Projeto de Trabalho de Conclusão de Curso

Uma ferramenta de apoio na obtenção de Agilidade em Teste de Software

Marco Antônio Lupércio Oliveira Freitas

JUIZ DE FORA

JUNHO, 2017

Uma ferramenta de apoio na obtenção de Agilidade em Teste de Software

Marco Antônio Lupércio Oliveira Freitas

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Marco Antônio Pereira Araújo

JUIZ DE FORA

JUNHO, 2017

Resumo

A cada dia as tecnologias estão mais inseridas nas atividades do cotidiano das pessoas e existe, cada vez mais, um forte apelo ou necessidade do mercado para que os softwares possam absorver constantes mudanças e que estas ocorram de forma rápida e com qualidade. E é exatamente neste ponto que a área de Teste de Software tem muito a contribuir. Além de ser fortemente responsável por garantir a qualidade de um produto, também é apontada como grande responsável pelo tempo e custo total de um projeto.

Assim, este trabalho visa a construção de uma ferramenta para apoiar a decisão de quais Práticas Ágeis são mais adequadas para serem utilizadas a partir das Características básicas de um projeto, visando identificar se o projeto é ou não indicado para utilização destas práticas e visando também obter maior agilidade no processo de Teste de Software: sendo mais ágil, visando um menor custo e sem perda de qualidade.

Palavras-chave: Teste de Software, Agilidade, Características ágeis, Práticas ágeis.

Abstract

Every day the technologies are more inserted in the daily activities of the people and there is, increasingly, a strong appeal or need of the market so that the software can absorb constant changes and that these occur quickly and with quality. And it is precisely at this point that the Software Test area has much to contribute. In addition to being strongly responsible for ensuring the quality of a product, it is also pointed out as being largely responsible for the time and total cost of a project.

Thus, this work aims at the construction of a tool to support the decision of which Agile Practices are most appropriate to be used from the basic characteristics of a project, in order to identify whether or not the project is indicated to use these practices and also to obtain greater agility in the process of Software Testing: being more agile, aiming a lower cost and without loss of quality.

Keywords: Software Testing, Agile, Agile Characteristics, Agile Practices.

Agradecimentos

Primeiramente a Deus, por permitir que tudo isso pudesse acontecer.

À minha mãe Simone, meu pai Marco Aurélio e minhas irmãs Michelle e Monique pelo encorajamento e apoio.

Ao professor Marco Antônio pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

A todos os meus amigos e colegas de curso que fizeram parte dessa tragetória.

"Ética é o que você faz quando está todo mundo olhando; o que você faz quando não tem ninguém por perto chama-se caráter".

Autor Desconhecido

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
IEEE	Institute of Electrical and Electronics Engineers
ISTQB	International Software Testing Qualification Board
BSTQB	Brazilian Software Testing Quality Board

1 Introdução

Neste capítulo são descritas as características básicas do trabalho, a fim de conceituá-lo quanto à apresentação do tema, o problema central que será abordado e as justificativas que motivam o real desenvolvimento da pesquisa. São também apresentadas as questões da pesquisa, que estão intimamente ligadas aos objetivos propostos. Por fim, é apresentada a metodologia de pesquisa que será adotada para a construção, organização e definição das demais etapas a serem executadas no decorrer do projeto.

1.1 Apresentação do Tema

Teste de Software trata-se de uma das subáreas de conhecimento associadas à Engenharia de Software. Ambas podem ser consideradas jovens se comparadas a outras áreas das engenharias (civil, elétricas ou mecânica), matemática, física, etc. O termo Engenharia de Software surgiu na década de 1960 e começou a ser comumente utilizado a partir dos anos de 1970. Já a área de teste começa a tomar forma a partir dos anos de 1980 com o desenvolvimento dos primeiros modelos e ferramentas de teste que formalizavam as técnicas de teste até ali desenvolvidas. Porém, o marco inicial em teste de software é considerado o lançamento do livro *The art of software testing*, de Glenford Myers, em 1979, sendo considerado um grande passo para a área.

É interessante notarmos, que muitos dos conceitos de teste já eram utilizados na prática, desde o primeiro desenvolvimento realizado, mesmo que não existisse uma formalização dos processos e métodos estudados. Assim, nesses mais de 30 anos, tivemos um grande amadurecimento da área, especialmente com a criação e formalização das técnicas e modelos, e o desenvolvimento de novas ferramentas para auxílio na preparação e na execução das atividades de teste. E hoje já possuí também instituições internacionais que atuam no país quanto à promoção de certificações na área, como é o caso do *International Software Testing Qualification Board (ISTQB)*, no Brasil possui o nome Brazilian Software Testing Quality Board (BSTQB).

Inicialmente, tendo em vista a definição e formalização inicial das métricas de teste, era necessário estabelecer um rigor e disciplina na execução dessas atividades, porém, cada vez mais identificamos uma exigência no mercado pela adaptabilidade e agilidade nos processos de desenvolvimento, visando garantir o atendimento das expectativas sobre o produto. Sendo que muitas vezes essas modificações são identificadas tardiamente ou sequer foram previstas.

A característica de adaptar-se rapidamente as mudanças é apontada como um dos maiores desafios do século XXI para as organizações (Boehm, 2008).

A necessidade de adaptação às características, em especial, de adaptabilidade às mudanças, curtos prazos de entrega e acréscimo de qualidade ao produto com baixos custos fez com que, nos últimos anos, os métodos de desenvolvimento de software conhecidos como métodos ágeis ganhassem mais popularidade e aceitação (Kongsli, 2006), como forma de resposta às mudanças no mercado (Borjesson e Mathissen, 2005). Podemos citar, por exemplo, dentre esses métodos: *Extreme Programming*, *Pragmatic Programming*, *dX*, *Crystal* e *Scrum*.

Aqui então, identificamos uma grande diferença de paradigmas entre as duas principais abordagens de desenvolvimento mais comuns, aonde podemos identificar um "embate" entre Métodos Tradicionais x Métodos ágeis.

É exatamente neste ponto que este trabalho irá se concentrar. Atualmente, a atividade de teste de software é considerada uma das que mais despende tempo, mas em contrapartida é também uma das atividades mais importantes para agregar qualidade ao software. E, portanto, é proposta uma ferramenta que possa apoiar a decisão de qual metodologia utilizar para acrescentar agilidade no processo de teste de software (seja o modelo de desenvolvimento de software utilizado tradicional ou ágil) como uma forma de promover uma otimização do processo de teste, reduzindo-se o tempo para a sua aplicação, e visando ganho de qualidade de processo e do produto.

1.2 Problema

Numa época na qual cada vez mais as pessoas buscam primar pela qualidade e velocidade, quaisquer que sejam os tipos de tarefas desenvolvidas, temos, em muitas dessas atividades

os softwares como uma parte integrante ou a própria informatização de uma atividade ou setor leva a um aumento da velocidade na execução dos processos.

Associada a esta, existe um grande apelo ou necessidade por parte do mercado de que as evoluções em software culminem em ganho de qualidade, assertividade das tarefas, aumento de agilidade do processo com baixo custo.

É exatamente neste ponto, que a área de teste de software pode e deve contribuir muito, adicionando uma garantia e fazendo com que um produto seja considerado mais confiável, o que consequentemente está ligado a um aumento de qualidade, tendo em vista que a realização de testes é um dos processos responsáveis por revelar a presença de defeitos (Myers, 2004).

É importante destacarmos também que toda a execução de um processo bem definido de teste de software não garante que um software não possui defeitos e nem que seja de qualidade, porém, aumenta e muito a confiança de que o mesmo atinja um bom nível, conforme destacado por Pádua (2000) em Engenharia de Software: fundamentos, métodos e padrões:

> "A qualidade, por outro lado, é consequência dos processos, das pessoas e da tecnologia."

E um dos processos citados por Pádua (2000) é exatamente o processo de teste. E que por muitas vezes é deixado como última atividade no desenvolvimento do software, e são executados às pressas, sem o devido planejamento, isso quando não são ignorados devido ao prazo apertado para entrega do produto ou ao orçamento já ter ultrapassado o limite.

É necessário que tanto o processo de construção de software como o processo de teste de software seja capaz de absorver da melhor maneira as mudanças não previstas previamente, apresentando-se de forma mais flexível. Neste caso a proposta é integrar características de agilidade ao processo, porém, como acrescentar agilidade no processo de teste? Quais características e práticas ágeis utilizar para agregar agilidade?

1.3 Justificativa

A proposta de trabalho apresentada nesta pesquisa se relaciona com itens pertinentes dentro da área de tecnologia da informação, em especial, por que teste de softwares é uma das áreas nas quais os processos impactam diretamente na validação do processo de desenvolvimento de softwares, e consequentemente na qualidade do produto final que será repassado ao cliente.

Por se tratar a própria área de teste de software de uma área considerada nova, para os padrões das áreas científicas, e também, de ser a área da informática muito dinâmica, estando em pleno crescimento e desenvolvimento, aonde possui mudanças todos os dias e existe sempre uma dificuldade de se acompanhar todas essas transformações. Este trabalho acadêmico propõe o uso de um sistema para apoio à decisão e realização das atividades de teste, buscando um aumento da agilidade na execução da tarefa de teste, o que pode gerar um ganho considerável para a área.

Além disso, a proposta tem abrangência para todas as empresas que possuem ou não um processo de teste de software, e caso possuam, seja ele considerado tradicional ou ágil, podendo diminuir seu tempo, custo e aumentar a assertividade.

Um dos desafios do trabalho é exatamente o fato de que a utilização de conceitos de agilidade em processos de teste de software é um ramo de pesquisa ainda em aberto e que não possui muitas aplicações práticas de grande relevância encontradas, sendo importante que a aplicação desenvolvida possa ser parametrizável ao ponto de absorver da melhor maneira possível novos métodos de teste que possam surgir com o passar do tempo.

1.4 Hipótese

A utilização de características e práticas ágeis no processo de teste de software contribuem de forma positiva no aumento da agilidade de execução do processo de teste e na diminuição de custos associados às manutenções, e no mínimo mantendo a mesma qualidade do produto final, porém, com possibilidade de acréscimo na qualidade do produto.

1.5 Objetivo

1.5.1 Objetivo Principal

Desenvolver uma ferramenta que possa apoiar a decisão de quais práticas são mais indicadas para serem utilizadas no processo de teste de software de acordo com as características de cada projeto para agregar maior agilidade no processo, menor custo e ao menos mantendo-se a qualidade do produto.

1.5.2 Objetivos Secundários

Além do item apresentado no objetivo principal, podemos destacar também que a pesquisa proposta é realizada em uma área que possui ainda muita possibilidade de ser explorada, e a respectiva pesquisa gera um aumento de conhecimento, difusão da informação e acréscimo de valor a área de pesquisa. Além disso, abre caminho para que outras pesquisas possam se aprofundar ainda mais na área tendo uma referência de um modelo a se seguir (em caso de confirmação do objetivo principal) ou que tenha uma referência de um modelo a ser aperfeiçoado (caso não obtenha totalmente a confirmação do objetivo principal).

1.6 Metodologia

Este tópico tem como objetivo apresentar os métodos que são utilizados na pesquisa para planejá-la, realizar as experiências e analisar os resultados, ou seja, caracterizar o projeto desta monografia a partir do seu estilo, natureza e procedimentos técnicos adotados.

1.6.1 Caracterização da Pesquisa

Conforme descrito em tópicos anteriores, este trabalho visa à construção de uma ferramenta prática para apoiar as decisões a cerca de quais métodos e/ou metodologias são mais indicadas para serem utilizadas no processo de teste de software com o intuito de agregar maior agilidade, e ao mesmo tempo, no mínimo, manter a mesma qualidade do produto dentro do processo de desenvolvimento de software. E por isso, do ponto de vista

da natureza do projeto, este é considerada uma pesquisa aplicada.

Tendo em vista a não existência de muitas referências diretamente ligadas ao tema a pesquisa têm um sentido exploratório quanto ao tema, tendo em vista a construção de uma nova ferramenta com o intuito de agregar valor ao processo.

