garantir uma qualitativa ao uso. Procurando minimizar tais problemas com o Sistema de Gerenciamento de Projetos (SGP), o objetivo do trabalho foi oferecer uma interface para que o mesmo tenha uma boa interação, utilizando-se de uma proposta de um guia de estilo que auxiliará os projetistas da aplicação a incorporar decisões de *design* que seja efetivamente aceito por parte dos usuários.

Palavra-chaves: Interface, Guia de estilo, *Design,* Interação.

ABSTRACT

Many interactive systems are confronted with problems caused by inconsistent decisions in the production of its interfaces. Much of this is due to the fact that the developers do not use a material that references the principles, recommendations and features an interface should have to ensure a qualitative to use. Seeking to minimize such problems with the Project Management System (SGP), the objective is to provide an interface so that it has a good interaction, using a proposal of a style guide that will assist the implementation of designers incorporate design decisions to be effectively accepted by users.

.

Keywords: *Interface, Style guide, Design, Interaction.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

CSS	*Cascading Style Sheets*
DAS	Desenvolvimento Ágil de *Software*
DDS	Desenvolvimento Distribuído de *Software*
DSDM	*Dynamic Systems Development Method*
FTS	*Follow-the-Sun*
GPS	*Global Positioning System*
GSD	*Global Software Development*
HFRG	*Human Factors Research Group*
IHC	Interface Humano-Computador
MVC	*Modeller-View-Controller*
PNG	*Portable Network Graphics*
SGV	*Scalable Vector Graphics*
SUMI	*Software Usability Measurement Inventory*
W3C	*World Wide Web Consortium*
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

O mercado consumidor está cada vez mais exigente. As pessoas valorizam seu tempo e suas escolhas, e por isso, ao utilizar *sites*, *softwares* ou aplicativos, esperam encontrar tudo que precisam de forma simples, com uma aparência amigável e uma experiência agradável. E é nesse contexto que fica evidente a importância de uma boa interface.

Como a interface é a principal ligação entre o usuário e o sistema em questão, seu planejamento é fundamental para garantir que o sucesso de um projeto não seja comprometido por uma experiência ruim. Uma interface mal projetada tende a problemas que deixam sua aplicação em desvantagem com relação aos demais, além de influir negativamente em como sua ferramenta será usada, prejudicando o desempenho dos usuários nas funções mais básicas.

Construir uma boa interface, que permita ao usuário manuseá-lo intuitivamente, é uma tarefa difícil. Souza (2009) esclarece que isso se deve ao fato de que há uma separação entre a área de engenharia de *software* e a área de Interação Humano-Computador (IHC), onde em muitos casos, a IHC é realizada de forma isolada ou inexistente no desenvolvimento de um projeto. É importante conciliar boas práticas de IHC ao processo de desenvolvimento de *software* para criar sistemas interativos com qualidade, mesclando os aspectos estruturais e funcionais do produto, com a qualidade, interação e experiência de uso em sua utilização.

Neste contexto, desejando que a ferramenta Sistema de Gerenciamento de Projetos (SGP) tenha um *design* acessível para todos os usuários, foi formulada uma interface que oferece uma boa interação humano-computador. O Sistema de Gerenciamento de Projetos (SGP) é uma ferramenta que apoia o controle e a coordenação de projetos de desenvolvimentos co-localizados (tradicional) ou distribuídos (desenvolvimento distribuído de *software*). Gerenciar projetos consiste em planejar, organizar, controlar e verificar se as atividades estão sendo executadas de acordo com o planejado. Em ambientes de desenvolvimento distribuído, novas variáveis, como distância geográfica, fuso horários e diferenças culturais entre equipes, acrescentam-se ao complexo problema de gerenciar projetos de *software*. A ferramenta apresenta módulos que se baseia no conceito de métodos ágeis,

principalmente o *scrum*, que visa o aceleramento e a melhoria contínua do processo de desenvolvimento de uma aplicação.

Para propor uma interface para a ferramenta, foi necessária a elaboração de um guia de estilo, levantando os requisitos mais essenciais para construir uma boa interface seguido de uma avaliação IHC para que permite identificar a satisfação de interação dos usuários perante ao uso. A ideia do guia surgiu com a necessidade dos implementadores da aplicação SGP decidirem que é preciso uma orientação para se dedicarem ao projeto visual, não somente em sua codificação.

1.1 Objetivo

O objetivo deste trabalho é propor um projeto de interface para a ferramenta Sistemas de Gerenciamento de Projetos (SGP), baseando-se nos elementos descritos em um guia de estilo, a fim de oferecer uma interação adequada para os usuários finais.

1.2 Estrutura da monografia

O trabalho está organizado da seguinte maneira:

- Capítulo 2: Fundamentação teórica do trabalho, onde serão apresentados conceitos relacionados a engenharia de software, interface humano-computador e desenvolvimento distribuído de *software*.
- Capítulo 3: Neste capítulo é apresentada a ferramenta utilizada como base para proposta deste trabalho.
- Capítulo 4: Será mostrado o trabalho proposto, com detalhes de cada atividade que serviu para o seu desenvolvimento e os resultados obtidos através do guia e da avaliação da interface.
- Capítulo 5: Considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Como todo trabalho, é necessário ter conhecimento dos principais recursos que o compõe. Neste capítulo, apresenta conceitos e ideais com base em obras de renomados autores, além de abordar as tecnologias utilizadas durante o desenvolvimento do projeto.

2.1 Engenharia de *software*

Em um ambiente globalizado, o *software* possui papel essencial no funcionamento da sociedade, atuando na indústria, na educação, na medicina, no governo e no entretenimento. Nos setores de produções primário, secundário e terciário, o uso de *software* vem promovendo grandes avanços e descobertas tecnológicas. Programas de previsão do tempo e GPS (*Global Positioning System*) são utilizados na lavoura para aumentar a produtividade. *Softwares* que gerenciam toda a cadeia de produção e que controlam máquinas e robôs são cada vez mais comuns na indústria. Sistemas de venda e controle de estoque são diferenciais estratégicos indispensáveis ao comércio.

Diante desse cenário, torna-se obrigatório mesclar boas práticas da engenharia de *software* com processos eficientes de desenvolvimento, para que uma aplicação tenha resultados consistentes, obedecendo estimativas de custos, prazos e qualidade.

A indústria se preocupa com a maneira pela qual é desenvolvido uma aplicação, preocupação que tem levado a adoção da prática de engenharia de *software*, que não só apoia o desenvolvimento profissional, mas também orienta desenvolvedores e engenheiros a utilizarem técnicas, métodos e ferramentas apropriadas à produção de *software* (PRESSMAN, 2006).

A utilização de um modelo adequado de produção aumenta a qualidade do produto a ser desenvolvido e reduz os custos de desenvolvimento. Isso acaba estimulando as organizações a utilizar processos de desenvolvimento que realmente atendam suas necessidades e que sejam eficientes.

A engenharia de *software* é um conjunto de técnicas e métodos aplicados ao desenvolvimento do software, auxiliando a evolução do projeto, oferecendo uma abordagem organizada e sistemática na produção de *software* com qualidade. Segundo Sommerville (2009), a engenharia de *software* é importante no desenvolvimento de sistemas, pois é necessário ter em mente que um planejamento envolvendo aspectos, especificações, custos, restrições, correções e validações são os passos essenciais para entregar um produto confiável.

O processo de *software* é como uma estrutura central para as tarefas que são necessárias para construir sistemas de alta qualidade. A produção de uma aplicação pode ser definida como um conjunto de atividades denominado processo de *software* (PRESSMAN, 2006), que constitui o principal alicerce da engenharia de *software*, que forma a base para o desenvolvimento e estabelece os métodos técnicos aplicados ao projeto, como modelos, documentações, dados, relatórios, etc.

Existem vários processos de desenvolvimento de *software*, mas todos devem incluir quatro atividades fundamentais (SOMMERVILLE, 2009):

Especificação: define as funcionalidades e as restrições de funcionamento do *software*.

Projeto e implementação: o *software* a ser produzido deve atender as especificações.

Validação: O *software* deve ser validado para garantir as exigências do cliente.

Evolução: O *software* deve evoluir para suprir aos novos requisitos que o cliente exigir, suportando assim a mudanças.

Não existe um processo ideal para produzir um *software*, o que existe é a melhoria de um processo, onde podem ser aprimoradas técnicas consideradas ultrapassadas, e removidas técnicas consideradas inadequadas (SOMMERVILLE, 2009). Diante das mudanças nos processos, acabam surgindo diferentes abordagens, que podem tornar-se padrões, resultando em melhorias na comunicação entre os desenvolvedores, redução no tempo de treinamento e nos custos dos processos automatizados.

2.1.1 Modelos de processos de *software*

Sommerville (2009) cita que um modelo de *software* é uma representação simplificada de um processo de *software*. Pode ser visto como uma forma de abstração das atividades envolvidas no processo, fornecendo informações essenciais para gerenciar o desenvolvimento (e o progresso) de *software*. Os modelos de processos proporcionam aos envolvidos no projeto uma melhor compreensão do sistema a ser construído, principalmente os programadores. A documentação obtida através do modelo serve de base para as atividades de construção e manutenção. Os modelos mais utilizados são: modelo cascata, modelo incremental e o modelo orientado a reuso (SOMMERVILLE, 2009). A escolha do modelo a ser utilizado no projeto de desenvolvimento tem uma profunda influência nos problemas e riscos que surgirão no processo e também nas possíveis soluções para contorná-los.

O **modelo cascata**, conhecido como ciclo de vida clássico ou modelo clássico, surgiu na década de 70 e é o modelo mais popular e estudado na engenharia de *software*. Tem esse nome devido ao processo de transição entre as etapas, que é sequencial e flui de cima para baixo como uma cascata. As etapas deste modelo são:

- **Definição de requisitos:** Os objetivos, as restrições e as funcionalidades do sistema são estabelecidos por meio de consulta aos potenciais usuários do sistema. Após isso, são definidos em detalhes e funcionam como uma especificação do sistema.
- **Projeto de sistemas e *software*:** Agrupa os requisitos por meio da definição do uma arquitetura geral do sistema. Nesta etapa, são identificados e descritos as abstrações e os relacionamentos do sistema.
- **Implementação e teste unitário:** O projeto de *software* é implementado na forma de um conjunto de módulos de programas, cujas especificações devem ser verificadas por meio de testes unitários, que envolvem verificar se cada módulo atenda sua especificação.
- **Integração e teste de sistemas:** Todos os módulos são integrados e testados como um sistema completo, com o intuito de garantir que os requisitos de *software* foram atendidos. Após os testes, o sistema é entregue ao cliente conforme solicitado.

- **Operação e manutenção:** Logo após a entrega, o sistema é instalado e colocado em operação. A manutenção envolve corrigir erros e *bugs* não identificados na etapa de desenvolvimento ou o acréscimo de novas funcionalidades conforme requisitos são descobertos. É considerada a etapa mais longa do ciclo de vida do *software*.

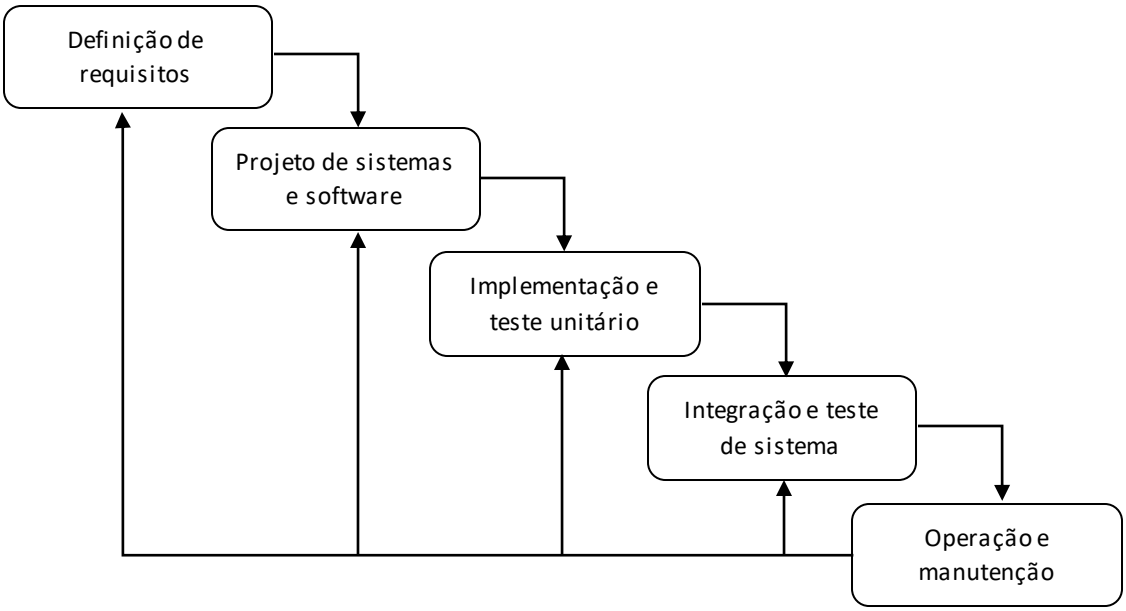

Figura 1 – Modelo Cascata

Fonte: Sommerville (2009).

Conforme demostrado na Figura 1, a adoção desse modelo torna o processo de desenvolvimento estruturado, pois permite seguir uma ordem sequencial de etapas. São produzidos documentos para cada etapa do ciclo do modelo e o resultado de cada etapa é obtido por meio da aprovação dos documentos, sendo possível que os gerentes monitorem o progresso de acordo com o plano de desenvolvimento. Somente quando uma etapa é aprovada e concluída, a etapa seguinte é iniciada, de modo que uma versão funcional do *software* somente estará disponível quando o processo de desenvolvimento alcançar a última etapa, ocasionando na demora para entrega do *software* ao cliente. Em princípio, esse modelo deve ser usado quando os requisitos são bem compreendidos ou quando suas alterações não causam grandes mudanças radicais ao projeto.

O **modelo incremental** é composto de etapas lineares e paralelas, e se baseia na ideia de desenvolver uma versão inicial, testar antecipadamente, avaliar e continuar por meio de várias versões até chegar em sua versão final. As atividades

que compõem o modelo incremental são lineares, mas funcionam de forma paralela, com rápidos *feedbacks* entre essas etapas.