Em resumo, podemos observar que a metodologia utilizada na pesquisa pode ser dividida em três tópicos principais: (1º) fundamentação do tema (Referencial Teórico), (2º) Implementação da ferramenta de apoio e (3º) Análise.

No primeiro tópico são realizados os estudos com o intuito de unificar o conhecimento, planejar, definir e estruturar a metodologia a ser desenvolvida para o apoio a decisão de quais métodos e práticas de teste de software devem ser utilizados para os projetos de software de acordo com as suas características e com as pretensões de obtenção de agilidade. Além disso, faz-se necessário realizar uma revisão bibliográfica sistemática para definição das metodologias e características de agilidade. Para este item, será utilizada como base a fundamentação realizada por Abrantes (2012) em sua tese de doutorado, Estudos Experimentais sobre Agilidade no Desenvolvimento de Software e sua útil Utilização no Processo de Teste. E esta servirá também de base para o desenvolvimento do software de apoio proposto.

No segundo tópico é realizado o desenvolvimento da metodologia estruturada a partir do primeiro item, contendo além do apoio a decisão da metodologia mais adequada, também a demonstração da forma de utilização da metodologia sugerida, facilitando a adequação e o uso no ambiente de trabalho. É importante destacarmos, que o conhecimento prévio dos conceitos e métodos ágeis por parte da equipe gera, sem a menor dúvida, uma maior facilidade de adequação e utilização das métricas selecionadas ao projeto de software desenvolvido.

Já o terceiro item, é referente à análise da ferramenta após a sua construção, visando identificar se os objetivos propostos, inicialmente, foram atendidos.

2 Referencial Teórico

Este tópico tem como objetivo apresentar os conceitos que servirão para um nivelamento de conhecimento a cerca da área de atuação da pesquisa, e que são necessários à composição de uma base de conhecimentos precursores aos assuntos que serão abordados com maior profundidade no decorrer do desenvolvimento do projeto. Aliado a estes, será apresentada uma contextualização a respeito do tema e será realizado o levantamento de trabalhos correlacionados a mesma proposta desta monografia.

2.1 Conceitos Básicos

2.1.1 Engenharia de Software

A Engenharia de Software é uma das grandes áreas da Computação. Ela teve o termo, Engenharia de Software, criado na década de 1960, e começou a ser comumente utilizado a partir dos anos de 1970. A IEEE Glossary (1990) define Engenharia de Software como:

> (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software; (2) O estudo de abordagens como definido em (1).

E que converge com as ideias apresentadas por Pressman (2010):

> [Engenharia de Software é] o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais.

Destacando-se que a importância é o que é produzido e como é produzido, ou seja, o produto que está sendo desenvolvido e o processo para a construção do mesmo. Assim, a visão sob o ponto de vista da redução do custo, acréscimo de qualidade e promoção da satisfação dos clientes com o produto demonstra uma relevância muito grande para área e é um dos motivos do seu crescimento.

2.1.2 Testes de Softwares

Com uma crescente demanda, especialmente relacionado à qualidade do produto, uma das áreas que ganhou bastante mercado é a área de teste de software. De acordo com Neto (2007) o teste tem por objetivo:

O seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final.

O processo de testar é analisar se o software está funcionando corretamente, verificar se está conforme especificado e se o sistema está sendo desenvolvido conforme a solicitação do usuário final (Moreira, 2003).

Para que possamos começar a trabalhar com testes de software, é importante definirmos alguns conceitos básicos referentes ao mesmo. Vamos começar diferenciando defeitos, erros e falhas. Elas estão definidas na terminologia padrão para engenharia de software do IEEE Glossary (1990 apud Neto, 2007):

> Defeito é um ato inconsistente cometido por um indivíduo ao tentar entender uma determinada informação, resolver um problema ou utilizar um método ou uma ferramenta. Por exemplo, uma instrução ou comando incorreto.
>
> Erro é uma manifestação concreta de um defeito num artefato de software. Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução de um programa constitui um erro.
>
> Falha é o comportamento operacional do software diferente do esperado pelo usuário. Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar uma falha.

Sendo os conceitos, também, destacados na Figura 2.1. Temos então que: os defeitos são aqueles causados pelas pessoas (mal uso de uma tecnologia) e que podem ocasionar erros, nas quais geram falhas quando o software está sendo executado pelo cliente.

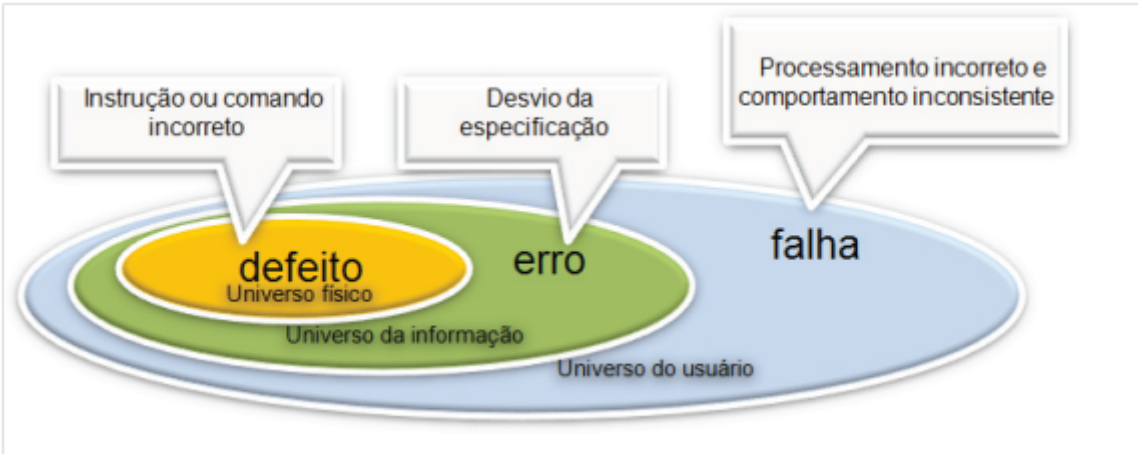

Figura 2.1: Exemplifica a relação entre os conceitos de Defeito x Erro x Falha em testede software

Segundo Bastos et al (2012, p. 31) o objetivo principal do processo de teste é simplesmente encontrar o maior número possível de defeitos no software. Mas é importante notarmos, que os testes podem mostrar a presença, mas não a ausência de defeitos (Dijkstra, 1979), tendo-se em mente que não existe a possibilidade de realizar testes para todos os diferentes casos possíveis.

Portanteo, é importante que o processo de teste seja realizado com qualidade e que seja mais ágil para que possa gerar cada vez menos problemas ao usuário ou processo associado.

2.1.3 Testes: Metodologias Ágeis x Metodologias Tradicionais

Cada vez mais as equipes de desenvolvimento focam em entregar produtos que agregam valor ao negócio e satisfaçam o cliente. E esta mudança de paradigma vem ocorrendo no decorrer de pouco mais de uma década, principalmente, por conta do surgimento e crescente utilização de boas Metodologias ou Práticas Ágeis, mas também pelo cliente que passa cada vez mais a ter força e conhecimento e que quer o melhor da tecnologia, o que for mais rápido, barato, com prazos menores, maior flexibilidade à mudanças e coloque mais qualidade no produto final.

De forma similar, as atividades de teste de software sofreram evoluções para se adequar a um processo mais dinâmico. Mas, na prática, não é fácil incorporar essas

metodologias ágeis ao processo de teste. Pois, por exemplo, as atividades já funcionam da mesma forma há décadas; existem fatores organizacionais que contribuem para dificultar (normalmente são equipes separadas, quando existe: Desenvolvimento e Testes).

Nas abordagens Tradicionais existe um processo formal e robusto na qual cada uma das etapas é bem determinada, normalmente sequenciadas e os testes são realizados ao final do processo. Para realização dos testes o responsável avalia a especificação para levantar todos os requisitos que devem ser atendidos e, normalmente, montar um documento para execução dos testes. Caso sejam encontrados problemas, o projeto retorna para correção e, após isso, nova avaliação integral a partir dos documentos. Fazendo com que erros encontrados comprometam consideravelmente, e comumente, os prazos estabelecidos.

Na perspectiva de abordagens ágeis, o teste de software é uma tarefa rotineira e ligada à todas as etapas do desenvolvimento do projeto. Independente da metodologia utilizada, um problema recorrente é a grande incidência de defeitos. E estes causam:

- **Lentidão no desenvolvimento**: cada novo problema gasta tempo para ser analisado e corrigido;
- **Efeitos colaterais**: frequentemente, a correção de um defeito acaba adicionando outros como efeito colateral. Isso, têm diversos fatores como origem;
- **Custos altos**: defeitos, se encontrados em produção tem um custo muito maior do que quando identificados durante o desenvolvimento e geram insatisfação dos clientes.

Os métodos ágeis entendem a ocorrência destes defeitos e visam mitigá-los através de testes, especialmente visando: constância, regressão e uma consequente automatização do processo, através da utilização de ferramentas para apoiar estas atividade.

As técnicas de testes pelas metodologias ágeis não se limitam apenas aos testes do código, mas também aos processos bem mantidos que minimizam a inclusão de defeitos no código, como: a programação em pares, técnicas como *Teste Driven Development (TDD)*, proximidade entre desenvolvedores e principais testadores e *feedbacks* constantes objetivando melhoria contínua.

Teste ágil é uma atividade desempenhada por todos os membros do time de forma contínua, automatizada (se possível), na qual cada membro irá realizar os testes sob uma perspectiva (cliente, desenvolvedor, testador), pois todos são responsáveis pela qualidade do produto. Os testes podem ser de diversos tipos, sob diversos aspectos e utilizar diversas práticas, por exemplo:

- Testes Unitários e Aceitação automatizados;
- Testes Exploratórios;
- *Teste Driven Development (TDD)*;
- *Acceptance Test-Driven Development (ATDD)*;
- *Refactoring*;
- Programação em par;
- Integração contínua;
- Design simplificado;

A implantação do teste ágil não é simples, pois enfrenta uma série de obstáculos (por exemplo: fatores organizacionais, falta de conhecimento, equipes separadas, resistência às mudanças), mas proporcionam melhorias ligadas à qualidade (processo e produto), tempo, agilidade e consequente satisfação do cliente final com o produto.

2.2 Contexto

Apesar de o processo de desenvolvimento de software existir e ser utilizado a muitos anos, ainda é comum identificarmos a ocorrência de problemas como perda de prazos, excesso de despesas e entrega de produtos com má qualidade em vários projetos de software de diferentes áreas de atuação. Além disso, cada vez mais o mercado demanda uma necessidade de acréscimo da qualidade, tendo em vista que cada vez mais o software está integrado com as atividades realizadas no dia a dia das pessoas, e a ocorrência constante

de falhas no software impacta diretamente no desempenho dessas atividades, tornando cada vez mais crítica essa necessidade.

Outro ponto, também relacionado ao anterior é o excesso dos gastos. Além de uma exigência por produtos de qualidade, também espera-se produtos mais baratos, e esses dois itens estão ligados não só no desenvolvimento do projeto, mas também nos processos de manutenção. Para termos real dimensão desse impacto é interessante observarmos que a correção de um defeito após a entrega do produto é frequentemente 100 vezes mais caro do que corrigi-lo durante as atividades de requisitos e projeto do sistema (Boehm e Basili, 2001).

As pessoas estão sempre correndo e a cada dia que passa tem menos tempo para fazer as mesmas atividades, e, além disso, quase todas as novas demandas que aparecem são "para ontem", assim, há uma crescente necessidade da área de desenvolvimento otimizar cada vez mais o processo de execução das suas tarefas a fim de atender a essa necessidade do mercado.

Agregando os três pontos discutidos acima com o contexto da pesquisa, podemos identificar, claramente, a inter-relação destes com as atividades de teste de software. E verificar que estas atividades estão diretamente ligadas a possibilidade de promover um aumento da confiabilidade e qualidade do produto, e principalmente, com o acréscimo dos conceitos de agilidade possibilitar um aumento na velocidade da execução das suas atividades, o que gera uma diminuição na duração do projeto e pode consequentemente reduzir o custo total.

2.3 Revisão Sistemática da Literatura

2.3.1 Objetivos

A revisão sistemática da literatura tem por objetivo embasar a pesquisa realizada quanto a pertinência de informações, conceitos e trazer um maior valor científico que uma simples pesquisa *ad hoc*.

2.3.2 A Revisão

As bases de dados consideradas para a revisão sistemática são todas eletrônicas, disponíveis no portal da CAPES e foram consideradas significativas (contendo publicações pertinentes), com acesso a recuperar os textos e referências. Assim, foram consideradas:

- ACM Digital Library
- IeeeXplorer
- Science Direct
- Scopus
- BDBComp

A linguagem base utilizada para a pesquisa é o inglês, por se tratar da maioria nas bases de dados pesquisadas e do tema da pesquisa.

As palavras-chave foram escolhidas e separadas em tópicos por contexto:

- Software, framework, plugin, tool, development, system, application, implementation;
- method, approach, technique, process, practice, methodology, characteristic, attribute, property, feature, aspect;
- Testing software;
- Agile, agility;
- Observação: Types of testing, types of software testing são opcionais e pode ser necessária pesquisa separada para associar os tipos de teste às características e/ou práticas;

Algumas fontes possuem limitação quanto a utilização de muitas palavras-chaves e nomenclatura para pesquisa, assim, a string de busca básica deve variar de acordo com a fonte, mas sempre mantendo a estrutura padrão a partir das palavras acima.

Após a execução das buscas, os resultados obtidos serão avaliados para inclusão e exclusão, a partir de:

- Os documentos devem estar disponíveis na web;
- Os documentos devem contemplar propriedades, características, práticas e/ou ferramentas de agilidade em desenvolvimento de software, especialmente na etapa de teste de software;
- Os documentos devem relatar algum exemplo ou caso de utilização de propriedades, características, práticas e/ou ferramentas no processo de desenvolvimento de software, especialmente na etapa de teste de software;

Os resultados retornados da execução da pesquisa serão avaliados em duas etapas:

- Serão avaliados como válidos a partir da análise do título e do resumo disponível;
- Em um segundo momento os textos válidos serão reavaliados quanto a pertinência para a pesquisa a partir da leitura crítica e caso seja considerado pertinente serão armazenadas as informações do mesmo e incluídos nas referências

2.3.3 Execução da Revisão

A revisão foi realizada conforme descrito anteriormente e foram obtidos e avaliados os trabalhos retornados.

Uma grande parte dos trabalhos encontrados não aborda de forma profunda a proposta desta monografia, mas servem de base para termos outras visões a cerca do mesmo tema, sendo notório que os trabalhos possuem um maior foco na definição de qual abordagem deve ser selecionada para utilização no processo de desenvolvimento do software como um todo.

Uma visão muito interessante e pouco identificada nos trabalhos encontrados é o foco em alguma das atividades específicas dentro do processo de desenvolvimento. Assim, podemos destacar alguns trabalhos aonde o objetivo se aproxima da proposta deste:

- A tese de doutorado de Abrantes (2012), denominada, Estudos experimentais sobre agilidade no desenvolvimento de software e sua útil utilização no processo de teste. A partir do levantamento de características de agilidade e seus relacionamentos com as práticas ágeis e as Etapas de Teste é proposta uma abordagem para estimar

a adequação de novos projetos as abordagens ágeis (a partir das características básicas do projeto, como: Duração Estimada do Projeto, Indicador de Complexidade do Problema, Indicador de Instabilidade dos Requisitos, Tamanho da Equipe do Projeto e Criticalidade do Projeto) e permite ao responsável selecionar quais são as características que ele deseja incluir ao projeto e a partir destas são indicadas quais as práticas mais relevantes para serem utilizadas em cada etapa do teste de software. Após a definição das práticas, as mesmas seriam utilizadas pelos responsáveis e posteriormente sendo preenchidas as avaliações sob cada uma das práticas utilizadas. Sendo, então, adotada como pesquisa base para o desenvolvimento da ferramenta proposta por esta monografia e melhor detalhada posteriormente;

- O trabalho de MNKANDLA (2008), denominado, *A Selection Framework For Agile Methodology Practices: A Family of Methodologies Approach.* A pesquisa visa em desenvolver uma estrutura ou ferramenta para selecionar o conjunto mais apropriado de práticas ágeis para um projeto de software a partir de suas características (por exemplo: tamanho do projeto, prazo para desenvolvimento, complexidade, criticidade e prioridades do cliente) ao invés de buscar selecionar uma metodologia ágil específica para o mesmo projeto (métrica comumente utilizada), e buscando adequar as práticas (realizando o levantamento de diversas técnicas ágeis utilizadas) para que a utilização das mesmas sejam viáveis. O projeto possuí foco no contexto sul-africano;

- A pesquisa de QUMER e HENDERSON-SELLERS (2008), denominado, *A framework to support the evaluation, adoption and improvement of agile methods in practice.* O projeto leva em consideração a dificuldade inical de conseguir instalar verdadeiramente abordagens ágeis, assim, possuí uma abordagem teórica e prática a partir de estudos de casos para começar a utilização de abordagens ágeis, mostrando as dificuldades e utilizando Agile Adoption e Improvement Model (AAIM). Assim, consegue apoiar na decisão entre utilização de alguma metodologia já existente e/ou possibilidade de mesclagem de conceitos de algumas metodologias que sejam aplicáveis ao projeto. É utilizada uma análise sobre 4 dimensões para apoiar as escolhas das organizações;

- O livro de BOEHM e TURNER (2004), denominado, *Balancing agility and discipline: A guide for the perplexed*. A obra discute a respeito da agilidade e disciplina e papéis de cada um nas abordagens ágeis. Vai realizar uma apresentação e nivelamento quanto aos métodos tradicionais e principalmente métodos ágeis (apresenta diversos, por exemplo: *Scrum*, *Adaptive Software Development (ASD)*, *Lean Development (LD)*, *Crystal*, *eXtreme Programming (XP)*, etc). Questiona quanto a possibilidade de balancear tradicionalismos e agilidade (quais são os riscos envolvidos para cada escolha), nos lembrando também que nem só por que uma metodologia é ágil que ela irá trazer ganho, pois a mesma pode não atender às necessidades de um processo e/ou empresa em específico.

2.3.4 Apresentação da Estratégia Adotada

Conforme citado anteriormente, a tese de doutorado de Abrantes (2012) está sendo utilizada como base para o desenvolvimento da ferramenta proposta. Assim, será apresentada uma visão geral da estratégia adotada no projeto (detalhada no próximo capítulo).

Podemos dividir a mesma em 3 fases:

- **Fase 1**: Carga inicial/atualização da base de conhecimento;
- **Fase 2**: Seleção e planejamento das características e práticas ágeis para um novo projeto;
- **Fase 3**: Avaliação dos resultados alcançados;

Fase 1: Carga inicial/atualização da base de conhecimento

Esta fase possui apenas um passo que trata-se da carga inicial da base de conhecimento com as informações que serão utilizadas durante o planejamento de um projeto. A ideia é que estas informações tenham uma baixa necessidade de alteração, apenas em casos de inclusão de novos itens e/ou adaptações aos processos da área.

A Figura 2.2 trás uma ideia das atividades associadas a estas cargas, destacando: Características de Agilidade, Práticas Ágeis e seus relacionamentos.

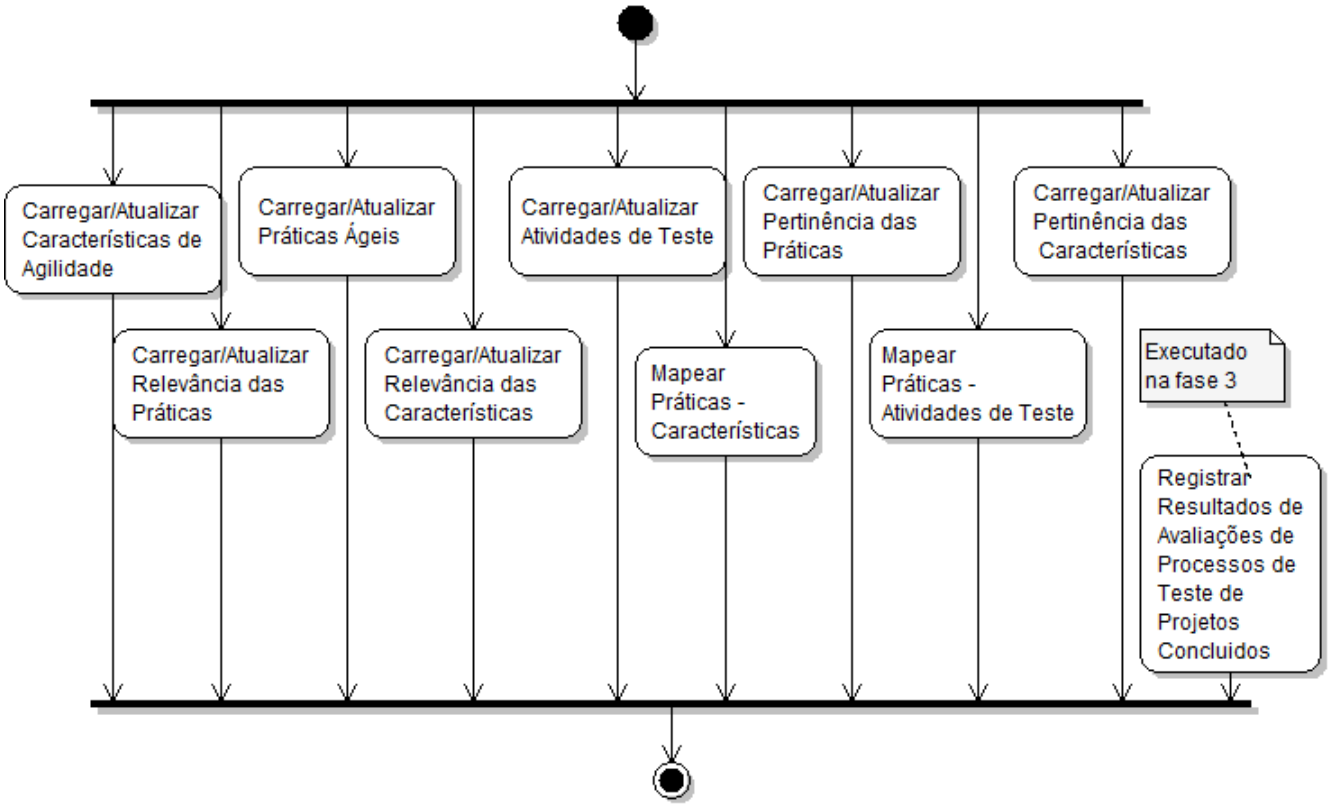

Figura 2.2: Carga/atualização da base de conhecimento sobre características e práticas ágeis

Fase 2: Seleção e planejamento das características e práticas ágeis para um novo projeto

Esta fase possui uma série de passos (listados abaixo) e é responsável pelo preenchimento de um projeto e planejamento das práticas ágeis que devem ser utilizadas.

Podemos destacar as atividades:

- Registro das características básicas dos projetos;
- Decisão da continuidade ou não a partir da adequação do projeto às abordagens ágeis (projetado a partir da atividade anterior);
- Seleção das características de agilidade e obtenção das práticas ágeis associadas às características;
- Relacionamento entre práticas ágeis e etapas de teste de software;

A Figura 2.3 trás uma ideia das atividades associadas a esta fase.