Após documentar os detalhes do sistema, devem ser implementadas as funcionalidades essenciais do sistema para que haja uma versão inicial, assim demonstrado na Figura 2. A cada incremento com uma nova funcionalidade, o sistema é liberado para que o cliente já o possa testar e averiguar se está conforme o requisitado. Em caso negativo, somente o incremento atual é alterado, sendo possível realizar mudanças com facilidade e com menores custos, visto que a quantidade de análise e documentação a ser refeita é muito menor do que a necessária no modelo em cascata (SOMMERVILLE, 2009).

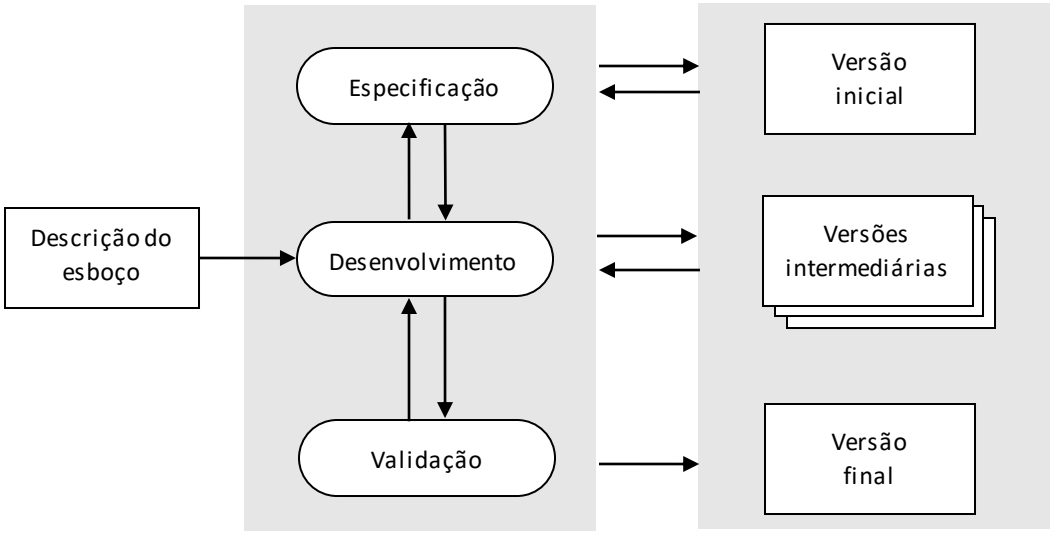

Figura 2 – Modelo incremental

Fonte: Sommerville (2009).

O desenvolvimento incremental é voltado particularmente para sistemas de grandes e complexos, sendo uma mescla das abordagens dirigida a planos e ágeis. O modelo incremental proporciona um melhor gerenciamento de riscos, pois é possível confirmar o resultado de cada versão juntamente com o cliente, verificando se o progresso do sistema em desenvolvimento está saindo conforme planejado.

O **modelo orientado a reuso** descreve a utilização de procedimentos, funções e classes previamente desenvolvidos, denominados componentes ou bibliotecas, a serem incorporados nas aplicações. A ideia central da reutilização é evitar a reconstrução de partes dos sistemas. Esse modelo de processo se concentra na

integração de componentes reusáveis, onde Sommerville (2009) advoga que abordagens orientadas a reuso dependem de uma ampla base de componentes reusáveis de *software* e de um *framework* de integração para a composição desses componentes. Esses componentes são capazes de oferecer funcionalidades específicas que atendem adequadamente as funções das quais foram encarregados, mas para isso, é preciso que haja um planejamento geral dos possíveis componentes que possam ser reutilizados no projeto de desenvolvimento, além de que cada componente tenha uma documentação associada, possibilitando aos desenvolvedores compreendê-los e adaptá-los. As etapas do modelo orientado a reuso, citada por Sommerville (2009), são (Figura 3):

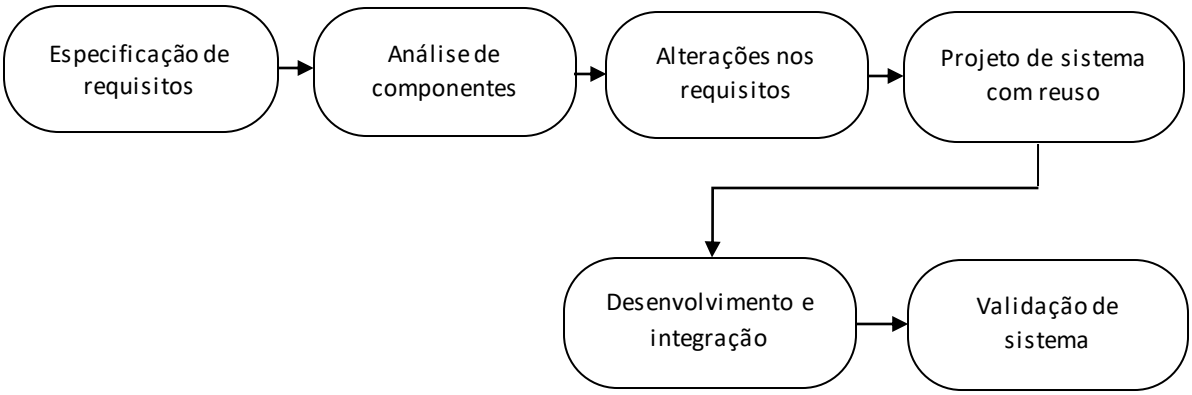

Figura 3 – Modelo orientado a reuso

Fonte: Sommerville (2009).

- **Especificação de requisitos:** Etapa que aborda as especificações do projeto de desenvolvimento.
- **Análise de requisitos:** Após a especificação de requisitos, é realizado uma busca de todos os componentes candidatos a fornecer ou implementar funcionalidades que correspondem a tal especificação.
- **Alterações nos requisitos:** Os requisitos são avaliados baseando-se nas documentações dos componentes. Em seguida, são feitas modificações nos requisitos correspondendo aos componentes. Caso algum requisito não seja possível ser modificado, a etapa anterior de análise é refeita para buscar outras soluções possíveis.

- **Projeto de sistema com reuso:** Etapa onde é projetado o *framework* específico ao desenvolvimento, contendo todos os componentes listados na etapa anterior, ou em outro caso, reutilizado um *framework* que seja apto ao projeto.
- **Desenvolvimento e integração:** Os componentes são integrados ao sistema, e as partes que não é possível integrar e reusar, é desenvolvido.
- **Validação de sistema:** Etapa que avalia e testa o sistema final com todos os componentes integrado.

O modelo orientado a reuso possui diversos benefícios. Oferece um desenvolvimento acelerado, proporciona uma entrega rápida do produto ao cliente e acrescenta confiabilidade ao programa, devido ao uso de componentes previamente testados em diferentes ambientes. Porém, o controle sobre a evolução do sistema torna-se mais complexo, visto que versões novas dos componentes reusáveis podem não estar sob o controle da organização que está utilizando-os (SOMMERVILLE, 2009).

2.1.2 Metodologias ágeis

Até o início dos anos 90, acreditava-se que a melhor maneira de obter um *software* de qualidade era a realização de um cuidadoso planejamento, seguida da utilização de métodos formalizados para um processo de desenvolvimento extremamente rigoroso e controlado. Essa visão vinha diretamente da comunidade de desenvolvedores, que participava de grandes projetos a longo prazo, como sistemas aeroespaciais que se integravam na forma de grandes equipes de diferentes empresas dispersas geograficamente. Porém, essa abordagem pesada causava uma grande insatisfação, pois os processos de *softwares* costumavam ser demorados, dispendendo mais tempo em análises de como o sistema deve ser desenvolvido do que no desenvolvimento de programas e testes, atrasando a entrega final do produto ao cliente (SOMMERVILLE, 2009).

Essa insatisfação levou os desenvolvedores a proporem métodos ágeis, que buscavam acelerar o processo de desenvolvimento, destacando a entrega rápida de protótipos em funcionamento ao cliente e reduzindo o tempo na documentação e em atividades com menor relação com o projeto. De forma simplificada, metodologias

ágeis são (SOMMERVILLE, 2009, p.39) “métodos de desenvolvimento incremental em que os incrementos são pequenos e, normalmente, as novas versões do sistema são criadas e disponibilizadas aos clientes a cada duas horas”. As diretrizes do desenvolvimento ágil enfatizam que os programas não são construídos como um todo, e sim, como uma série de incrementos evolutivos intercalados com *feedbacks* entre a equipe e o cliente, sendo que o cliente tem uma participação maior com a equipe de desenvolvedores.

Embora a ideia de métodos ágeis tenha sido definida durante a década de 90, somente em 2001 ela se concretizou em forma de um manifesto, que sugeria alternativas revolucionárias aos modos de desenvolvimentos tradicionais. “Em essência, os métodos ágeis foram desenvolvidos em um esforço para vencer as fraquezas percebidas e reais da engenharia de *software* convencional” (PRESSMAN, 2006, p.58). Em um comparativo, no modelo Cascata, foca-se muito em documentar o levantamento de requisitos, na árdua tarefa de abstrair todos os possíveis problemas que possam vir a surgir durante a implementação. Isso resulta em possíveis atrasos e a uma demora considerável até o início do processo de desenvolvimento. Além disso, é pouco frequente a participação do cliente, sendo normalmente limitada ao início do projeto e à entrega final.

Em métodos ágeis, o tempo empregado em documentação é menor, pois são levantados somente os requisitos mais relevantes para o primeiro protótipo. Em alguns casos, a documentação é totalmente descartada ou acaba ficando em segundo plano, já que ela é descrita no momento inicial do projeto, e conforme o progresso, vai ficando desatualizada. Os entusiastas de métodos ágeis argumentam que é um desperdício de tempo criar essa documentação, pois a chave para a implementação de *software* manutenível é a produção de códigos de alta qualidades e legíveis (SOMMERVILLE, 2009). A participação do cliente é de extrema importância, pois ele é consultado a cada fim de ciclo para testar os protótipos a fornecer *feedback*.

O manifesto ágil foi idealizado pelos membros da *Agile Alliance*, um grupo de profissionais reunidos no intuito de melhorar suas práticas no desenvolvimento e compreender melhor os aspectos na área de *software*. Dois integrantes da *Agile Alliance*, Fowler e Highsmith (2001), publicaram um artigo intitulado *The Agile Manifesto* (2001), citando 12 princípios que descrevem valores e aspectos importantes relacionados ao sucesso de um processo de desenvolvimento. Tais princípios foram

definidos com a troca de experiências, práticas e teóricas sob consenso de todos os envolvidos na reunião, fazendo surgirem novos paradigmas de desenvolvimento de *software*. Os princípios são:

1 **Nossa maior prioridade é satisfazer o cliente através de entregas rápidas e contínuas de *software* funcional.** Este primeiro princípio foca na satisfação do cliente, onde a estratégia é entregar partes prontas do produto que possam ser avaliadas no decorrer do processo de desenvolvimento.

2 **Aceitar as mudanças de requisitos do projeto, mesmo no fim do desenvolvimento. Os processos ágeis se adequem a mudança, para que o cliente possa tirar vantagens competitivas.** De modo simples, o segundo princípio descreve a necessidade de o projeto de desenvolvimento estar apto a realizar alterações nos requisitos sempre que o cliente exigir.

3 **Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferências aos períodos mais curtos.** Enfatiza que a entrega do *software* funcional ao cliente deve ser frequente, e para isso, a equipe precisa calcular a duração de tempo ideal para o ciclo de desenvolvimento, que permita entregar o produto para avaliação.

4 **Pessoas relacionadas à negócio e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.** Princípio que descreve que é necessária a interação entre as pessoas que precisam do produto e as pessoas que o desenvolvem.

5 **Construir projetos ao redor de indivíduos motivadas. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.** Devem ser identificados pontos que motivam os membros e os estimulam para desempenharem suas tarefas com o melhor resultado possível.

6 **O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa face a face.** Princípio que enfatiza que a melhor prática de troca de informações é a comunicação direta. O contato interpessoal permite perceber fatores como: insegurança, autoestima ou insatisfação de sua equipe que não podem ser descritos em documentações, mas podem influenciar na tomada de decisões.

7 ***Software* funcional é a medida primária de progresso.** Princípio que prioriza a entrega, a cada iteração, do produto funcionando.

8 **Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.** É necessário promover um desenvolvimento em longo prazo, desde que todos os participantes estejam cientes de manter um ritmo constante.

9 **Contínua atenção à excelência técnica e bom *design*, aumentam a agilidade.** Uma monitoração da implantação adequada das tecnologias de desenvolvimento, somada a um *design* claro e intuitivo tornam o processo muito mais ágil, com maiores chances de entregar a aplicação rapidamente.

10 **Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.** Esse princípio se diz respeito de como é importante o uso de abordagens simples por, teoricamente, serem mais fáceis de serem alteradas e adaptadas. O importante é manter foco maior no que é minimalista, porém funcional, do que no que é exuberante, mas extremamente complexo.

11 **As melhores arquiteturas, requisitos e designs emergem de times auto organizáveis.** Uma equipe auto organizável é similar a um startup: um grupo multidisciplinar, competente, decidido e que busca ampliar seus conhecimentos e ideias, fazendo emergir novas descobertas.

12 **Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.** É importante a equipe fazer, em intervalos regulares, uma reflexão a respeito do progresso no trabalho. Essa prática permitirá que melhorias e ajustes sejam feitos para o aprimoramento do projeto.

2.1.2.1 *Scrum*

Existem muitos modelos ágeis com apresentam características que os tornam singulares e que satisfazem em maior grau ou menor grau ao manifesto para o desenvolvimento ágil (PRESSMAN, 2006), como XP (*Extreme Programming*), DAS (Desenvolvimento Adaptativo de *Software*) e DSDM (*Dynamic Systems Development*

Method). *Scrum* é considerado um dos mais utilizados por proporcionar benefícios na forma de conduzir projetos. Segundo Rafael Sabbagh (2013), os principais benefícios oferecidos pelo *Scrum* são:

- Entregas frequentes de retorno ao investimento dos clientes;
- Redução dos riscos do projeto;
- Maior qualidade no produto gerado;
- Mudanças utilizadas como vantagem competitiva;
- Visibilidade do progresso do projeto;
- Redução do desperdício;
- Aumento de produtividade .

O *Scrum* foi desenvolvido por Jeff Sutherland e Ken Schwaber (1993) fortemente baseado nos princípios do manifesto ágil. Seu nome vem da jogada/formação do *Rugby*, onde os jogadores se encaixam e forma uma parede humana e caso a formação falhe, a jogada é comprometida. Isso é bastante evidente no modelo, cujo ponto principal é o trabalho em equipe, e os papéis empenhados necessitam de um alto grau de cooperação para alcançar um objetivo comum.