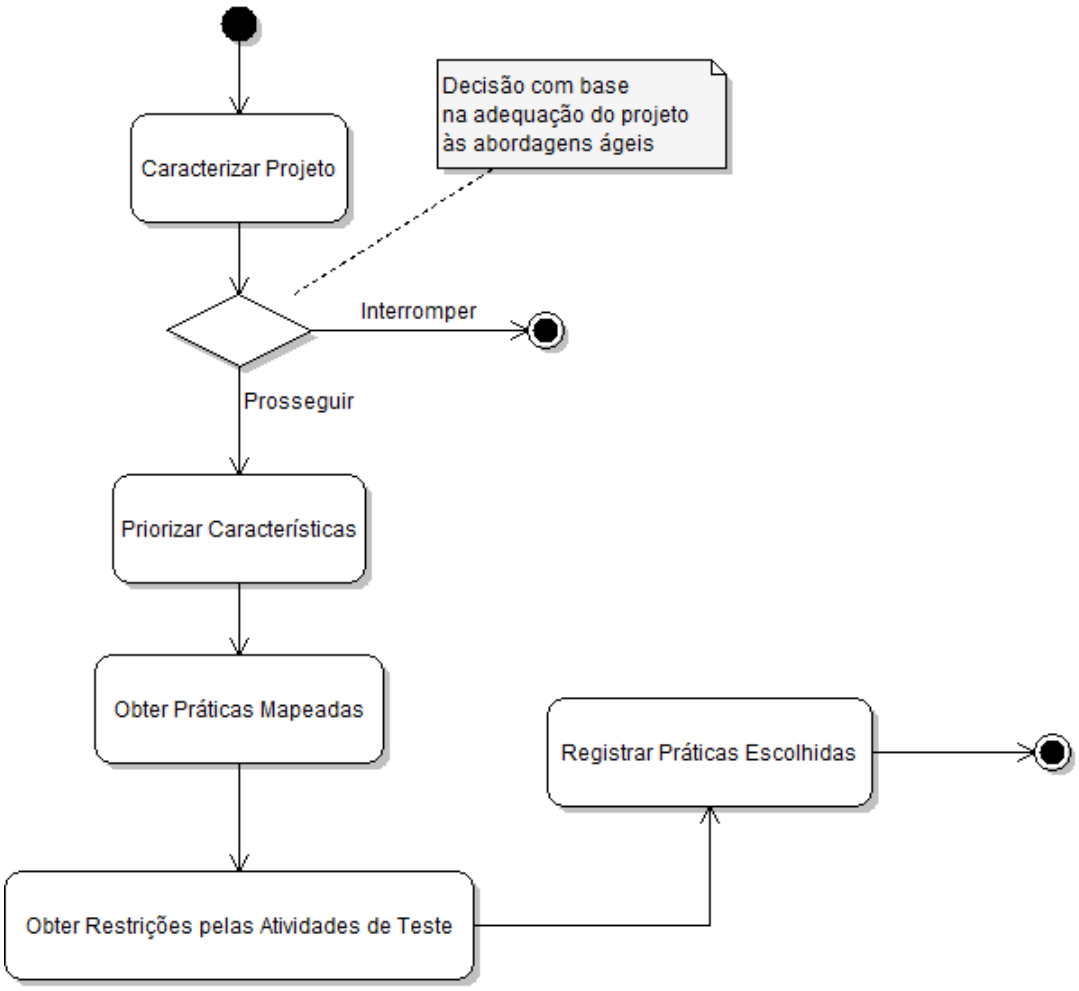

Figura 2.3: Seleção e planejamento das características e práticas ágeis

Fase 3: Avaliação dos resultados alcançados

Esta fase constitui-se na avaliação dos resultados observados após a execução das práticas selecionadas na fase anterior.

Para isso, devem ser respondidas as seguintes questões:

1. Para cada uma das práticas selecionadas na etapa anterior:
   - Qual o grau em que a prática foi adotada e seguida pela equipe?
   - Qual o nível de contribuição para o sucesso das atividades de teste?
   - Qual o grau de avaliação da prática pela equipe (em %)?
   - Observação;

2. Para o projeto como um todo:

   - Como você avalia o resultado das práticas para o processo de testes?
   - Observação;

   A Figura 2.4 trás uma ideia das atividades associadas a esta fase.

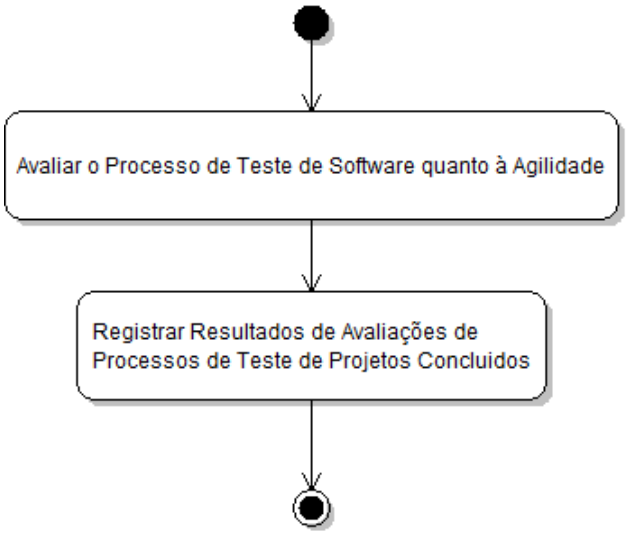

Figura 2.4: Avaliação dos resultados alcançados

2.3.5 Conclusões

A partir da execução da revisão da liberatura e, principalmente, da leitura dos documentos recuperados, podemos inferir que além do objetivo citado anteriormente, a proposta de acrescentar agilidade dentro das etapas do processo de software é uma possibilidade interessante e que já está sendo estudada e possui trabalhos publicados a respeito. Assim, a proposta deste projeto pode dar continuidade aos trabalhos de pesquisa na área.

3 Solução Proposta

Neste capítulo será apresentado o detalhamento da solução proposta para construção do software para apoio à etapa de teste de software.

3.1 Detalhamento da Solução Proposta

A solução proposta será subdividida nas seguintes parte:

- Cadastros Básicos;
- Inclusão/Edição de novos projetos:
- Avaliação de Eficácia;
- Documentação Técnica;
- Detalhamento das Telas do Sistema;

3.2 Cadastros Básicos

Este tópico objetiva detalhar os cadastros necessários dentro do software. Como já foi destacado anteriormente, o processo de desenvolvimento se modifica constantemente, com o surgimento e remodelação da forma de se trabalhar, assim, os cadastros foram preparados para permitir a inclusão de novas necessidades e descobertas, a partir do uso e vivência dos profissionais que estiverem utilizando a ferramenta.

A partir da revisão bibliográfica previamente realizada, será feita carga inicial dos itens considerados pertinentes em cada uma das etapas.

3.2.1 Cadastro dos Modelos de Ciclo de Vida de Projetos

Ciclos de Vida de um projeto, são o conjunto de etapas, atividades, processos ou fases que um projeto vai passar desde o levantamento dos requisitos até a implantação (Macêdo e

Spínola, 2011) (levaremos em conta a construção de novos projetos e por isso a etapa fim sendo considerada implantação).

Assim, este cadastro é responsável por permitir identificar as etapas pelas quais o projeto está previsto para passar no decorrer do mesmo e trata-se de um dos itens primordiais para auxiliar na definição de quais metodologias de teste são mais adequadas.

Será incluído campo para descrição resumida, porém é considerada a existência de conhecimento prévio dos usuários do sistema.

Este cadastro deve possuir os seguintes campos:

(A) Nome do Ciclo de Vida

(B) Descrição Resumida

3.2.2 Cadastro de Características de Agilidade

Trata-se do conjunto de Características de Agilidade mapeadas em processos de desenvolvimento de software através da utilização de revisão sistemática e pesquisa de opinião com profissionais da área, o estudo realizado por Abrantes (2012) foi considerado pertinente e será utilizado como carga inicial e testes no software desenvolvido.

A lista das Características de Agilidade pode ser consultada na seção Lista de Características de Agilidade presente no capítulo Apêndice.

Este cadastro deve possuir os seguintes campos:

(A) Nome da Característica de Agilidade

(B) Descrição Resumida

(C) Pertinência

(D) Relevância

Os campos de Pertinência e Relevância podem ser melhor entendidos a partir da leitura da seção Identificando a Pertinência das Características de Agilidade e das Práticas Ágeis presente no capítulo Apêndice.

3.2.3 Cadastro de Práticas Ágeis de Software Adotadas

Da mesma maneira que as Características de Agilidade citadas no item anterior, tratam-se do conjunto de Práticas Ágeis de Software mapeadas em processos de desenvolvimento de software através da utilização de revisão sistemática e pesquisa de opinião com profissionais da área, o estudo realizado por Abrantes (2012) foi considerado pertinente e será utilizado como carga inicial e testes no software desenvolvido.

A lista das Práticas Ágeis pode ser consultada na seção Lista de Práticas Ágeis presente no capítulo Apêndice.

Este cadastro deve possuir os seguintes campos:

(A) Nome da Práticas Ágeis

(B) Descrição Resumida

(C) Pertinência

(D) Relevância

(E) Características Associadas

Os campos de Pertinência e Relevância podem ser melhor entendidos a partir da leitura da seção Identificando a Pertinência das Características de Agilidade e das Práticas Ágeis presente no capítulo Apêndice.

3.2.4 Cadastro de Etapas de Teste de Sotware

Assim como as Características de Agilidade e Práticas Ágeis citadas nos itens anteriores, tratam-se das Etapas Básicas de Teste de Software mapeadas em processos de desenvolvimento de software através da utilização de revisão sistemática e pesquisa de opinião com profissionais da área, o estudo realizado por Abrantes (2012) foi considerado pertinente e será utilizado como carga inicial e testes no software desenvolvido.

A lista das etapas de Teste de Software pode ser consultada na seção Lista das Etapas de Teste de Software presente no capítulo Apêndice.

Este cadastro deve possuir os seguintes campos:

(A) Nome da Etapa de Teste

(B) Descrição Resumida

(C) Produtos de Trabalho ou Artefatos Produzidos

(D) Práticas Associadas

3.2.5 Cadastro de Indicador de Complexidade do Problema

Como o próprio nome apresenta, trata-se de um indicador para ponderamos a complexidade do problema para o projeto em destaque, quanto aos cálculos e regras de negócio. Assim, quanto menos complexo o processamento envolvido (valores mais altos), mais adequada é a aplicação de uma abordagem ágil ao projeto.

Logo, este cadastro deve possuir os seguintes campos:

(A) Nome do Indicador de Complexidade;

(B) Peso Associado;

As possibilidades de classificação serão preenchidas, inicialmente, da seguinte maneira (lista utilizada a partir de Abrantes (2012)):

(A) 4- Muito Baixa, para projetos que não envolvem cálculos complexos nem regras de negócio complicadas;

(B) 3- Baixa, para projetos que envolvem cálculos pouco complexos ou regras de negócio um pouco complicadas;

(C) 2- Média, para projetos que envolvem cálculos complexos ou regras de negócio complicadas;

(D) 1- Alta, para projetos que envolvem cálculos significativamente complexos ou regras de negócio significativamente complicadas;

(E) 0 - Muito Alta, para projetos que envolvem cálculos muito complexos ou regras de negócio muito complicadas;

3.2.6 Cadastro de Indicador de Instabilidade dos Requisitos

De forma semelhante ao item anterior, trata-se de um indicador para apontar o grau de instabilidade dos requisitos do projeto. Assim, quanto mais instável, ou seja, mais propenso a mudanças constantes, maiores os valores associados a estes e mais adequada é a aplicação de uma abordagem ágil para o projeto.

Logo, este cadastro deve possuir os seguintes campos:

(A) Nome do Indicador de Instabilidade;

(B) Peso Associado;

As possibilidades de classificação serão preenchidas, inicialmente, da seguinte maneira (lista utilizada a partir de Abrantes (2012)):

(A) 4- Muito Alta, para requisitos muito instáveis, com risco muito alto de afetar o escopo do projeto;

(B) 3- Alta, requisitos instáveis com razoável risco de afetar o escopo do projeto;

(C) 2- Média, para requisitos com grau médio de estabilidade, com algum grau de risco de afetar o escopo do projeto;

(D) 1- Baixa, requisitos praticamente estáveis com baixo risco de afetar o escopo do projeto;

(E) 0- Muito Baixa, requisitos estáveis, praticamente sem risco de afetar o escopo do projeto;

3.2.7 Cadastro de Faixa do Tamanho da equipe

Trata-se da quantidade de pessoas que estão envolvidas no projeto. Logo, quanto menor a quantidade de pessoas, mais indicada é a abordagem ágil para o projeto.