Autores como Rafael Sabbagh (2013) não consideram o *Scrum* como uma metodologia ou técnica, e sim, um *framework* simples, já que em sua estrutura básica é constituída de valores, princípios e eventos associados a uma organização que possa organizar e gerenciar o desenvolvimento de seus projetos, e diferentemente de uma metodologia, que possui padrões rigorosos e séries de etapas sequenciais, baseados em prazos e orçamentos. Utilizando uma analogia para entender mais este conceito, *Scrum* está mais para um esqueleto, que sem a agregação de órgãos, músculos e tecidos não é capaz de movimento; enquanto um método está para um corpo humano completo, que possui tudo que é necessário para se movimentar.

"*Scrum* utiliza-se de poucos conceitos novos, e essa é uma de suas grandes qualidades: juntar práticas de mercado já conhecidas e consagradas de uma forma organizada e que funciona" (SABBAGH, 2013, p. 4). Por ser um *framework,* pode ser combinado com diferentes métodos, como o *Extreme Programming* (XP), o que possibilita a utilização de técnicas (programação em par, integração contínua, desenvolvimento dirigido por testes, etc.) que podem auxiliar nos diversos cenários de desenvolvimento de produtos, desde os mais simples até os mais complexos.

Uma das características mais interessantes está associada à equipe, denominada de equipe *Scrum*, que é formada por poucos membros - cerca de sete a nove pessoas - todas coletivamente com habilidades para produzir com qualidade. Nada impede que sejam formadas equipes com muitos mais membros, mas equipes pequenas são mais disciplináveis e se auto organizáveis. O time *Scrum* é formado pelo *Product Owner*, *Scrum Master* e equipe de desenvolvimento, onde cada indivíduo entrega o resultado dos seus trabalhos para os demais de forma independente e autônoma. A Figura 4 demonstra a formação de um time *scrum:*

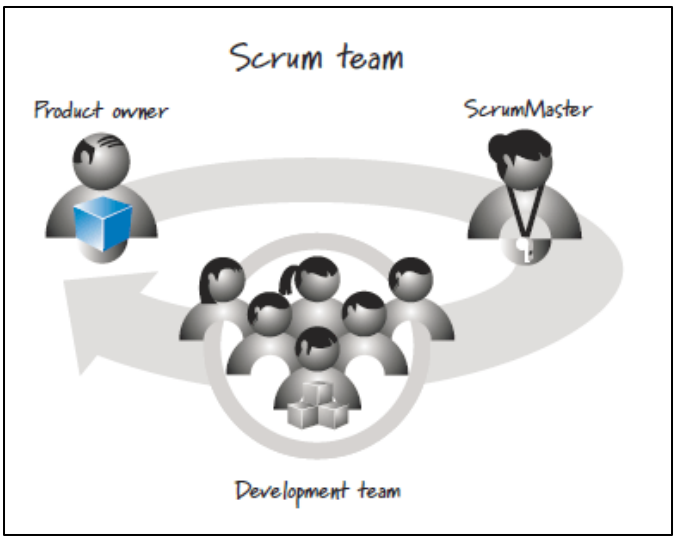

Figura 4 – Formação de um time *scrum*

Fonte: Google (2016).

O *Product Owner* é o membro com poderes de liderança, delegando os recursos, as funcionalidades e a ordem que serão construídos. Não é considerado um gerente de projetos, mas é de sua inteira responsabilidade manter e comunicar a todos os participantes uma visão clara que a equipe deseja alcançar no projeto. Ele é o que prioriza os itens do *Product Backlog*.

O *Scrum Master* obrigatoriamente deve conhecer muito bem o *Scrum*, sendo seu papel ajudar a equipe a entender e abraçar as práticas. Ele deve liderar o processo, auxiliando a equipe na utilização do *Scrum*. Tem o papel de facilitador, não sendo chefe de ninguém.

A equipe de desenvolvimento são as pessoas que irão construir o projeto, sendo os responsáveis por como fazer e quais ferramentas a serem utilizados. De fato, a ideia é que a equipe se auto organiza para determinar a melhor maneira de alcançar a meta estabelecida pelo *Product Owner*.

A dinâmica do *Scrum* começa com a visão do produto, sendo o *Product Owner* o responsável por prover essa visão. A visão é uma abstração do projeto, representado por um *business case*, modelagem, *blueprint*, ou outro tipo de macro-planejamento, desde que tenha descrito o que “ele quer” e “onde ele quer chegar”. Essa visão é desmembrada em uma lista de funcionalidades que compõem o projeto. Com o *Scrum Master* auxiliando o *Product Owner*, as funcionalidades são classificadas em prioridades, atribuindo valores, como por exemplo, "funcionalidades imprescindíveis", "funcionalidades importantes” e “funcionalidades extras”. Essa lista priorizada é chamada de *Product Backlog*.

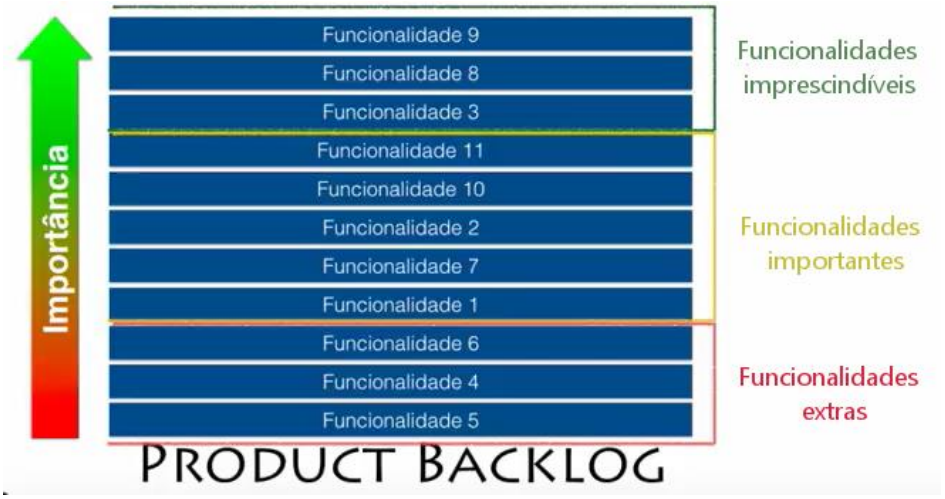

Figura 5 - *Product Backlog* priorizado

Fonte: Google, adaptado pelo autor (2006).

Definido o *Product Backlog*, é realizada uma reunião, denominada *Sprint Planning Meeting*, envolvendo o *Product Owner*, o *Scrum Master* e o time de desenvolvimento com o intuito de planejar os *sprints*, que são períodos de tempos onde itens selecionados do *Product Backlog* serão construídos e entregues. Há uma regra básica em que cada *sprint* obrigatoriamente possui duração fixa, normalmente de 2 a 4 semanas. Além de planejar o *sprint*, são levantados pontos de estimativas baseando-se no esforço que a equipe empenha em construir uma funcionalidade e quantas funcionalidades podem ser completamente construídas dentro do tempo de duração do *sprint*.

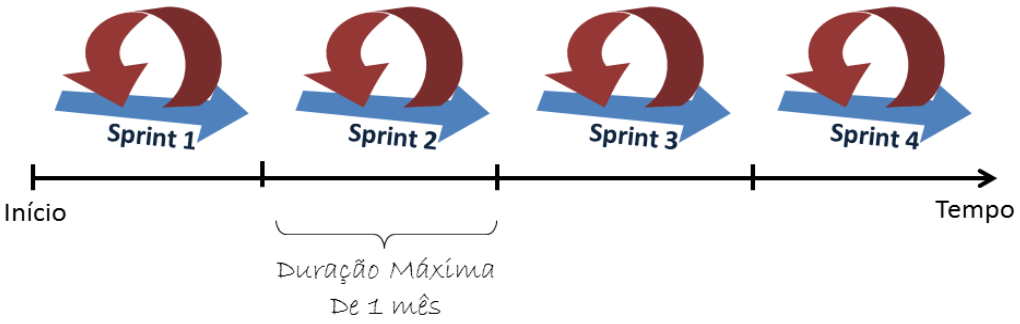

Figura 6 – Sequência de *sprints* para a construção do projeto

Fonte: www.mindmaster.com.br (2016).

Após a *Sprint Planning Meeting*, começa o ciclo do primeiro *sprint*, sendo selecionadas as funcionalidades priorizadas no topo do *Product Backlog*. Após o término do *sprint*, é esperado que o incremento do produto seja entregue, testado e aprovado. O *Product Owner* poderá verificar se há necessidades de mudanças. Essas mudanças são inseridas imediatamente no *Product Backlog*, com sua devida prioridade. Todo esse processo com *sprints*, incrementos e mudanças será repetido até que todo o *Product Backlog* seja construído e o produto final esteja pronto, contemplando todas as mudanças solicitadas.

Ao final de cada *sprint* ocorrem duas atividades fundamentais: A *Sprint Review Meeting*, uma reunião onde a equipe de desenvolvimento demonstra o que foi alcançado e se foi construído de acordo com o esperando. Resumidamente, é a apresentação do que foi feito no *sprint*, e é nesse momento em que o *backlog* é atualizado com as mudanças. A segunda atividade é a *Sprint Retrospective*, reunião que tem como objetivo a revisão e avaliação do que foi feito, e a reflexão no que pode ser melhorado ou removido do processo.

Em todos os dias é realizada a *Daily Scrum*, uma reunião de 15 minutos onde são feitas 3 perguntas a cada membro da equipe:

- **O que fez ontem?** Se individuo ajudou a equipe a atingir a meta;
- **O que vai fazer hoje?** Se individuo ajudará a equipe a atingir a meta;
- **Tem algum impedimento?** Se há algum obstáculo que impeça de ajudar a equipe.

Ao responder essas perguntas, todos conseguem visualizar de uma maneira geral como está progredindo o trabalho do *sprint*.

Figura 7 – Atividades que procedem o *scrum*

Fonte: mindmaster.com.br, adaptada pelo autor (2016).

Defensores de métodos tradicionais argumentam que o *scrum* é uma abordagem criada por programadores problemáticos com autoridade, já que não há um gerente de projetos, e que não há como gerenciar e estimar sem um cronograma rigoroso. Já adeptos do *scrum* fundamentam que nenhuma metodologia existente resolve totalmente os problemas de engenharia ou de qualidade de *software*, mas oferece mecanismos para que a equipe vá atrás de soluções para contorná-los.

2.2 Interação Humano-Computador (IHC)

Um dos principais conceitos objetivos da tecnologia é aprimorar e facilitar a realização de atividades, sejam elas diárias ou eventuais. Com as pessoas interagindo cada vez mais com os computadores e outros aparatos tecnológicos, torna-se necessário que essa interação seja fácil e amigável. A Interação Humano-Computador (IHC) é "uma disciplina interessada no projeto, implementação e avaliação de sistemas computacionais interativos para o uso humano" (BARBOSA; DA SILVA, 2011, p. 10). Em outras palavras, ela busca entender o processo de uso e comunicação na relação humano e computador.

A IHC investiga e avalia a implementação de sistemas interativos para o uso humano, considerando não somente os aspectos computacionais, mas também características do comportamento humano como a psicologia da leitura, percepção visual, e o modelo mental do usuário. Com isso, ela é considerada um campo de

estudo multidisciplinar por envolver outras áreas além da computação: a psicologia, a sociologia e antropologia. O conhecimento adquirido dessas áreas contribuíra com a compreensão de técnicas para o *design*, a ergonomia, a linguística e semiótica (BARBOSA; DA SILVA, 2011).

2.2.1 Interação, interface e *affordance*

A Interação humano-computador é um processo que engloba manipulação, comunicação, conversa, troca e influência. O autor Norman (1986 apud BARBOSA; DA SILVA, 2011) cita que a interação se dá quando o usuário formula uma intenção, planeja suas ações, atua sobre a interface, percebe e interpreta a resposta do sistema e avalia se seu objetivo foi alcançado. Kammersgaard (1988 apud BARBOSA; DA SILVA, 2011) aponta que há quatro perspectivas de interação, conforme demostrado na Figura 8:

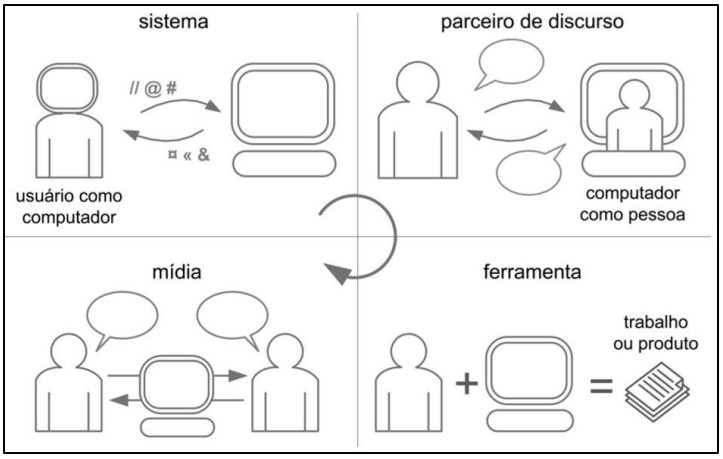

Figura 8 – Perspectivas de interação humano-computador

Fonte: Babosa; Da silva (2011).

Perspectiva de sistema: O usuário é interpretado pela máquina como se fosse outro sistema operacional, dando a impressão de ser uma mera transmissão de dados entre pessoas e sistemas. Nesta perspectiva, é comum o uso de linguagem de comandos (terminal ou *prompt* de comando), linguagens de programação ou *scripts.*

Parceiro de discurso: O sistema atua como um ser humano, sendo capaz de tomar decisões, raciocinar, adquirir informações. Esta perspectiva surgiu a partir dos

estudos na área de Inteligência Artificial e visa tornar a interação humano-computador mais próxima de uma interação entre seres humanos.

Perspectiva de Mídia: O sistema é visto como um meio intermediador na comunicação entre pessoas. O usuário se comunica com outros usuários por meio do sistema, e não com o sistema. A perspectiva de mídia é composta de sistemas de e-mail, fóruns, *chats* e redes sociais.

Ferramenta: o usuário utiliza o sistema como um instrumento que auxilia na criação, realização de atividades e obtenção de resultados. O processo de interação dessa perspectiva é encadeado pelo conjunto de ações e reações que a ferramenta, ou sistema interativo, oferece.