Logo, este cadastro deve possuir os seguintes campos:

(A) Tamanho da Equipe;

(B) Peso Associado;

As possibilidades de classificação serão preenchidas, inicialmente, da seguinte maneira (lista utilizada a partir de Abrantes (2012)):

(A) 4- até 6 pessoas;

(B) 3- de 7 a 10 pessoas;

(C) 2- de 11 a 39 pessoas;

(D) 1- de 40 a 80 pessoas;

(E) 0- mais de 80 pessoas;

3.2.8 Cadastro de Faixa de Duração do Projeto

Refere-se a duração estimada do projeto, em meses. Desta forma, menor o prazo, mais significativa é a abordagem ágil para o projeto.

Logo, este cadastro deve possuir os seguintes campos:

(A) Duração do Projeto (em meses);

(B) Peso Associado;

As possibilidades de classificação serão preenchidas, inicialmente, da seguinte maneira (lista utilizada a partir de Abrantes (2012)):

(A) 4- até 2 meses;

(B) 3- de 3 a 6 meses;

(C) 2- de 7 meses a 1 ano (12 meses);

(D) 1- de 13 meses a 2 anos (24 meses);

(E) 0- mais que 2 anos (acima de 24 meses);

3.2.9 Cadastro de Criticidade do Projeto

Este item refere-se a criticidade do projeto, quanto aos riscos que falhas no mesmo possam ocasionar. Quanto menos graves os riscos (valores mais altos), mais adequada é a abordagem ágil para ser aplicada no projeto. Os valores atribuídos a este indicador pesam no sentido de contra-indicar a abordagem ágil para o projeto e devem ser ajustados à medida em que a experiência com o guia for avançando.

Logo, este cadastro deve possuir os seguintes campos:

(A) Criticidade do Projeto;

(B) Peso Associado;

As possibilidades de classificação serão preenchidas, inicialmente, da seguinte maneira (lista utilizada a partir de Abrantes (2012)):

(A) 0- Muito Baixa, falhas do software podem causar desconfortos como atrasos na entrega de algum produto ou serviço ou longos tempos de espera;

(B) (-1)- Baixa, falhas do software podem resultar em perdas de recursos que podem ser recuperados sem maiores problemas;

(C) (-4)- Média, falhas do software podem resultar em perdas de recursos que podem ser recuperados, mas com dificuldade razoável;

(D) (-5)- Alta, falhas do software podem resultar em perdas de recursos que não podem mais ser recuperados;

(E) (-8)- Muito Alta, falhas do software podem resultar em grave ameaça a lesões corporais ou a perda de vida(s);

3.2.10 Cadastro de Plataforma de Execução

Responsável por informar em qual ambiente o software irá executar. Podendo ser complementada a medida que forem desenvolvidos softwares em outras plataformas.

Logo, este cadastro deve possuir o seguinte campo:

(A) Nome da Plataforma;

As possibilidades de classificação podem ser:

(A) Mobile;

(B) Sistema Embarcado;

(C) Desktop;

(D) Web;

3.2.11 Cadastro de Domínio da Aplicação

Trata-se da natureza da informação que é origem do software que está sendo desenvolvido. Assim, este item deve ser complementado a medida que o sistema for utilizado em projetos de outros domínios de aplicação.

Logo, este cadastro deve possuir os seguintes campos:

(A) Nome do Domínio;

As possibilidades de classificação foram determinadas como sendo as macro áreas de atendimento dentro de uma empresa, por exemplo:

(A) Tecnologia da Informação;

(B) Comercial;

(C) Financeiro;

(D) Jurídico;

(E) Tributário;

(F) Conveniência;

(G) Vendas;

(H) Logística;

3.3 Inclusão/Edição de novos projetos

Este tópico objetiva detalhar os passos necessários para inclusão e/ou edição dos projetos cadastrados no sistema, listando os requisitos obrigatórios em cada uma das etapas.

3.3.1 Dados Básicos do Projeto

Trata-se do conjunto de informações básicas a cerca do projeto que está sendo incluído e que servirão para contextualizar e também para identificar o grau de adequação do projeto com abordagens ágeis. A partir desta informação a equipe de testes pode decidir se deseja prosseguir e partir para a etapa de escolha das características desejadas para o processo de teste.

Essa etapa deve conter os seguintes campos:

(A) Nome do Projeto;

(B) Descrição Resumida do Projeto;

(C) Status do projeto: sendo exibido automaticamente pelo sistema a medida que as etapas vão sendo concluídas e pode assumir:

  (a) Inicidado

  (b) Concluído

  (c) Avaliado

(D) Data de Início do Projeto;

(E) Data de Fim do Projeto;

(F) Nome do Responsável do Cliente;

(G) Nome do Responsável da Equipe de Desenvolvimento;

(H) Data de Planejamento da Agilidade no Projeto;

(I) Data da Avaliação da Agilidade no Projeto: sendo exibida automaticamente pelo sistema a medida que as etapas vão sendo concluídas;

(J) Resultado da Avaliaçãoo de Agilidade: sendo exibida automaticamente pelo sistema a medida que as etapas vão sendo concluídas e pode assumir:

  (a) Não Avaliado;

  (b) Sucesso;

  (c) Fracasso;

  (d) Não Definido;

(K) Modelo de Ciclo de Vida: informação originada a partir do respectivo cadastro básico;

(L) Duração Estimada do Projeto: informação originada a partir do respectivo cadastro básico;

(M) Indicador de Complexidade do Problema: informação originada a partir do respectivo cadastro básico;

(N) Indicador de Estabilidade dos Requisitos: informação originada a partir do respectivo cadastro básico;

(O) Tamanho da Equipe de Projeto: informação originada a partir do respectivo cadastro básico;

(P) Criticidade do Projeto: informação originada a partir do respectivo cadastro básico;

(Q) Ambiente Físico: podendo assumir:

  (a) Distribuído;

  (b) Localizado;

(R) Plataforma de Execução: informação originada a partir do respectivo cadastro básico;

(S) Domínio da Aplicação: informação originada a partir do respectivo cadastro básico;

Essa etapa também deve conter os seguintes botões de ação:

1. Voltar: não atualiza as informações e retorna a etapa imediatamente anterior anterior;

2. Cancelar: não atualiza as informações e retorna a página inicial;

3. Salvar e Continuar: atualiza as informações do projeto e direciona para a próxima;

3.3.2 Adequação do Projeto às abordagens Ágeis

Após o preenchimento das informações gerais do projeto, será calculada e apresentado ao usuário a adequação do mesmo às abordagens ágeis levando em consideração apenas os seguintes itens:

1. Duração Estimada do Projeto;

2. Indicador de Complexidade do Problema;

3. Indicador de Estabilidade dos Requisitos;

4. Tamanho da Equipe do Projeto;

5. Criticidade do Projeto;

O cálculo utilizará os pesos associados a cada um dos itens listados acima da maneira descrita abaixo e será sugerido ao usuário continuar, caso o retorno seja acima de 50%, porém o mesmo tem autonomia para decidir:

$GrauAdequacao = ROUND((\frac{\sum P(IReferenciado)}{\sum P(IMaximo)}) * 100, 2)$, onde:

- P(IMaximo) é o maior peso possível a ser atribuído a a cada item (I) considerado;
- P(IReferenciado) é peso de cada item (I) escolhido na etapa anterior;

Id	Parâmetro	Valor Informado	Valor Máximo	Valor Atribuido
2	Duração Estimada do Projeto	5 meses	4	3
3	Indicador de Complexidade do Problema	média	4	2
4	Indicador de Estabilidade dos Requisitos	média	4	2
5	Tamanho da Equipe do Projeto	7	4	3
6	Criticalidade do Projeto	baixa	0	-1
	TOTALIZADORES		**16**	**9**
	Grau de adequação do projeto com as abordagens ágeis			56,3%

Figura 3.1: Exemplo: Grau de Adequação do Projeto com Abordagens Ágeis

3.3.3 Escolha das Características de Agilidade e Exibição das Práticas Associadas

Caso o usuário opte por continuar, ele terá a opção de escolher quais as características de agilidade que deseja inserir no processo de teste de software do projeto dentre as parametrizadas e o sistema exibe também as práticas ágeis associadas a cada característica selecionada e é calculado o grau de agilidade de cada prática levando em consideração a soma dos pesos (pertinência) das características de agilidade escolhidas e associadas a esta prática, multiplicada pelo peso da própria prática e este produto é dividido pela quantidade de características selecionadas.

As práticas serão exibidas pelo grau de agilidade estimado do maior para o menor.

Após isso, o usuário irá avançar mais uma etapa.

3.3.4 Práticas Ágeis x Etapas de Teste de Software

A partir do mapeamento entre características de agilidade (selecionadas anteriormente) e as práticas ágeis associadas são exibidas as etapas de teste associadas as práticas e caso alguma prática não esteja relacionada a nenhuma etapa a mesma será desconsiderada.

Assim, o usuário pode nortear-se quanto aos itens e práticas que deve utilizar em cada etapa do processo de teste do software.

3.4 Avaliação de Eficácia

Na etapa anterior, o sistema irá exibir o relacionamento entre as Práticas Ágeis e as Etapas de Teste de Software relacionadas. Após isto, a equipe de teste terá concluído a etapa de planejamento dos testes, sendo então necessário aplicar as práticas durante o processo de execução das etapas de teste, e após este, o responsável pela equipe de teste deve avaliar os resultados alcançados para o processo de teste quanto a agilidade.

Para isto, o mesmo irá avaliar cada prática utilizada, respondendo as seguintes questões:

- Qual o grau em que a prática foi adotada e seguida pela equipe? Podendo assumir:
  - 0 - BAIXO;
  - 1 - MÉDIO;
  - 2 - ALTO;
- Qual o nível de contribuição para o sucesso das atividades de teste? Podendo assumir:
  - 0 - BAIXO;
  - 1 - MÉDIO;
  - 2 - ALTO;
- Qual o grau de avaliação da prática pela equipe (em %)?;
- Observação;

  E, mais uma pergunta direcionada agora para todo o projeto:

- Como você avalia o resultado das práticas para o processo de testes? Podendo assumir:
  - 0 - MUITO RUIM;
  - 1 - RUIM;
  - 2 - REGULAR;

  - 3 - BOM;
  - 4 - MUITO BOM;

- Observações;

  E, após o preenchimento desta etapa o projeto será encerrado.

3.5 Documentação Técnica

Neste item será apresentado o detalhamento mais técnico da solução proposta no capítulo, sendo informadas questões como: ambiente e ferramentas utilizadas, modelos de banco de Dados e telas do sistema.

3.5.1 Ambiente de Desenvolvimento

Este programa foi desenvolvido na linguagem Java, sendo considerado uma Aplicação Web. E para implementá-lo foi utilizado o ambiente de desenvolvimento NetBeans IDE 8.0.2, os serviços Apache-Tomcat e as informações foram armazenadas em banco de dados MySQL.

3.5.2 Metodologia

A aplicação utiliza o Padrão Arquitetural MVC (Model View Controller) para organização e separação das execuções das tarefas. Atrelado a estes são utilizados também outros recursos, tais como:

- JSP (JavaServer Pages) + JSTL (JavaServer Pages Standard Template Library);
- Padrão Singleton;
- Padrão DAO (Data Access Object) para desacoplar e para manipulação das informações do banco de dados;

3.5.3 Modelo de Banco de Dados

A Figura 3.2 descreve o modelo Entidade Relacionamento da aplicação:

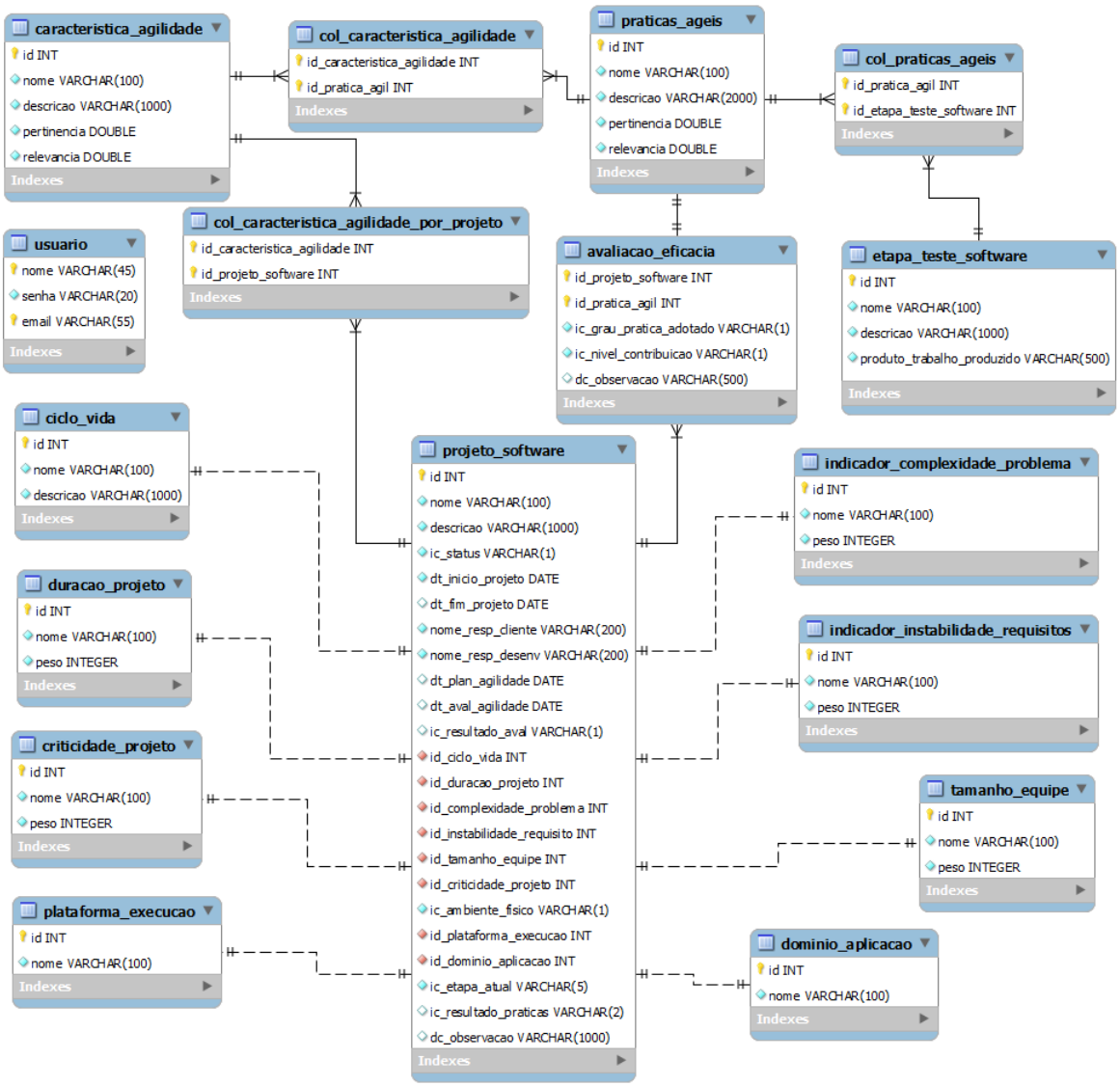

Figura 3.2: Modelo de Banco de Dados do Sistema

3.6 Detalhamento das Telas do Sistema

A solução proposta será subdividida nas seguintes parte:

- Login;
- Cadastros Básicos (Inclusão/Edição);
- Inclusão/Edição de novos projetos;
- Avaliação de Eficácia;

3.6.1 Login

Ao acessar a página inicial o usuário pode:

- Realizar o Login, opção de Esqueci minha Senha e acesso a informações do projeto;

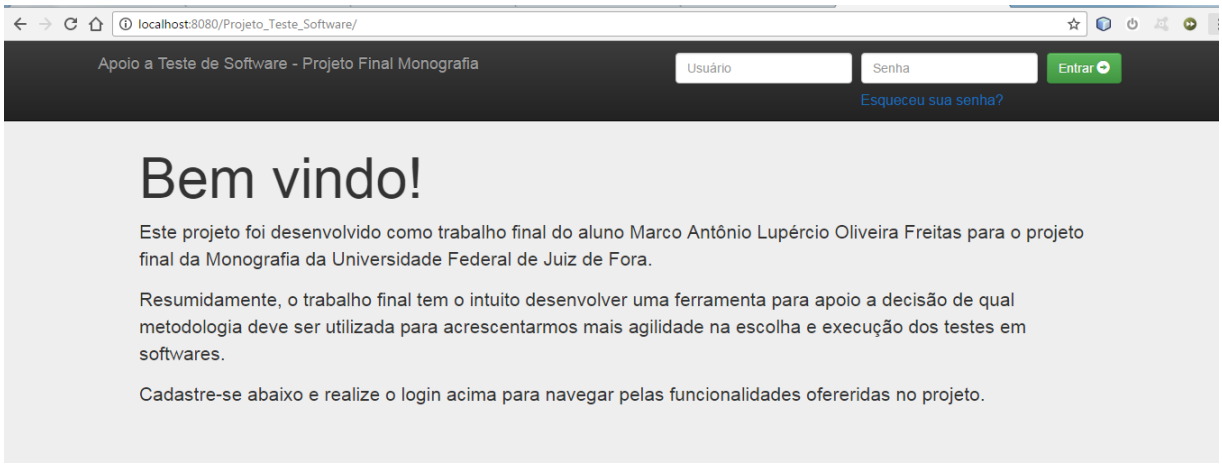

Figura 3.3: Tela referência do Login do Sistema

- Cadastrar um novo usuário (primeiro acesso);

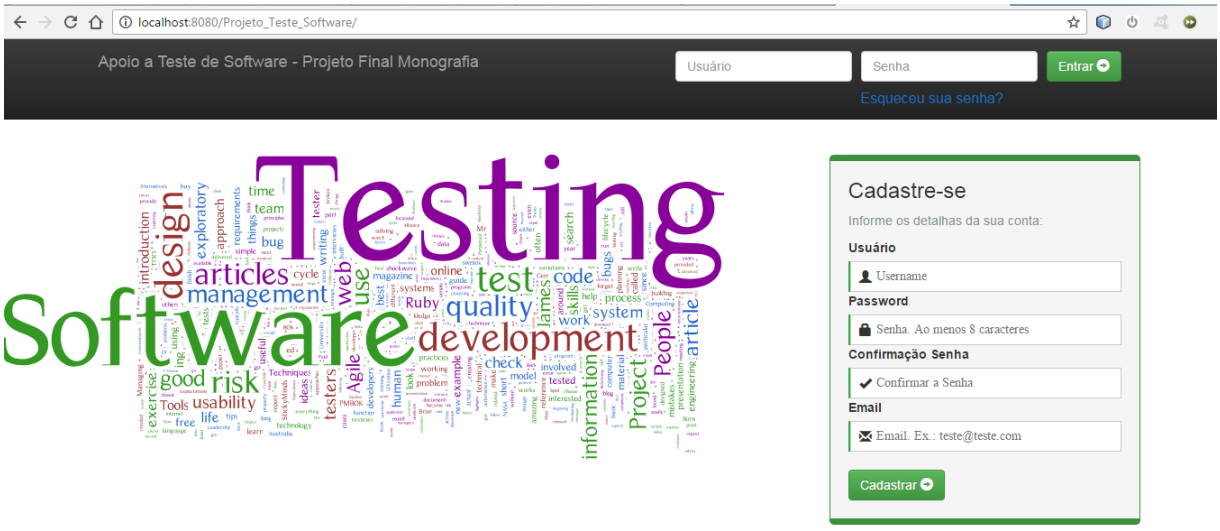

Figura 3.4: Cadastrar Novo usuário no Sistema

- Acesso a informações da UFJF/Curso de Ciência da Computação;

Figura 3.5: Informações do Curso

Após realizar o Login no sistema o usuário tem acesso às funcionalidades:

- Descrição resumida de funcionalidades do sistema;

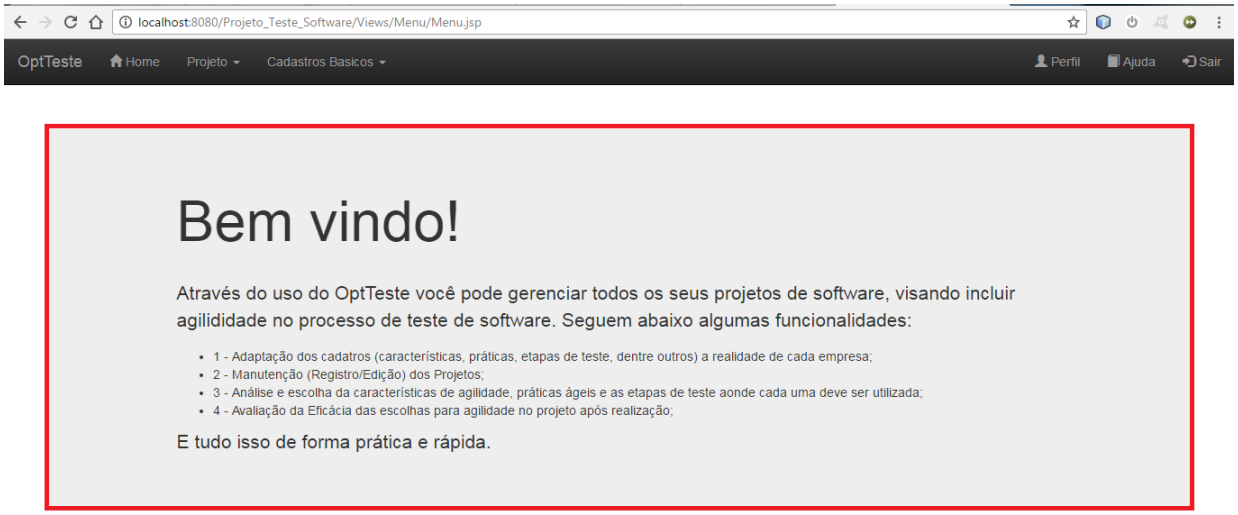

Figura 3.6: Informações básicas das funcionalidades do sitema

- Manter Perfil Cadastrado, Ajuda e Sair;

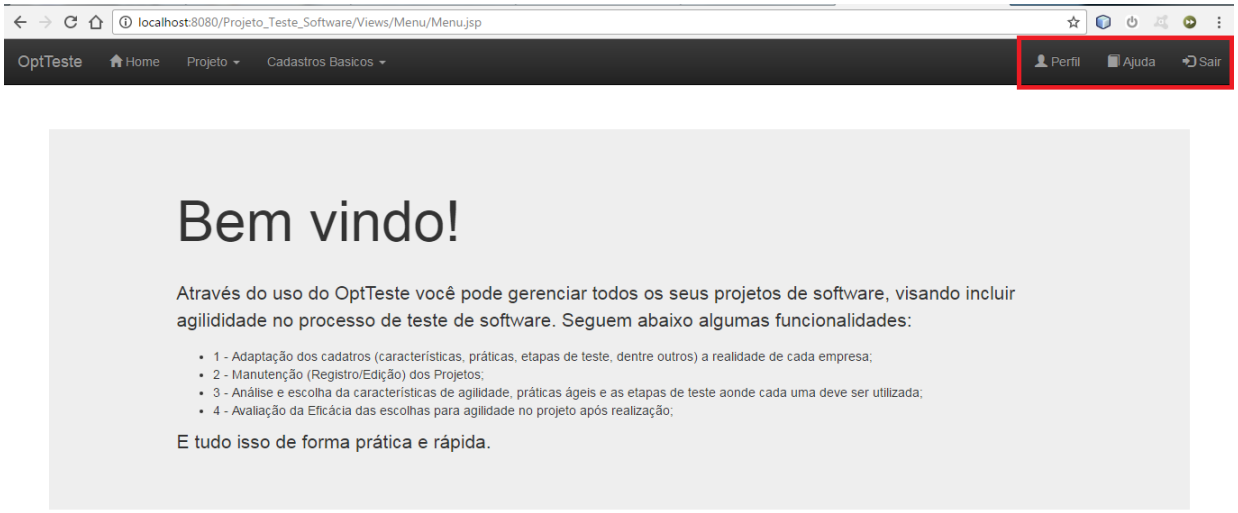

Figura 3.7: Opções de alterar perfil, Ajuda/Help e Sair

- Cadastros Básicos (em destaque na Figura 3.8);