Se a interação é o processo que ocorre durante o uso, a interface é o meio de contato entre o usuário e o sistema. “A interface de um sistema interativo compreende toda a porção do sistema com a qual o usuário mantém contato físico (motor ou perceptivo) ou conceitual durante a interação” (MORAN, 1981 apud BARBOSA; DA SILVA, 2011, p. 25).

A interface envolve tanto *software* quanto *hardware*. Dispositivos de entrada saída, como teclados, *mouses*, *joysticks* e microfones permitem ao usuário interagir ativamente sobre a interface, enquanto os dispositivos de saída, como monitores, impressoras, caixas de som permitem que o mesmo perceba as reações do sistema, agindo passivamente na interação. Essa relação é descrita como **contato físico** por possibilitar a manipulação dos elementos de interface. O **contato conceitual** consiste na percepção e raciocínio do usuário desencadeado pelo contato físico, com base nos dispositivos de entrada e saída.

O contexto de uso influencia a forma como o usuário interpreta a interface. As características da interface que evidenciarão a maneira de como utilizá-lo e o que é possível fazer com ele são definidas como *affordances*. O autor Norman ainda complementa que *affordance* é “o conjunto das características de um objeto capazes de revelar aos seus usuários as operações e manipulações que eles podem fazer com ele” (NORMAN, 1988 apud BARBOSA; DA SILVA, 2011, p. 27). Em um exemplo, a *affordance* de um objeto, como um botão de uma interface gráfica, dá a possibilidade de pressioná-lo e consequentemente, acionar alguma operação associado a ele. No desenvolvimento de uma interface, o *designer* precisa tomar cuidado para não criar

falsas *affordances* que dêem a entender que a interface funciona de uma maneira, quando na verdade funciona de outra maneira.

2.2.2 Processos de *design* de IHC

Para conhecer o que é *design*, é necessário primeiro entender o conceito de artefatos. "Artefatos são produtos artificiais, fruto da inteligência e do trabalho humano, construídos com um determinado propósito em mente" (BARBOSA; DA SILVA, 2011, p.92). Os artefatos são objetos cujas forma, estrutura, qualidade e função foram determinados e construídos por pessoas. O uso dos artefatos interfere direta e indiretamente no cotidiano, tanto de maneira negativa quanto positiva. Como exemplo, a aquisição de uma bicicleta, que proporcionará uma intervenção positiva na situação atual: a prática de exercícios, do lazer, da locomoção e a economia em meios de transportes, mas consequentemente, precisará de investimento em equipamentos de segurança e de um espaço reservado para armazená-la, os quais são aspectos considerados negativos.

O *design* consiste no estudo de uma situação para que essa possa ser melhorada, através da criação ou do melhoramento de artefatos, e assim mudar o estado corrente das coisas. Podemos caracterizar a atividade de *design* como sendo composta de três etapas fundamentais (LAWSON, 2006; LOWGREN & STOLTERMAN, 2004 apud BARBOSA; DA SILVA, 2011):

Análise da situação atual: estudar e interpretar a situação atual. Aponta a necessidade de estudar pessoas ou artefatos relacionados a uma situação, e encontrar problemas ou algo que possa ser melhorado.

Síntese de uma intervenção: planejar e executar uma intervenção na situação atual, onde tenta-se prever as possíveis consequências que possa ser impactado.

Avaliação da nova situação: verificar o efeito da intervenção, comparando a situação analisada anteriormente com a nova situação, atingida após a intervenção.

O *design* estuda as situações e as formas de melhorá-la, identificando antecipadamente os "problemas", planejando e aplicando uma intervenção para que

no fim, avalia-se a nova situação, compara-se as duas situações para concluir se a intervenção foi ou não bem-sucedida.

A execução das atividades de análise da situação atual, síntese da intervenção e avaliação da nova situação dá-se por meio de processos de *design*. Cada processo funciona de maneira distinta e não necessariamente linear, possibilitando que o designer após ter concluído uma etapa, retroceda à etapa anterior para alteração e avance novamente no projeto, caracterizando a iteração. A navegação entre as etapas sem uma linearidade faz com que o *designer* aprenda ainda mais ao longo do processo, com chance de voltar a uma etapa anterior caso aconteçam descobertas no decorrer do desenvolvimento. Os principais processos de *design* são:

Ciclo de vida em estrela: Processo definido por Hix e Hartson (1993), sendo o primeiro processo a ser difundido. É composto de seis atividades: análise de tarefas, de usuário e funções; especificação de requisitos; projeto conceitual; especificação do *design*; prototipação; implementação e avaliação. Todas as atividades citadas são ligadas à atividade de avaliação**,** que constitui o ponto central do processo, em que são avaliados os resultados de cada uma das demais atividades.

Engenharia de usabilidade de Nielsen: Proposto por Jakob Nielsen (1993), ressaltando um conjunto de atividades que ocorrerá durante todo o ciclo de vida do produto proposto. Ao todo, são 10 (dez) atividades que são:

- Conheça seu usuário;
- Realize a análise competitiva;
- Defina metas de usabilidade;
- Faça *designs* paralelos;
- Adote o *design* participativo;
- Faça o *design* coordenado da interface como um todo;
- Aplique diretrizes e análise heurística;
- Faça protótipos;
- Realize teste empíricos;
- Pratique design iterativo.

Engenharia de usabilidade de Mayhew: Processo criado por Deborah Mayhew (1999), que reúne e organiza diferentes atividades propostas na área de IHC, orientando o trabalho do *designer* a uma boa solução interativa. Apresenta três fases: A **análise de requisitos**, que define as metas de usabilidade com base no perfil dos

usuários, a análise de tarefas, possibilidades e limitações da plataforma em que o sistema será executado e os princípios gerais de *design* de IHC; o **design/avaliação/desenvolvimento**, que conceber uma solução de IHC que atenda às metas de usabilidade estabelecidas na fase anterior; e a **instalação**, onde coleta opiniões dos usuários depois de algum tempo de uso. É um processo que possui um esquema simples de ser seguido por possuir possui um gráfico informativo, funcionando no estilo perguntas e respostas.

***Design* contextual:** O *designer* procura entender as necessidades dos usuários por meio de uma investigação minuciosa do contexto de uso. A investigação inclui: investigação contextual; modelagem do trabalho; consolidação; re-projetar; projetar uma solução de interação e interface; construir protótipos; avaliar os protótipos.

***Design* baseado em cenários:** Um cenário é simplesmente uma história sobre pessoas executando uma atividade. Esse processo de *design* considera que para conceber uma solução de IHC, é preciso utilizar diferentes tipos de cenários para representá-lo (BARBOSA; DA SILVA, 2011). A equipe de *designers*, trabalha juntamente com a equipe de pessoas interessadas no sistema. As atividades propostas pelo *design* baseado em cenários são: cenários de problemas; cenário de atividade; cenário de informação e cenário de interação. Ao escrever, ler e revisar cenários, a equipe de *design* tem a oportunidades de discutir e analisar como as atividades dos usuários são afetadas pelas tecnologias existentes e como elas poderiam ser afetadas pelo sistema sendo desenvolvido.

***Design* dirigido por objetos:** Orienta o *designer* a projetar uma solução de IHC criativa que apoie os usuários em atingirem seus objetivos (COOPER, 2007 apud BARBOSA; DA SILVA, 2011). O *design* dirigido por objetivos é sistemático e se propõe a investigar e atender às necessidades e aos objetivos dos usuários, bem como atender aos requisitos técnicos, do negócio e da organização. É o método que mais incentiva a criatividade do *design*, possibilitando analisar também as tecnologias disponíveis e usá-las a seu favor. Também é dividido em etapas: pesquisar, modelar, definição de requisitos, projeto conceitual, refinamento, manter. Cada fase é iterativa e a avaliação é feita ao final de cada atividade.

***Design* centrado na comunicação:** Tem como base teórica a engenharia semiótica, ou seja, transmitir a metacomunicação do *designer* de forma eficiente e eficaz, afim de produzir um sistema interativo. Propõe três atividades: análise do

usuário, domínio e contexto de uso; projeto de interação e interface; e avaliação do que foi projetado. Nesse processo, a solução de IHC é projetada envolvendo os usuários e para eles, mas não por eles.

2.2.3 Avaliação de IHC

Qualquer processo de desenvolvimento que busque produzir um sistema interativo precisa ser avaliado para garantir a experiência particular do usuário quanto ao seu uso. A avaliação de IHC é uma atividade que orienta a identificar e julgar a qualidade da interface e de interação que um sistema tem a oferecer (BARBOSA; DA SILVA, 2011).

Durante o processo de desenvolvimento de sistemas interativos, costumam ocorrer problemas despercebidos que acaba prejudicando a qualidade final do produto. Para entregar ao usuário final um produto de qualidade é preciso, além de continuar seguindo os processos de desenvolvimento e de *design*, avaliar se o produto resultante atende aos critérios de qualidades desejados.

A questão fundamental de uma avaliação é definir objetivos que determinam os aspectos relacionados ao uso, motivados por requisições, reclamações ou comportamento de qualquer um dos interessados no sistema, sejam usuários, clientes, *designers* ou desenvolvedores. Por exemplo, o cliente pode querer verificar se a qualidade de uso é o diferencial do seu produto, o *designer* propor outras alternativas de *layout*, e os usuários demostrarem desinteresse ou reclamar do sistema. É possível definir os objetivos de uma avaliação, basta o avaliador estar atento a situações como essas e os interesses dos envolvidos. Os aspectos relacionados ao uso são avaliados em: apropriação de tecnologia pelos usuários, ideias e alternativas de *design*, conformidade com um padrão e problemas na interação e na interface.

A **apropriação de tecnologia pelos usuários** permite investigar como os usuários realizam suas atividades antes e depois da intervenção com o sistema, permitindo uma melhor compreensão sobre o contexto em que o sistema avaliado afeta no cotidiano dos utilizadores. Pode ser avaliando se são capazes de atingir metas, estabelecidos no sistema com um tempo razoável e com poucos erros, se há necessidade de treinamento, em que grau as tecnologias disponíveis satisfazem suas

necessidades e preferências, se há motivação para explorar e aprender as funcionalidades.

A **avaliação de ideias e alternativas de *design*** compara diferentes formas de soluções de interface. São utilizados vários protótipos com vários níveis de detalhes, onde o avaliador observa quais as preferências dos avaliados, como ele costumam satisfazê-los e por que o fazem assim.

A **conformidade com um padrão** avalia se uma solução de IHC segue algum padrão já estabelecido. Em exemplo, se a solução de interação de um usuário com certa limitação física com o sistema estar de acordo com os padrões de acessibilidade do W3C; ou realizar operações básicas como abrir, redimensionar e fechar janelas estar de acordo com os padrões estabelecidos pelos ambientes de trabalho *Windows, MacOS* ou *Linux*.

Problemas na interação e na interface são os aspectos mais avaliados na área de IHC. Na avaliação destes aspectos, o avaliador pode contar ou não com a participação dos usuários para coletar dados relacionados ao uso de sistemas interativos. A análise dos dados permite identificar os problemas na interação e na interface que prejudiquem a qualidade de uso do sistema. Tais problemas são classificados de acordo com sua gravidade, com a frequência em que tendem a ocorrer e com os fatores que compõem os critérios de qualidade de uso prejudicados: usabilidade, experiência do usuário, acessibilidade ou comunicabilidade.

Todos os objetivos de uma avaliação de IHC são detalhados em perguntas mais especificas para torná-los operacionais (Sharp, 2007, apud BARBOSA; DA SILVA, 2011). O quadro 1 apresenta exemplos de perguntas associadas aos objetivos de avaliação de IHC descritos anteriormente.

Objetivos	Exemplos de perguntas
Analisar a apropriação da tecnologia.	De que maneira os usuários utilizam o sistema? Em que difere do planejado? Como o sistema interativo afeta o modo de as pessoas se comunicarem e relacionarem?
Comparar ideias e alternativas de *design.*	Quais das alternativas é a mais eficiente? Mais fácil de aprender?
Verificar a conformidade com um padrão.	O sistema está de acordo com os padrões de acessibilidade do W3C? A

	interface segue o padrão do sistema operacional?
Identificar problemas na interação e interface.	Considerando cada perfil de usuário: O usuário consegue operar o sistema? Ele atinge seu objetivo? Com quanta eficiência? Em quanto tempo? Após cometer quantos erros? Que parte da interface e da interação o deixa insatisfeito? Que parte da interface o desmotiva a explorar novas funcionalidades?

Quadro 1 – Exemplo de perguntas de um avaliação IHC

Fonte: Barbosa; Da Silva (2011).

Para começar uma avaliação, o avaliador primeiramente deve entender o contexto em que o sistema é ou será utilizado; estar ciente da atual situação, incluindo o domínio do problema, os papéis e perfis dos usuários; e conhecer as interfaces complementares ou semelhantes em que os usuários estão habituados a utilizar. Se possível, saber quais os comportamentos e dificuldades que os mesmos possuem na utilização de sistemas interativos. Com todo esse conhecimento obtido, já é possível conduzir uma avaliação que poderá fornecer resultados úteis e confiáveis. A avaliação de IHC possui as seguintes atividades: preparação, coleta, interpretação, consolidação e relato dos resultados.

A **preparação** é definida pelas respostas obtidas através de questões mais especificas, mostrado anteriormente no quadro 1, onde o avaliador passa a compreender melhor as motivações dos avaliados e ajuda a definir adequadamente os objetivos da avaliação com base nas respostas. Em seguida, são delimitadas as partes da interface aptas a serem avaliadas, o perfil, a quantidade de participantes e o método da avaliação, baseado em inspeção (onde o próprio avaliador se coloca no lugar do usuário para antever as possíveis consequências das decisões de *design*) ou baseado em observação (coleta os dados sobre a situação em que os participantes realizam suas atividades). A preparação consiste também em alocar pessoal, recursos, ambiente, equipamentos e elaborar termos de consentimentos e questionários.

A **coleta de dados** é realizada de acordo com a preparação da avaliação e do método de avaliação escolhido. No método de avaliação por inspeção, são envolvidos apenas avaliadores que seguem o procedimento, examinando a interface para identificar e prever experiências de uso. No método de avaliação por observação, a atividade consiste em observar e registrar todas as experiências apresentadas na interação com os usuários. Neste último, o avaliador apresenta aos participantes o laboratório, a sala de observação, explicações sobre os objetivos do estudo, o sistema de interesse, o procedimento da avaliação, esclarecer dúvidas e por fim, entregar os termos de consentimentos e os formulários pré-testes. A sessão de observação é iniciada, com os *softwares* e *hardwares* registrando os dados, com os avaliadores tomando notas de tudo que ocorre entre humano-computador. Terminada a sessão, o formulário pós-teste é entregue para coletar opiniões e experiências do participante.

A atividade de **interpretação** embasa o avaliador analisar e interpretar individualmente. Esse tipo de análise pode ser feito de forma automática e manual.

De forma automática, são utilizados programas que fazem uma monitoração de toda a interação, registrando todo o caminho que o avaliado fez enquanto ele tentava atingir o objeto esperado. Na forma manual, o avaliador calcula os resultados, comparando os usuários que conseguiram e os que não conseguiram atingir as metas, identificando as partes da interação que tiveram mais problemas.

A **consolidação dos dados e relatos dos resultados** vem logo após ser concluída a interpretação individual dos dados coletados, onde os resultados individuais são consolidados e analisados em conjunto. Os resultados mais comuns referentes a um grupo de participantes possibilitam fazer distinções de características. Os avaliadores relatam os resultados consolidados, incluindo os objetivos e escopo da avaliação; o método de avaliação empregado, o perfil e a quantidade de usuários, um sumário dos dados coletados com tabelas e gráficos, um relato da interpretação e análise dos dados; uma lista dos problemas encontrados e por fim, um planejamento para o reprojeto do sistema.

2.3 Desenvolvimento distribuído de *software* (DDS)

Atualmente, é possível perceber que empresas, visando elevar o nível de qualidade de seus produtos ou reduzir custos, trabalham com equipes de diferentes

localidades. A esta configuração de equipes de profissionais dispostas de forma dispersa chamamos Desenvolvimento Distribuído de *Software* (DDS), onde equipes e ambientes são distribuídos em países, ou em regiões diferentes do mesmo país, trabalhando no desenvolvimento dos módulos de um mesmo projeto (CARMEL, 1999 apud PRIKLADNICKI, AUDY, 2008).

Um ambiente DDS possui modelos de negócios que apresentam características específicas em relação às atividades. Prikladnicki (2008) caracteriza os modelos quanto ao relacionamento entre empresas da seguinte maneira:

Outsourcing: Onde a empresa contrata uma outra empresa para delegar uma ou mais atividades a ela, ou seja, a terceirização. Segundo o autor, é uma das formas mais implementada de ser operacionalizada.

Insourcing: Onde a empresa possui seus próprios departamentos para desenvolverem seus projetos internamente. É o modelo que demanda uma operacionalização mais complexa e mais tempo.

E quanto à distância geográfica, os modelos podem ser definidos como:

Offshore: Quando o desenvolvimento é localizado em um país diferente da empresa contratante, podendo ser um departamento de desenvolvimento da própria empresa matriz ou uma empresa terceirizada. No caso de desenvolvimento distribuído em mais de um continente, temos o que se chama de Desenvolvimento Global de Software (GSD ou *Global Software Development*).

Onshore: A contrário do *offshore*, o desenvolvimento concentra-se no mesmo país.

Buscando uma visão mais unificada, os modelos são combinados, sendo apresentados como:

Onshore Insourcing: departamento dentro da empresa ou uma subsidiária da empresa no mesmo país.

Offshore Insourcing: subsidiária da empresa para prover serviços de desenvolvimento de *software* em um país diferente da empresa contratante.

Onshore Outsourcing: contratação de uma empresa terceirizada localizada no mesmo país da empresa contratante.

Offshore Outsourcing: contratação de uma empresa terceirizada localizada em um país diferente da contratante.

Gerenciar projetos de DDS é uma tarefa complexa, não somente devido ao crescimento dos projetos, mas devido às barreiras criadas por esse tipo de ambiente.

Prikladnicki (2008) destaca as seguintes características da distribuição geográfica como sendo relacionadas à eficiência das tarefas dos projetos:

Distância geográfica: Esse critério define o quão distante se encontram as equipes envolvidas e suas respectivas áreas. Uma equipe dispersa no mesmo local (cidades ou estados diferentes) é diferente de uma equipe dispersa globalmente (países diferentes). A distância entre equipes apresenta níveis de dispersão que envolvem a decisão de distribuir ou não um projeto e para qual unidade distribuída o projeto será enviado, tendo por base análises de risco e custo-benefício envolvidos. Apesar da tecnologia permitir um contato mais próximo entre as equipes e a organização, ainda existem algumas limitações, como por exemplo, obter um consenso de todos em uma tomada de decisões no momento que é preciso reunir os membros distantes do projeto, principalmente aos que trabalham em organizações diferentes. É uma tarefa difícil que consumirá bastante tempo.

Distância temporal: Há diferenças nos horários de trabalhos, fusos horários ou ritmo de trabalho nos membros de uma equipe dispersa. Os horários das localidades envolvidas, bem como os horários disponíveis de trabalho, serão incompatíveis, que acaba acarretando em atrasos no cronograma do projeto. Mesmo adotando estratégias de desenvolvimento 24 horas contínuas, como *Follow-the-Sun* (FTS) que consiste quando uma equipe encerra suas horas normais de trabalho outra equipe dispersa geograficamente com o fuso horário diferente assume as tarefas (KROLL; AUDY; PRIKLADNICKI, 2011), poderá amenizar os fatores que envolvem a distância temporal, mas não soluciona totalmente. Para gerenciar várias equipes com fusos-horários diferentes é necessário um complexo planejamento, além de ter que lidar com a comunicação informal e os fatores que acarreta a distância cultural entre equipes.

Distância cultural: O nível de distância cultural existente entre as equipes afeta na performance de projetos distribuídos. Diferenças culturais como o idioma, as tradições, os costumes, as normas, os comportamentos ou determinadas ações podem causar conflitos, já que cada um pode empenhar e enxergar de forma diferente, acreditando que a sua própria percepção é a mais correta. O desafio em ambiente com multiculturas está em tentar encontrar um equilíbrio entre encorajar o fluxo de ideais e o controle das diferenças culturais.

Carmel (1999 apud PRIKLADNICKI, 2002) aponta duas forças que contribuem para o sucesso ou fracasso da formação de equipes distribuídas de desenvolvimento

de *software*: as forças centrífugas e centrípetas. As forças centrífugas são constituídas dos fatores: falta de coordenação, dispersão geográfica, comunicação ineficiente, diferenças culturais e perda do espírito de equipe; e resultam em um afastamento entre os membros e péssima performance da equipe. As forças centrípetas, consistem nos fatores: arquitetura do produto, construção de equipe, técnica de gerência, tecnologia de colaboração, infraestrutura de comunicação e metodologia de desenvolvimento; e aproximam a equipe contribuindo para que a tarefa apresente resultados satisfatórios.

O processo de desenvolvimento de um *software* é a peça-chave para a construção de um produto com qualidade. Além dos problemas apresentados anteriormente (distância geográfica, temporal e cultural), o DDS apresenta desafios focados no processo de desenvolvimento. Tais desafios, listados por Rafael Prikladnicki e Jorge Audy (2008) são:

Arquitetura do *software*: A arquitetura de *software* em um desenvolvimento co-localizado difere de um distribuído. Ela deve ser baseada no princípio da modularidade, que permite um desenvolvimento paralelo podendo reduzir a complexidade do projeto, sendo assim, determinante na efetividade das dificuldades do DDS (PRIKLADNICKI; AUDY, 2008).

Engenharia de requisitos: Em DDS, a engenharia de requisitos torna-se mais complexa porque contém tarefas que necessitam de alto nível de comunicação e coordenação.

Gerência de configuração: É difícil gerenciar as modificações simultâneas em cada um dos locais distribuídos. Apesar de existir ferramentas no mercado que suportam a gerência de configuração distribuída, a maioria da empresa opta por um controle manual em cada local disperso, gerando problemas inesperado no projeto.

Processo de desenvolvimento: No DDS são críticas a falta de sincronização e a padronização das atividades nas diversas localidades durante o processo de desenvolvimento. Sem uma sincronização nas tarefas, uma equipe não poderá saber em que ponto do processo o projeto se encontra.

3 SISTEMA DE GERENCIAMENTO DE PROJETOS (SGP)

Esse capítulo descreve a ferramenta Sistema de Gerenciamento de Projetos (SGP) retratando detalhes de especificações, funcionalidades e tecnologias empregadas para a construção do sistema.

3.1 A ferramenta

O SGP é um sistema que gerencia projetos de softwares locais ou distribuídos, destinados ao desenvolvimento iterativo e incremental de um produto, baseado em técnicas específicas de métodos ágeis, principalmente, o *Scrum*. Trata-se de uma ferramenta que auxilia a equipe de desenvolvimento a atingir o objetivo de um projeto. Desenvolvido para a plataforma *web* por um grupo de alunos do curso de Bacharelado em Sistemas de Informação na Universidade Federal do Piauí, Campus Senador Helvídio Nunes de Barros, Picos-PI; onde foi definido o S*crum* (vide tópico 2.1.2.1) como a metodologia ideal ao desenvolvimento da aplicação por se tratar de um sistema complexo, modularizado e que está em constante modificações.

A linguagem de programação utilizada para o desenvolvimento do protótipo da aplicação foi *Ruby,* com o *framework Ruby on Rails*. A escolha da linguagem *Ruby* por deve-se a ela ser orientada a objetos, com sintaxe simples e enxuta, focada na produtividade. O *Ruby on Rails* por ser um *framework* que incentiva a fazer um bom uso da reutilização de código, além de permitir que funcionalidades de um sistema posse ser implementada de maneira incremental, sendo assim, uma escolha óbvia a metodologia ágil adotada para o SGP.

Como toda aplicação desenvolvida em *Rails* tem em suas estruturas bases a arquitetura MVC (*Model-View-Controller*), consequentemente o SGP segue esse padrão de arquitetura, que de forma clara, a camada *model* é a responsável pela manipulação de dados e informações que o usuário trabalha, a camada *view* responsável pela interface do usuário e o *controller* como intermediador, fazendo assim a comunicação direta de ambas as camadas.

O banco de dados da aplicação é o *SQLite3*, já inclusa no *Ruby on Rails*, gerando de maneira fácil, arquivos de configurações prontos para fazer a comunicação da aplicação.

Os elementos visuais que formam o seu *layout* foram construídos utilizando o *framework front-end Bootstrap,* por oferecer uma gama de componentes amigáveis e modernos, responsivos e que se integram perfeitamente com a linguagem *Ruby*.

3.2 Funcionalidades

O SGP é dividido em 2 subsistemas: um para o administrador, que terá o controle das atividades que compõe um projeto e o outro para o usuário comum, que poderá visualizar as tarefas que realizou e as que estão ainda em aberta. O administrador, após realizar seu *login*, será apresentado à **tela de projetos**, ilustrada pela Figura 9, contendo todos os projetos gerenciados por ele. Os projetos serão listados em campos próprios, contendo nome, data de início, data final e o progresso. Nesta mesma tela, poderá criar um novo projeto.

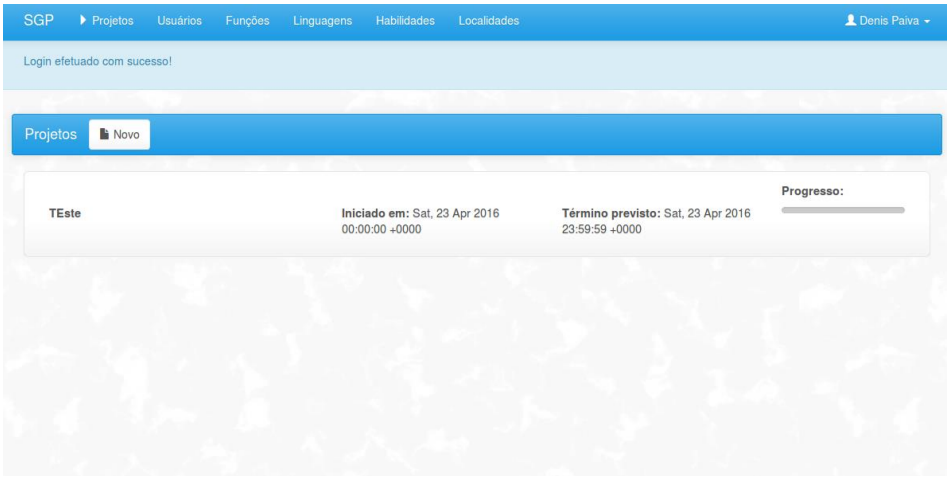

Figura 9 – Tela de projetos

Fonte: O autor (2016).

A Figura 10 mostra a **tela de novo projeto**. Ao salvar um projeto recém-criado, ele será listado juntamente com os outros projetos ativos na tela de projetos.

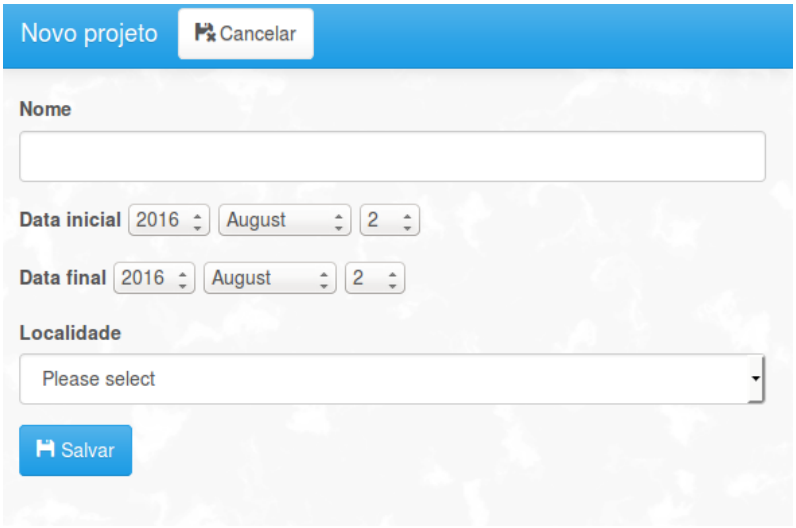

Figura 10 – Tela de novo projeto

Fonte: O autor (2016).

Ao entrar em um projeto específico, mostrará as funcionalidades que compõe um projeto: ***Product Backlog***, ***Sprints***, ***Releases***, **Membros, Temas, Critérios de aceitação, Tarefas** e **Habilidades Requeridas da Tarefa**, ilustrado pela Figura 11.