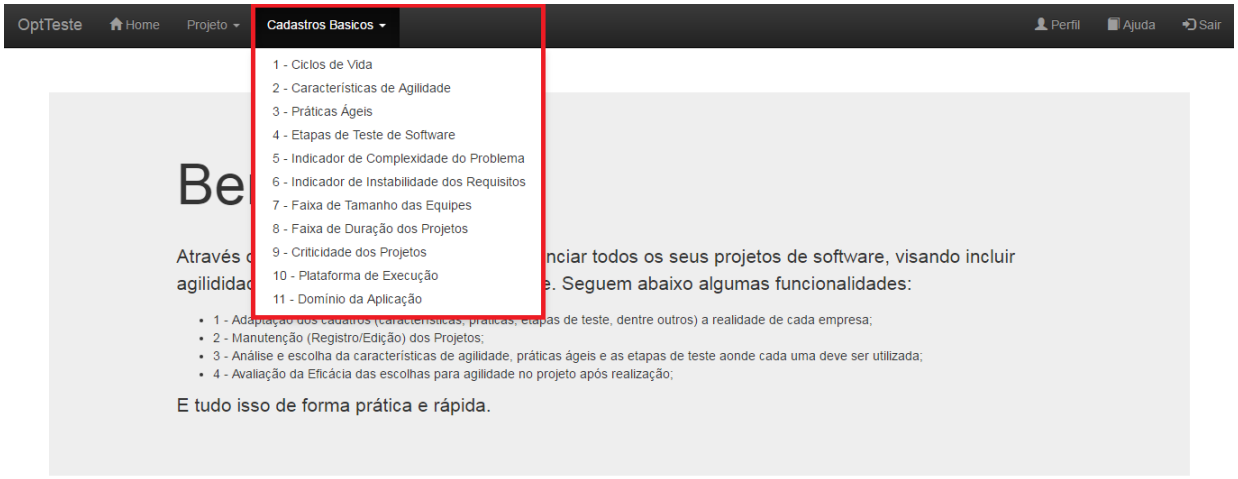

Figura 3.8: Menu de Cadastros Básicos

- Inclusão/Edição de novos projetos;

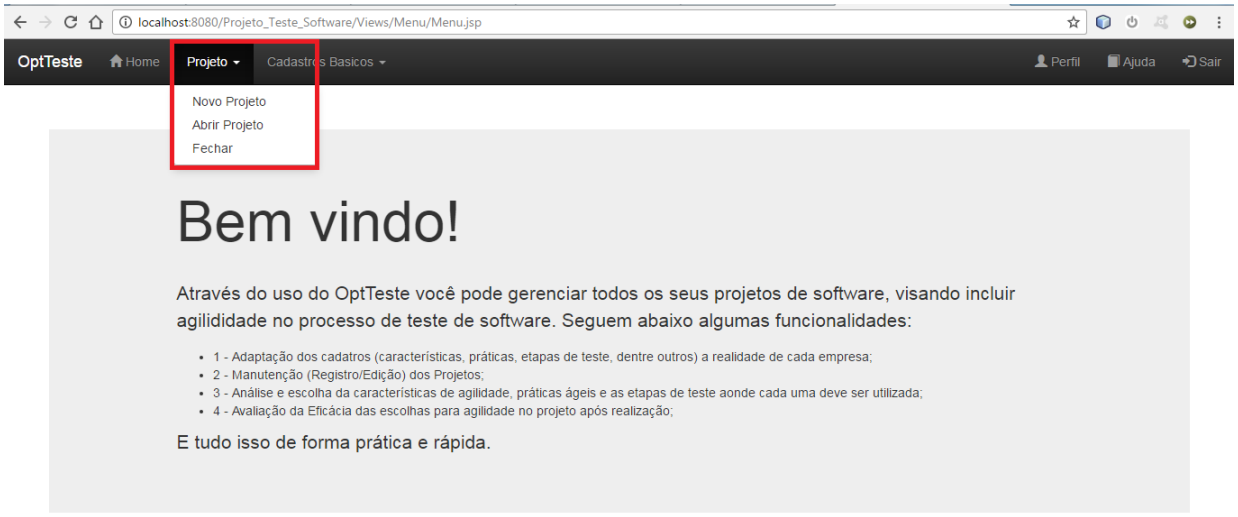

Figura 3.9: Menu de Inclusão e Edição de Projetos

3.6.2 Cadastros Básicos

Em todos os cadastros listados abaixo o usuário possui todas as opções para manter o cadastro (criar, editar e excluir os dados), sendo que criar e editar são as mesmas funções e por isso serão exibidos apenas as telas: tela inicial do cadastro e criação/edição.

Cadastro dos Modelos de Ciclo de Vida de Projetos

- Tela dos Modelos de Ciclo de Vida de Projetos;

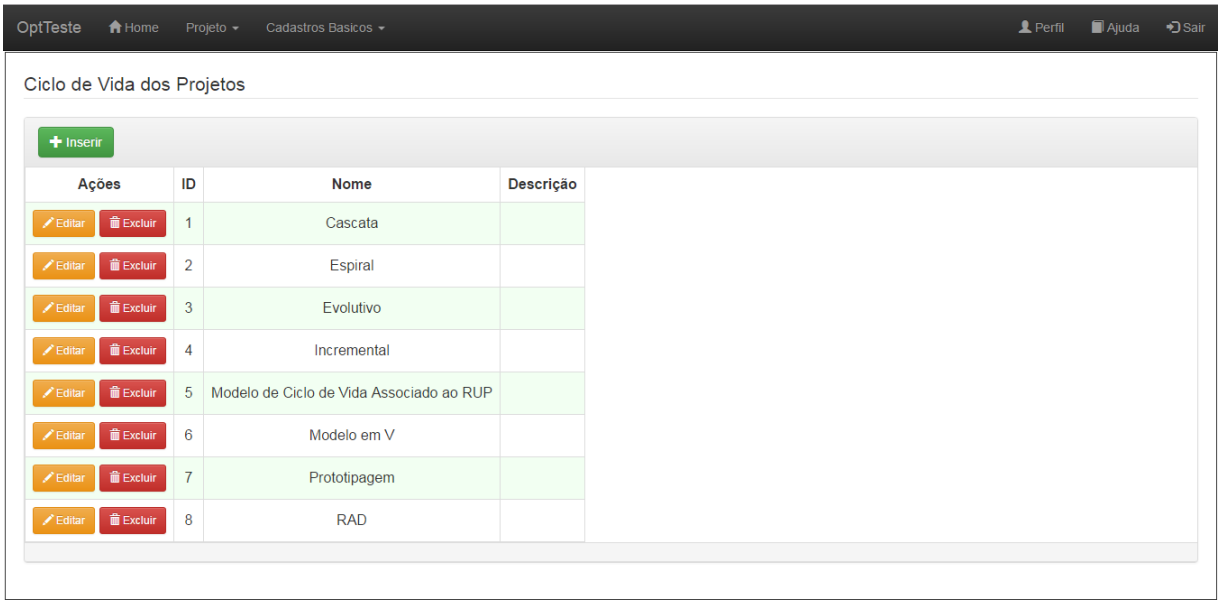

Figura 3.10: Tela inicial dos Modelos de Ciclo de Vida de Projetos

- Inclusão/Edição de Modelos de Ciclo de Vida de Projetos;

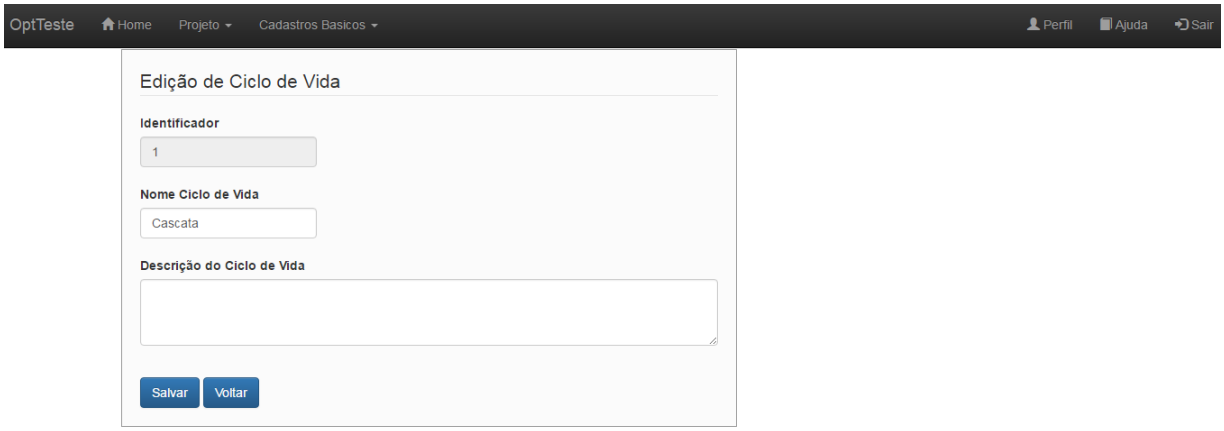

Figura 3.11: Tela de Inclusão e Edição de Modelos de Ciclo de Vida de Projetos

Cadastro de Características de Agilidade

- Tela das Cadastro de Características de Agilidade;

Figura 3.12: Tela inicial das Cadastro de Características de Agilidade

- Inclusão/Edição de Características de Agilidade;

Figura 3.13: Tela de Inclusão e Edição de Características de Agilidade

Cadastro de Práticas Ágeis de Software Adotadas

- Tela das Práticas Ágeis de Software Adotadas;

Figura 3.14: Tela inicial das Práticas Ágeis de Software

- Inclusão/Edição de Práticas Ágeis;