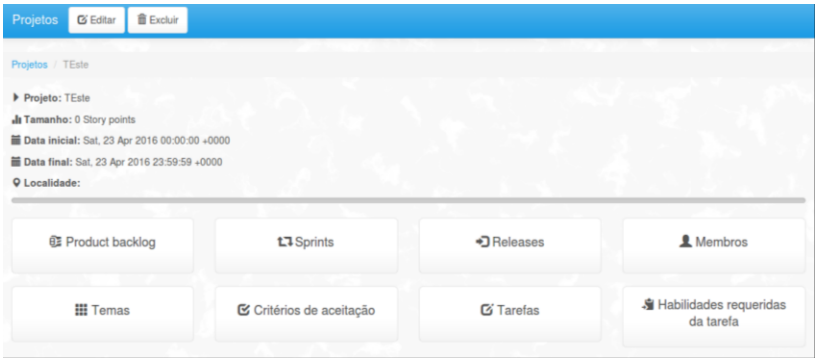

Figura 11 – Tela específica de um projeto

Fonte: O Autor (2016).

A **funcionalidade *Product Backlog*** conterá todas as *users stories* (histórias de usuários) de um projeto. Ao criar uma *user story* (Figura 12), é preciso informar uma descrição para a história; valor de negócio, que define o quão importante aquela *user story* é para o projeto; *story points*, estimativa do custo para implementá-la;

status, se a história estará pendente ou ativa; e o tema, que informa qual módulo a história pertencerá.

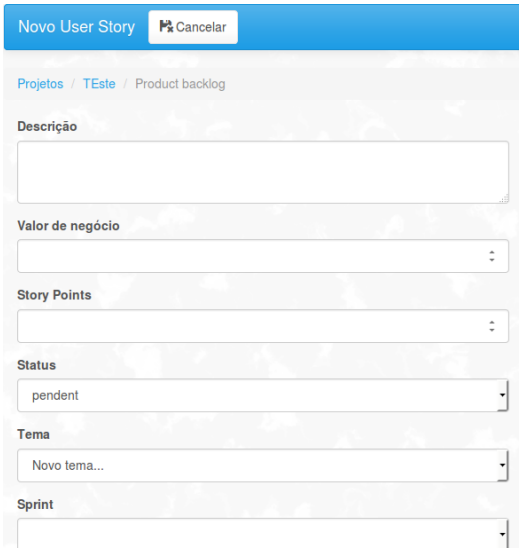

Figura 12 – Tela Novo *User Story*

Fonte: O autor (2016).

A **funcionalidade *Sprint*** lista todos os ciclos de desenvolvimentos ativos. Ao criar um *sprint* (Figura 13) é preciso informar as datas iniciais e de términos de um *sprint*, da reunião de planejamento, do período de execução/desenvolvimento, da reunião de revisão e retrospectiva. Por fim, definir a *release*, uma versão funcional prototipada do produto final.

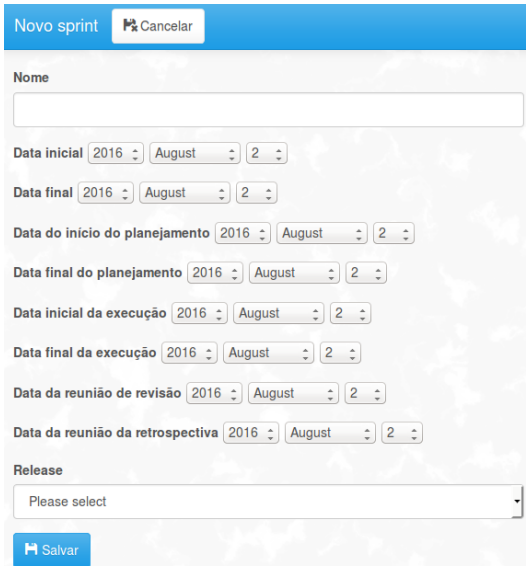

Figura 13 – Tela Novo *Sprint*

Fonte: O autor (2016).

A **funcionalidade *Release*** mostra todos os módulos concluídos do projeto. Quando um módulo fica pronto no encerramento de um s*print*, deverá ser registrado na tela Novo *Release* (Figura 14) contendo a versão e a data que foi concluída.

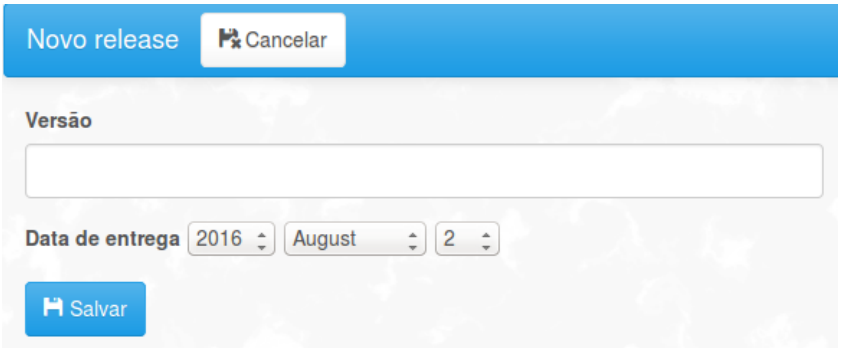

Figura 14 – Tela Novo *Release*

Fonte: O autor (2016).

A **funcionalidade Membros** lista todos os usuários envolvidos com o projeto (Figura 15). Cada membro adicionado, deverá ter uma função, como programador, *designer*, modelador de banco de dados, avaliador; e definir o tipo de usuário, se é um administrador ou usuário comum (Figura 16).

Figura 15 – Tela Membros do projeto

Fonte: O autor (2016).

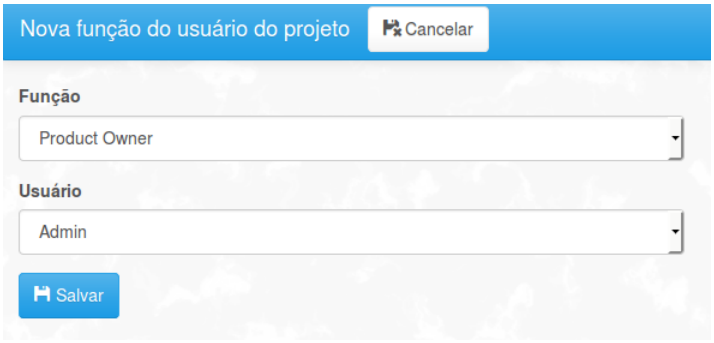

Figura 16 – Tela de cadastro de função a um membro

Fonte: O autor (2016).

Tema é um conjunto de *user stories* de um mesmo contexto, que quando agrupados, formam um módulo do projeto. A **funcionalidade Tema** (Figura 17) lista todos os temas. Esses temas serão selecionados ao criar as *user stories* lá na função *product backlog*.

Figura 17 – Tela de temas

Fonte: O autor (2016).

Um critério de aceitação é o detalhamento de um requisito que define se a funcionalidade é considerada válida ou não para ser implementada. A funcionalidade **Critérios de aceitação** (Figura 18) lista todos os critérios criados. Esses critérios serão selecionados ao criar as *user stories* na função *Product Backlog*.

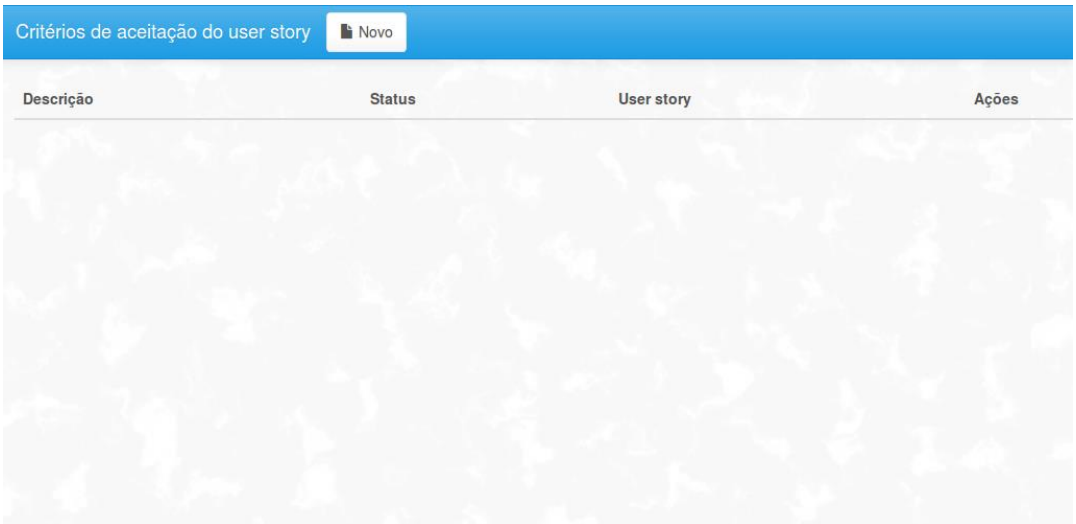

Figura 18 – Tela de critérios de aceitação

Fonte: O autor (2016).

A **funcionalidade Tarefas** (Figura 19), o administrador poderá definir as tarefas para os membros. O membro ao logar com sua conta de usuário comum, saberá que há uma tarefa disponível para ser concluída.

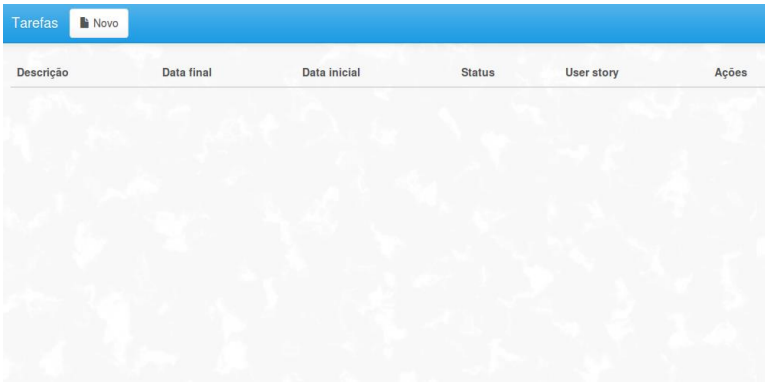

Figura 19 – Tela de tarefas

Fonte: O autor (2016).

A tarefa para ser construída requer que o usuário tenha a habilidade correspondente. Uma tarefa do tipo “modelar banco de dados” é preciso especificar que somente os usuários com funções de *Data Base Administrator* (DBA) possa empenhá-la. A **funcionalidade Requerimento da tarefa** (Figura 20) lista todos os requerimentos de tarefas criados. Esses requerimentos serão selecionados ao criar tarefas na função tarefas.

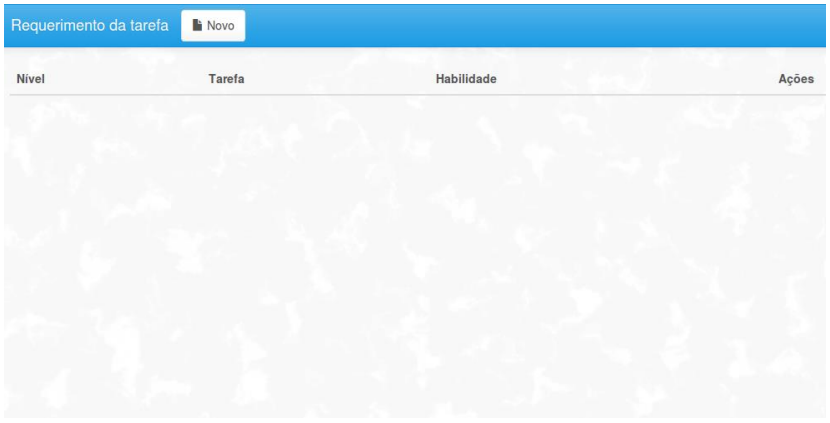

Figura 20 – Tela de requerimento da tarefa

Fonte: O autor (2016).

Um usuário comum, ao realizar o *login* do sistema, acessará o segundo subsistema do SGP. Todos os projetos em que ele participa serão listados, mostrando todas as atividades concluídas e as que ainda faltando serem concluídas por ele. Só serão exibidas atividades que correspondem à sua função, ou seja, caso o membro tenha a função de programador, mostrará somente atividades para codificação.

4 PROPOSTA DE INTERFACE PARA A FERRAMENTA SISTEMA DE GERENCIAMENTO DE PROJETOS (SGP)

Toda a proposta da interface do SGP foi descrita por meio do guia de estilo descrito neste tópico. Um guia de estilo, segundo Marcus (1992, apud BARBOSA; DA SILVA, 2011) é um documento contendo as características de *design* decididos para uma interface. A concepção do guia serviu como abordagem de orientação do *designer* com os desenvolvedores, incitando a criação de páginas que assegurassem todos os elementos necessário para uma boa interação centrado no usuário final da aplicação.

4.1 Guia de estilo da interface do Sistema de Gerenciamento de Projetos (SGP)

4.1.1 Descrição do ambiente

O SGP é uma aplicação *web*, e dessa forma, necessitará de um navegador *desktop* ou *mobile* com conexão para acessá-lo. Todos os elementos e componentes foi implementada com CSS, uma "folha de estilo" utilizada para definir a aparência em páginas da *internet,* utilizando os componentes do *framework front-end bootstrap*.

4.1.2 Elementos de interface

O escopo da tela inicial da aplicação foi abstraído em um rascunho de papel, como mostra a Figura 21. Tal rascunho, denominado *mockup,* é o modelo de representação consolidado de como os elementos serão apresentados e distribuídos. A partir dele, serviu de referência para criar a página inicial da aplicação, apresentando uma interface simples e minimalista, agregando textos breves escrito de forma compreensível, desenhos com arte amigável que condiz com as funcionalidades oferecidas.

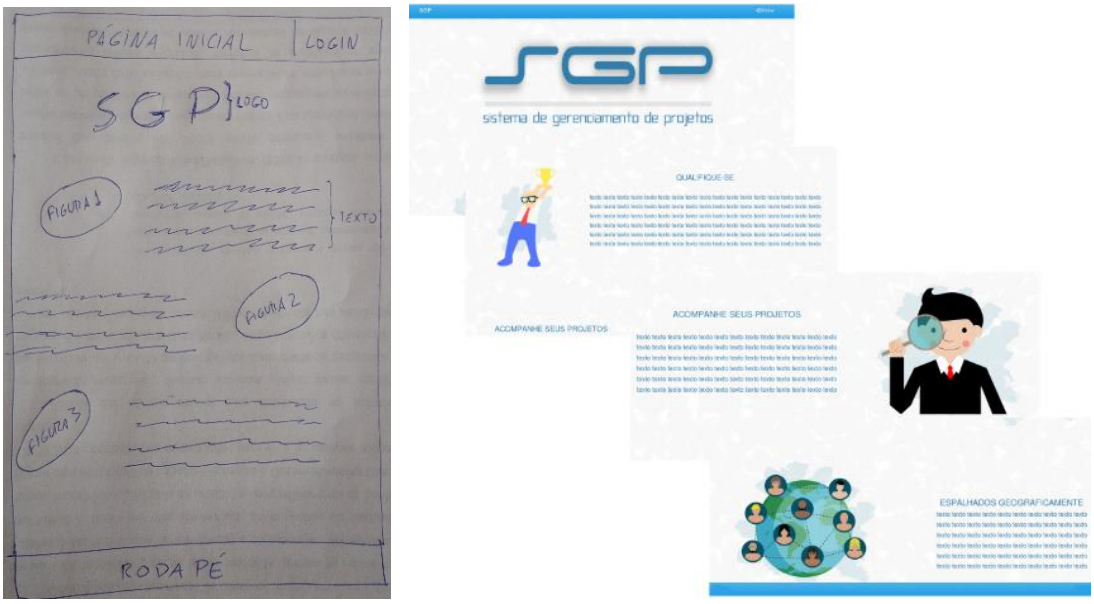

Figura 21 – Rascunho e página inicial

Fonte: O autor (2016).

A interface é dividida em três *grids*: cabeçalho, conteúdo e rodapé (Figura 22). As informações contidas no cabeçalho e no rodapé têm prioridade sobre a área de conteúdo, sendo sempre visualizadas não importando em que página o usuário está.

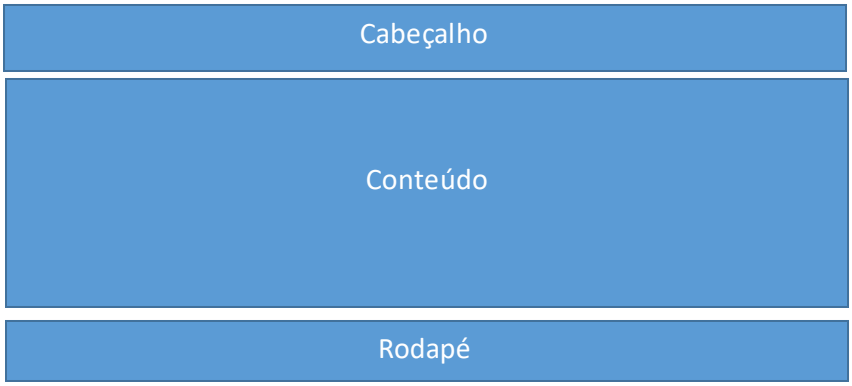

Figura 22 – Divisão do *layout*

Fonte: O autor (2016).

Todas as páginas seguirão essa divisão de *layout*, com exceção da tela de projetos do usuário comum que possui uma *grid* a mais, um painel lateral que com os dados cadastrais e status do membro (Figura 23).

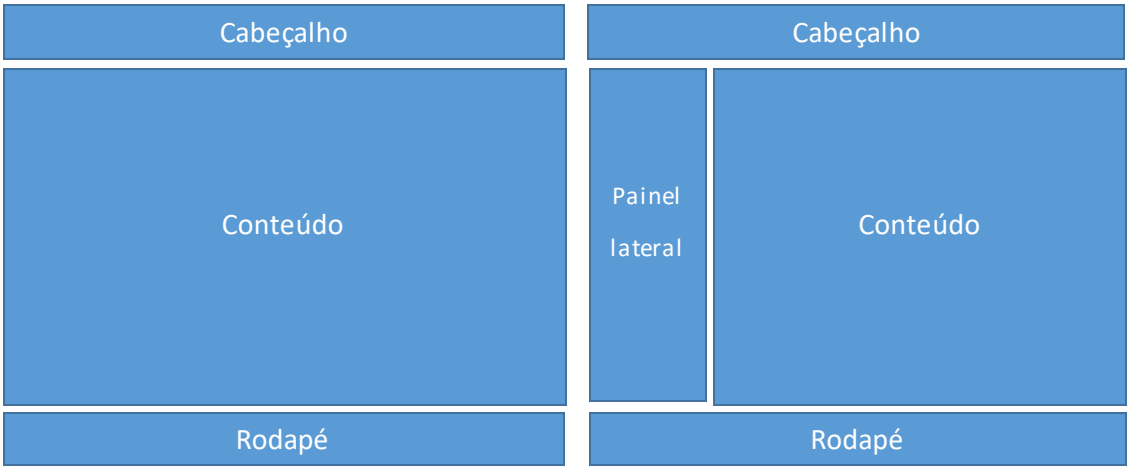

Figura 23 – Diferentes telas de projetos

Fonte: O autor (2016).

4.1.3 Tipografia

As fontes utilizadas é uma variante da família *Sans Serif,* a *Open Sans Regular* (Figura 24), fonte que apresenta um visual amigável e uma excelente legibilidade.

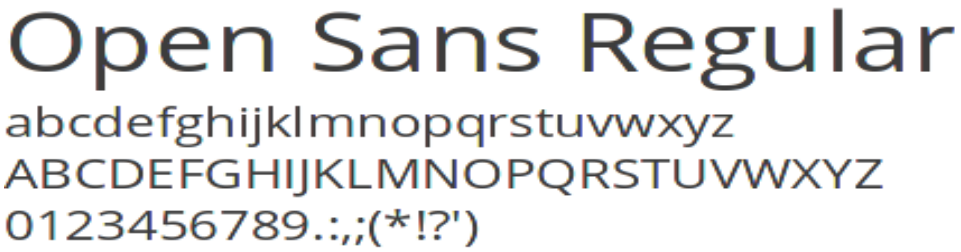

Figura 24 – Fonte *Open Sans Regular*

Fonte: O autor (2016).

4.1.4 Ícones

Os ícones escolhidos para o sistema (Figura 25) são do pacote *glyphicon,* retirados do *bootstrap*. Tais ícones apresentam visuais minimalistas que não poluem esteticamente o *layout* e formas que se associa perfeitamente com as funcionalidades da aplicação.

Figura 25 – Ícones do sistema

Fonte: O autor (2016).

4.1.5 Imagens

A imagens que compõem a tela inicial são em formatos *scalable vector graphics* (svg). A escolha deste formato é justamente por permitir redimensionamento sem perda de qualidade e nitidez.

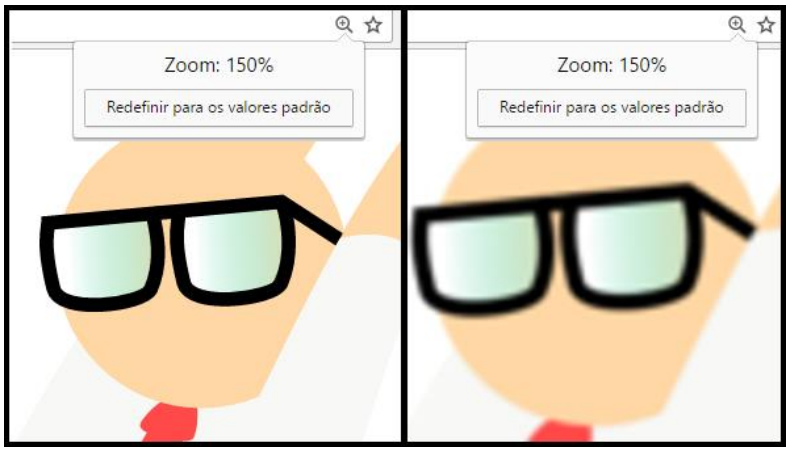

Figura 26 – Imagens sgv e png respectivamente

Fonte: O autor (2016).

4.1.6 Cores

Na Figura 27 são demonstradas as cores escolhidas para a aplicação. A cor predominante é o azul variado em diversas intensidades. O tom azul, segundo a teorias das cores, transmite harmonia, tranquilidade e serenidade. De modo técnico, é uma cor que não perde tonalidade com o excesso de luminosidade ou falta dele, sendo o ideal para se comportar em diferentes níveis de brilhos das telas dos computadores.

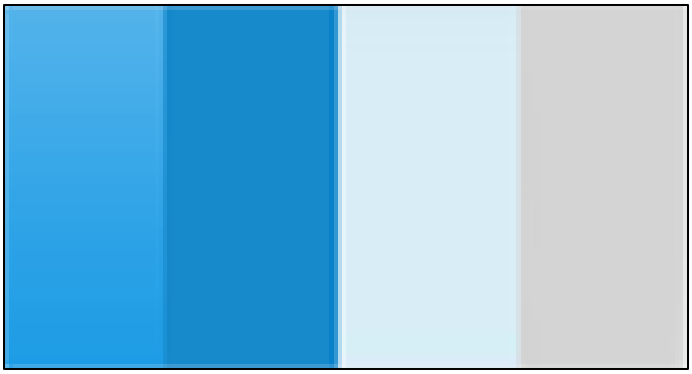

Figura 27 – Paleta de cores utilizada no sistema

Fonte: O autor (2016).

4.1.7 Botões e efeitos

Os botões, menus e campos contêm animações que indica profundidade para transmitir a mensagem de que foi clicado (no caso de ser um botão) ou está em ênfase (campo ou menu). Todos os efeitos baseiam-se na paleta de cores definidos no item 4.1.6.

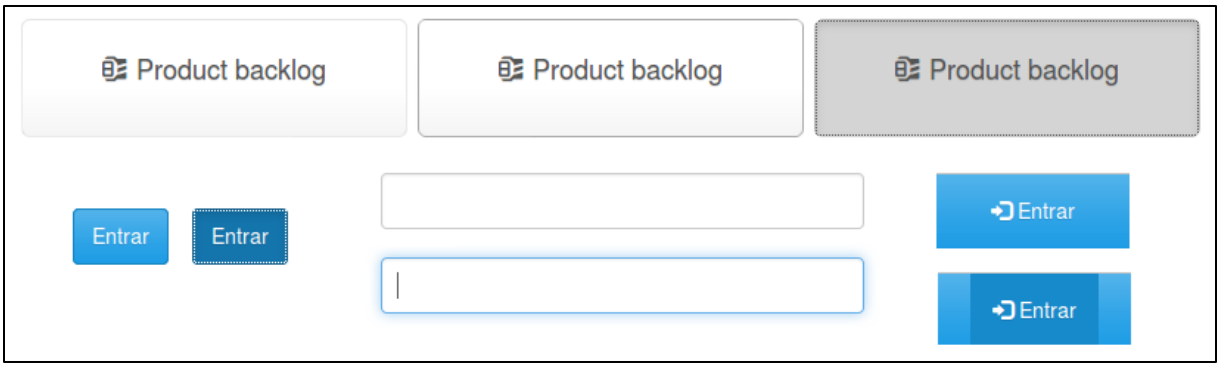

Figura 28 – Botões, campos e menus com efeitos

Fonte: O autor (2016).

4.1.8 *Background*

A aplicação utiliza como background figuras de formas indefinidas misturando-se com uma cor cinza transparente com o propósito, não somente estético, mas de não causar incômodo aos olhos em uma exposição prolongada. Aplicações com fundo branco causam efeitos de irritação aos olhos, mesmo diminuindo o brilho do monitor.

Figura 29 – *Background* utilizada no sistema

Fonte: O autor (2016).

4.1.9 Mensagens

O sistema trata 3 tipos principais de mensagens: erro, sucesso e alertas. A mensagem de erro indica qual tipo de erro ocorreu, como “Erro ao efetuar essa operação!”, o de alerta indica que algo de errado pode acontecer caso o usuário realize alguma ação. Exemplo: “Preencha todos os campos obrigatórios no formulário!” e a mensagem de sucesso, que indica que a ação intentada pelo usuário foi concluída com sucesso. Exemplo: “O projeto foi salvo com sucesso!”.

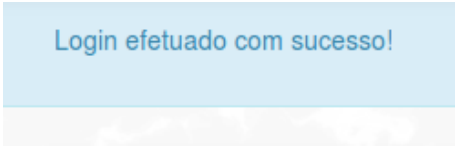

Figura 30 – Mensagem de sucesso

Fonte: O autor (2016).

4.2 Resultados

Quando o guia de estilo foi elaborado, foi comunicada adequadamente sua existência para os demais envolvidos do projeto do SGP. Toda a estrutura do guia foi colocada em prática, resultando em uma grande repaginação e evolução na interface. Antes, haviam telas sem padrão algum de interatibilidade, com elementos básicos e informações soltas, passaram a possuir um *layout* elegante, harmônico, informativo e que atende os propósitos que a aplicação tem a oferecer ao usuário final. Abaixo, na

sequência de figuras, demonstra as diferenças de telas antes e depois de seguir o guia de estilo.

Figura 31 – Tela inicial antes e depois

Fonte: O autor (2016).

Figura 32 – Tela de acesso antes e depois

Fonte: O autor (2016).

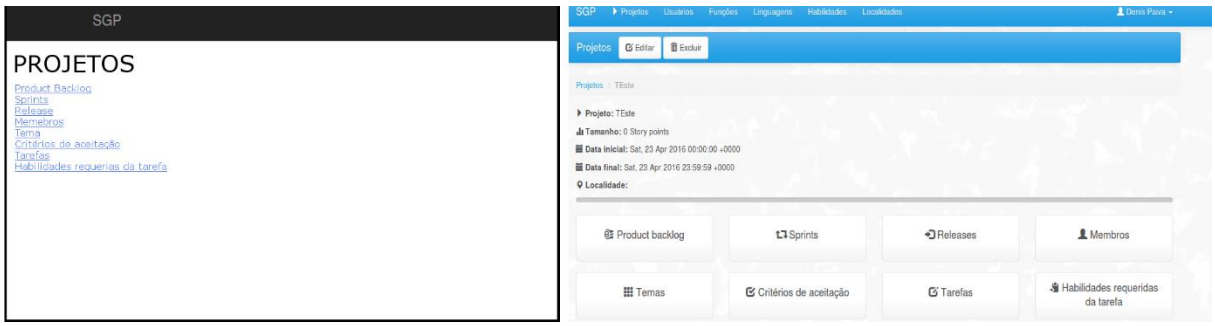

Figura 33 – Tela de projetos antes e depois

Fonte: O autor (2016).

4.2.1 Avaliação da interface

Para identificar os efeitos positivos e negativos em relação ao uso da interface do Sistema de Gerenciamento de Projetos (SGP), foi realizada uma avaliação com um grupo de 15 (quinze) alunos do curso de Sistemas de Informação da Universidade Federal do Piauí – UFPI, Picos-PI; com faixa etária entre 18 e 21 anos, no laboratório de Sistemas de Informação - UFPI. O sistema foi apresentado aos participantes, com uma explicação sobre cada módulo e suas funcionalidades, um termo de consentimento (Anexo I) e um questionário avaliativo (Anexo II) denominado SUMI (*Software Usability Measurement Inventory*) desenvolvido pelo grupo *Human Factors Research Group* (HFRG) da *University of Nottingham* - Reino Unido. O questionário original do SUMI, encontrado em seu site oficial (http://sumi.ucc.ie/en/) possui 50 perguntas em inglês com respostas escalares correspondentes a três níveis: “concordo”, “discordo” e “indeciso”. Para a aplicação desta avalição, foram adaptadas 20 (vinte) das 50 (cinquentas) perguntas oferecidos pelo SUMI que mais se encaixavam com o perfil do SGP com os participantes, além de uma tradução coerente para o português para melhor compreensão para os participantes. Após o término da avaliação e do preenchimento dos questionários, houve uma discussão para apontar as dificuldades encontradas por cada um em alguns elementos do sistema e sugerir melhorias de *design.* Os resultados obtidos no teste avaliativo foram:

QUANTIDADE DE USUÁRIOS			RESULTADO
Concordaram	Discordaram	Indecisos	
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que o *software* responde rapidamente suas requisições.
15 (nenhum)	0 (nenhum)	0 (nenhum)	Responderam que as instruções oferecidas são úteis.
9 (nove)	6 (seis)	0 (nenhum)	Responderam que apresentaram facilidade ao utilizar o *software* pela primeira vez.
10 (dez)	4 (quatro)	1 (um)	Responderam que houve momentos que sabia realmente o que estavam fazendo com o *software.*

12 (doze)	3 (três)	0 (nenhum)	Responderam que levaram pouco tempo para aprender as funções do *software.*
7 (sete)	8 (oito)	0 (nenhum)	Responderam que não sentiram-se em dúvida ao usar uma função.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que as informações do sistema são apresentados de maneira clara e compreensível.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que sentiram-se seguro com as funções familiares (excluir, editar, novo).
6 (seis)	4 (quatro)	5 (cinco)	Responderam que usaria este *software* frequentemente.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que há pouco texto para ler antes de poder utilizar alguma funcionalidade.
2 (dois)	10 (dez)	3 (três)	Responderam que não sentiram nenhuma frustação ao utilizar o *software*.
10 (dez)	4 (quatro)	1 (um)	Responderam que o software ajuda a superar todos os problemas que teve em usá-lo.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que sabia que tipo de entrada era necessário para preencher algum campo do sistema.
10 (dez)	5 (cinco)	0 (nenhum)	Responderam que a organização dos menus parece bastante lógico.
13 (treze)	2 (dois)	0 (nenhum)	Responderam que o *software* permite que o usuário utilize pouco as teclas do teclado
9 (nove)	6 (seis)	0 (nenhum)	Responderam que há poucos passos necessários para concluir a tarefa.
14 (catorze)	1 (um)	0 (nenhum)	Responderam que as mensagens de erros são adequadas.
10 (dez)	4 (quatro)	1 (um)	Responderam que acha que aprenderá a usar tudo o que é oferecido neste software.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que a interface do software apresentando é atraente.
15 (quinze)	0 (nenhum)	0 (nenhum)	Responderam que é relativamente fácil navegar de uma funcionalidade para outra.

Quadro 2 – Resultados da avaliação

Fonte: O autor (2016).

A partir dos resultados da coleta de dados, foi gerado um gráfico em relação ao teste avaliativo da interface do sistema:

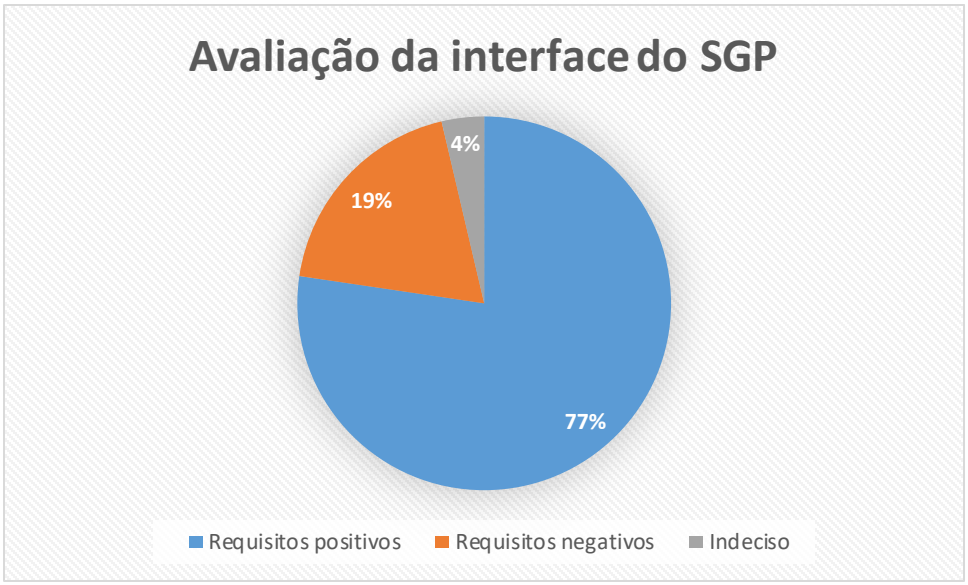

Figura 34 – Gráfico percentual da avaliação da interface

Fonte: O autor (2016).

A Figura 34 demonstra que 77% dos avaliados afirmaram que a interface possui requisitos positivos que oferecem uma boa interação ao um usuário, enquanto 19% apontaram que alguns dos requisitos não conduzem de forma correta e 4% encontram-se neutros. O ideal para o produto final, definido pela equipe do sistema, é que a interface tenha a margem de satisfação de uso de no mínimo 90%. Por tratar ainda de um protótipo, as melhorias serão implementadas conforme a evolução do sistema.

A grande maioria dos participantes apontaram como sugestão a adoção de botões de ajuda com breves textos descritivos sobre cada funcionalidade, ou então, um módulo dedicado apenas a um tutorial passo a passo de como gerenciar um projeto e controlar as atividades utilizando o sistema. Levando em consideração as opiniões, isso poderá orientar usuários mais causais que não estão familiarizados com termos de práticas *scrum*, como *product backlog*, *sprints* e *releases.*

5 CONCLUSÃO

Este trabalho foi desenvolvido com a finalidade de apresentar uma interface para o Sistema de Gerenciamento de Projetos (SGP) utilizando de um guia de estilo, abordando todo o conjunto de elementos que compõe o seu *design*. O ato de empregar um guia de estilo, trouxe bons resultados para o desenvolvimento da interface da ferramenta, além de ter contribuído em potencializar a equipe de programação o interesse em considerar aspectos de interação no desenvolvimento de *softwares*.

A fim de mensurar a satisfação dos usuários em relação ao uso da interface do SGP, realizou-se uma avaliação de IHC para um grupo de avaliadores identifique quais critérios apresentaram de maneira qualificável a interatibilidade e que retornem suas ideias e opiniões sobre o *design*. Neste estudo, a interface apresentou um resultado ótimo, provendo um certo nível de facilidade na utilização. Porém não o suficiente para que o usuário possa ter total domínio nas tarefas do sistema, sendo necessário atualizar o guia de estilo com as possíveis melhorias dos elementos reportados na avaliação e sugestões.

Como parte da evolução do trabalho futuramente, espera-se a partir da reformulação do guia de estilo e do reprojeto de *design,* realizar uma nova avaliação cujo o resultado alcance o máximo de satisfação do usuário quanto a sua utilização da ferramenta. Pôde-se concluir que a experiência de trabalhar com interfaces muda totalmente a percepção em investigar como os elementos de *design* de uma interface afeta a forma de trabalhar dos utilizadores.

6 REFERÊNCIAS

BARBOSA, Simone Diniz Junqueira; DA SILVA, Bruno Santana. **Interação Humano-Computador.** Rio de Janeiro: Elsevier, 2011.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto para Desenvolvimento Ágil de *Software***, 2001, disponível em <http://agilemanifesto.org/iso/ptbr/manifesto.html>, acessado em: 20 de julho de 2016.

FOWLER, Martin; HIGHSMITH, Jim 2001. ***The Agile Manifesto***, 2001. Disponível em <http://www.drdobbs.com/open-source/the-agile-manifesto/184414755>, acessado em: 20 de julho de 2016.

KROLL, Josiane; AUDY, Jorge L. N.; PRIKLADNICKI, Rafael. **Desmistificando o Desenvolvimento de *Software* Follow-the-Sun: Caracterização e Lições Aprendidas.** In: *WORKSHOP* DE DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE*. 5., 2011, São Paulo. *Artigos aceitos...* São Paulo: Universidade Presbiteriana Mackenzie, 2011. p. 8.

PRESSMAN, Roger S. **Engenharia de *Software.*** 6 ed. São Paulo: McGraw-Hill, 2006.

PRIKLADNICKI, Rafael. **Desenvolvimento Distribuído de *Software* e Processos de desenvolvimento de *Software*.** 2002. 66f. Dissertação (Mestrado) – Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre. 2002.

PRIKLADNICKI, Rafael; AUDY, Jorge. **Desenvolvimento Distribuído de *Software*.** Rio de Janeiro: Elsevier, 2008.

SABBAGH, Rafael. ***SCRUM*: Gestão ágil para projetos de sucesso.** São Paulo: Casa do Código, 2013.

SUTHERLAND, Jeff; SCHWABER, Ken. ***The Definite Guide to Scrum: The Rules of the Game,*** 2013. Disponível em < http://www.scrumguides.org/>, acessado em: 01 de agosto de 2016.

SOMMERVILLE, Ian. **Engenharia de *Software*.** 9 ed. São Paulo: Pearson Prentice Hall, 2009.

SOUZA, Celso Luiz de. **Sistemas de Informação versus usuário.** [Editorial]. Revista e-xata, v.2, n. 2, 2009.

ANEXOS

Anexo I – Termo de Consentimento

UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS – CSHNB

TERMO DE CONSENTIMENTO

Avaliação de Proposta de Interface

Convidamos o(a) Sr.(a) para participar da Pesquisa “Avaliação de Proposta de Interface”, sob responsabilidade do pesquisador Denis Costa Paiva, a qual pretende avaliar a qualidade do protótipo do Sistema de Gerenciamento de Projetos (SGP). Sua participação é voluntária e se dará por meio da utilização do protótipo da interface em um computador, seguido do preenchimento das questões relacionados ao uso da aplicação.

Os riscos decorrentes de sua participação na pesquisa são nulos. Ao aceitar, estará contribuindo para o conhecimento do aluno do curso de Bacharelado em Sistemas de Informação da Universidade Federal do Piauí. Em caso de desistência após o termo de consentimento devidamente assinado, tem o direito de se retirar em qualquer momento da pesquisa, independente do motivo. O Sr. (a) não terá nenhuma despesa e nenhuma remuneração. Os resultados da pesquisa serão divulgados, mas sua identidade será preservada.

Consentimento pós-informativo

Eu, _____, fui informado sobre o que o pesquisador desejar realizar e porque precisa da minha colaboração. Por isso, concordo em participar da avaliação, ciente dos detalhes informados. Este documento será assinado por mim e pelo pesquisador.

_____ _____

Assinatura do participante Assinatura do Pesquisador

Picos-PI, ___ / ___ / ____

ANEXO II

TESTE AVALIATIVO DE MEDIÇÃO DE USABILIDADE

SUMI

As informações fornecidas são mantidas completamente confidenciais. Este questionário possui 50 declarações. Por favor, responder a todas elas. Após cada afirmação, há três caixas.

- Assinale a primeira opção se você concordar com a afirmação.
- Assinale a segunda se você discorda com afirmação.
- Assinale a terceira opção se você está indeciso, ou se a declaração não tem qualquer relevância para o seu software ou para a sua situação.

Questionário 1 de 25	**concordar**	**discordar**	**indeciso**
1 - Este *software* responde lentamente suas requisições?	○	○	○
2 - As instruções oferecidas são úteis?	○	○	○
3 - Apresentou facilidade ao utilizar o *software* pela primeira vez?	○	○	○
4 - Houve momentos que sabia o que estava fazendo com o *software*?	○	○	○
5 - Levou pouco tempo para aprender as funções do *software*?	○	○	○
6 - Não sentiu em dúvida usando uma função?	○	○	○
7 - As informações do sistema são apresentados de maneira clara e compreensível?	○	○	○
8 - Sentiu-se seguro com algumas funções familiares (excluir, editar, novo)?	○	○	○
9 - Usaria este *software* frequentemente?	○	○	○
10 - Há pouco texto para ler antes de poder utilizar alguma funcionalidade?	○	○	○
11 – Não sentiu nenhuma frustação ao usar este *software*?	○	○	○
12 - O software ajuda a superar todos os problemas que teve em usá-lo?	○	○	○
13 – Sabia qual tipo de entrada de dado era necessário para preencher um campo especifico do sistema?	○	○	○

14 - A organização dos menus parece bastante lógico?	○	○	○
15 - O *software* permite que o usuário utilize pouco as teclas do teclado?	○	○	○
16 - Há poucos passos necessários para concluir a tarefa?	○	○	○
17 - As mensagens de erros são adequadas?	○	○	○
18 - Acha que aprenderá a usar tudo o que é oferecido neste software?	○	○	○
19 - A interface do software apresentando é atraente?	○	○	○
20 - É relativamente fácil navegar de uma funcionalidade para outra?	○	○	○

TERMO DE AUTORIZAÇÃO PARA PUBLICAÇÃO DIGITAL NA BIBLIOTECA "JOSÉ ALBANO DE MACEDO"

Identificação do Tipo de Documento

( ) Tese

( ) Dissertação

(X) Monografia

( ) Artigo

Eu, Denis Costa Paiva,

autorizo com base na Lei Federal nº 9.610 de 19 de Fevereiro de 1998 e na Lei nº 10.973 de 02 de dezembro de 2004, a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação Proposta de interface para a ferramenta SGP – Sistema de Gerenciamento de Projetos de minha autoria, em formato PDF, para fins de leitura e/ou impressão, pela internet a título de divulgação da produção científica gerada pela Universidade.

Picos-PI 18 de janeiro de 2017.

Assinatura

Assinatura