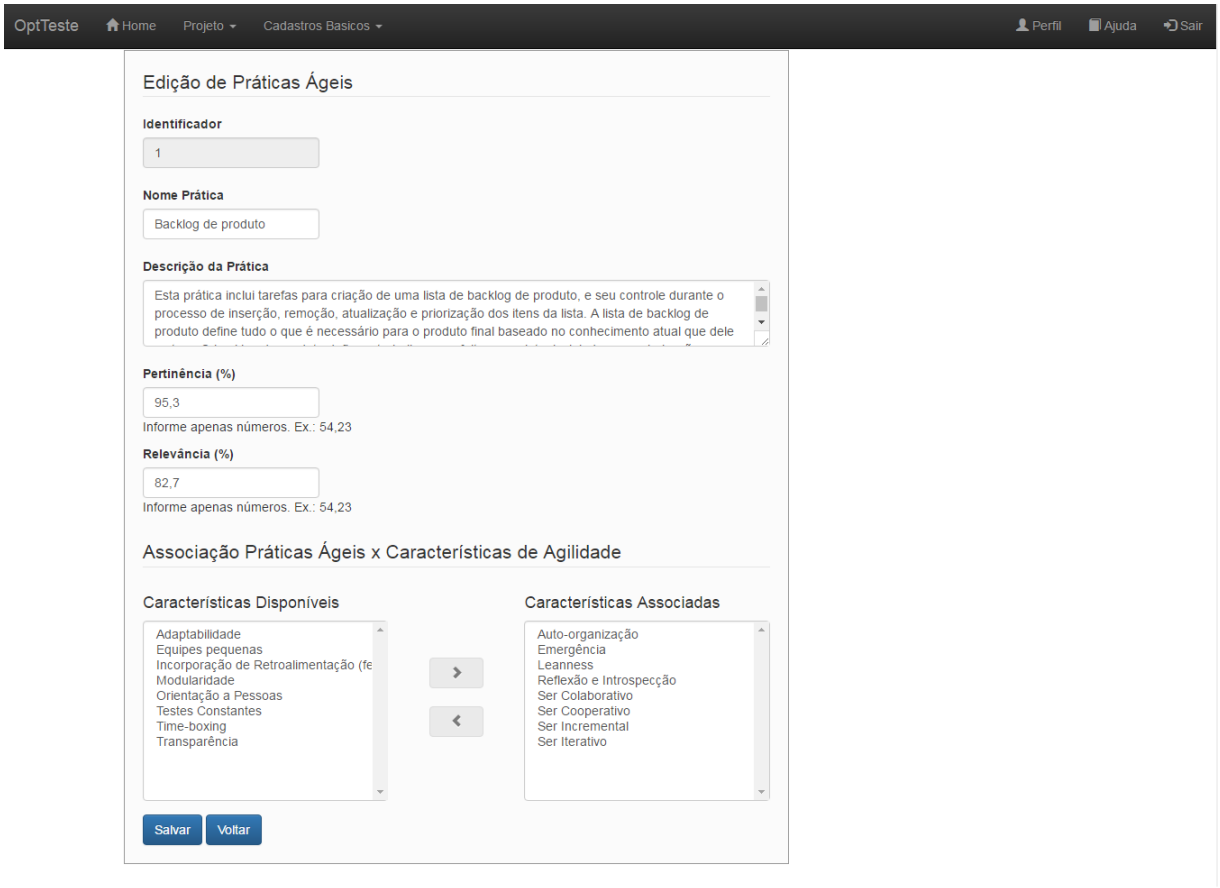

Figura 3.15: Tela de Inclusão e Edição de Práticas Ágeis de Software Adotadas

Cadastro de Etapas de Teste de Sotware

- Tela das Etapas de Teste de Sotware;

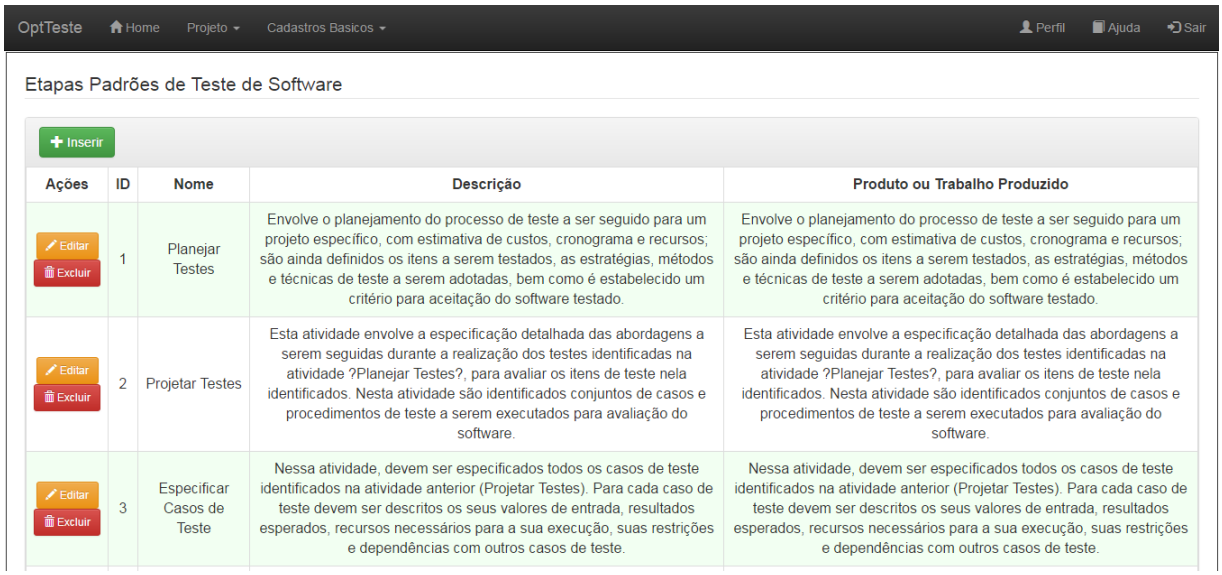

Figura 3.16: Tela inicial das Etapas de Teste de Sotware

- Inclusão/Edição das Etapas de Teste de Sotware;

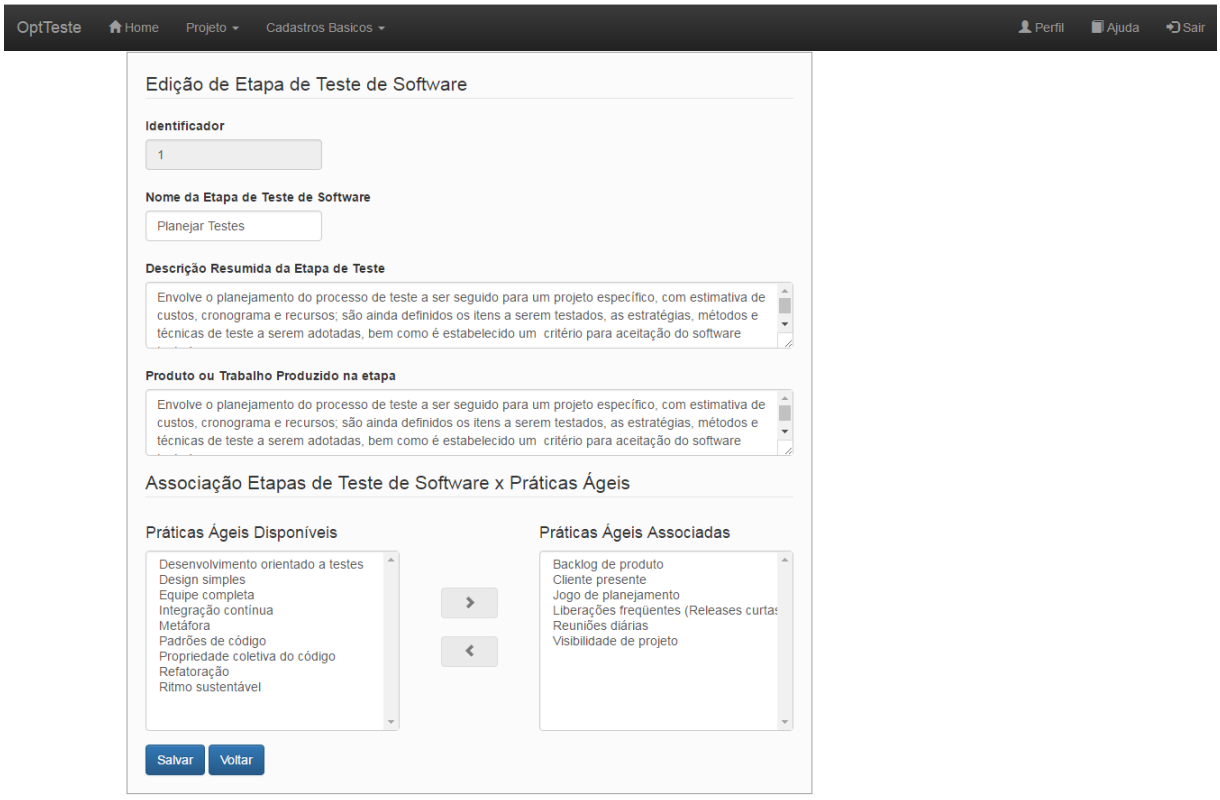

Figura 3.17: Tela de Inclusão e Edição das Etapas de Teste de Sotware

Cadastro de Indicador de Complexidade do Problema

- Tela dos Modelos de Indicador de Complexidade do Problema;

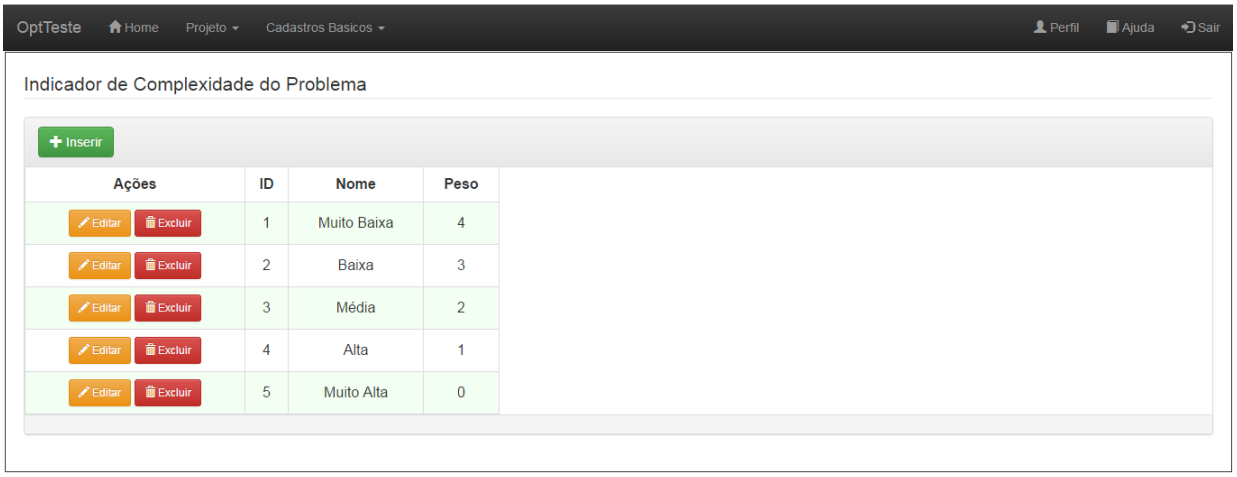

Figura 3.18: Tela inicial do Indicador de Complexidade do Problema

- Inclusão/Edição do Indicador de Complexidade do Problema;

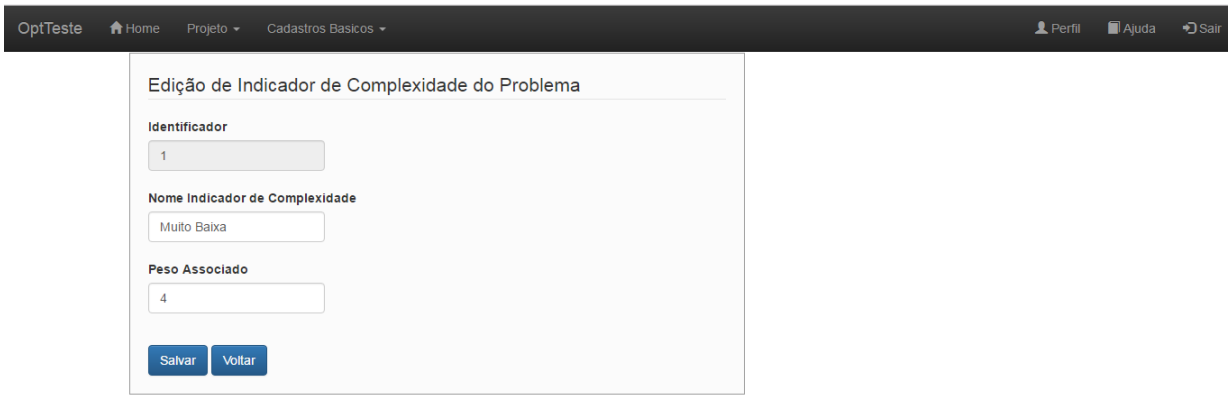

Figura 3.19: Tela de Inclusão e Edição do Indicador de Complexidade do Problema

Cadastro de Indicador de Instabilidade dos Requisitos

- Tela do Indicador de Instabilidade dos Requisitos;

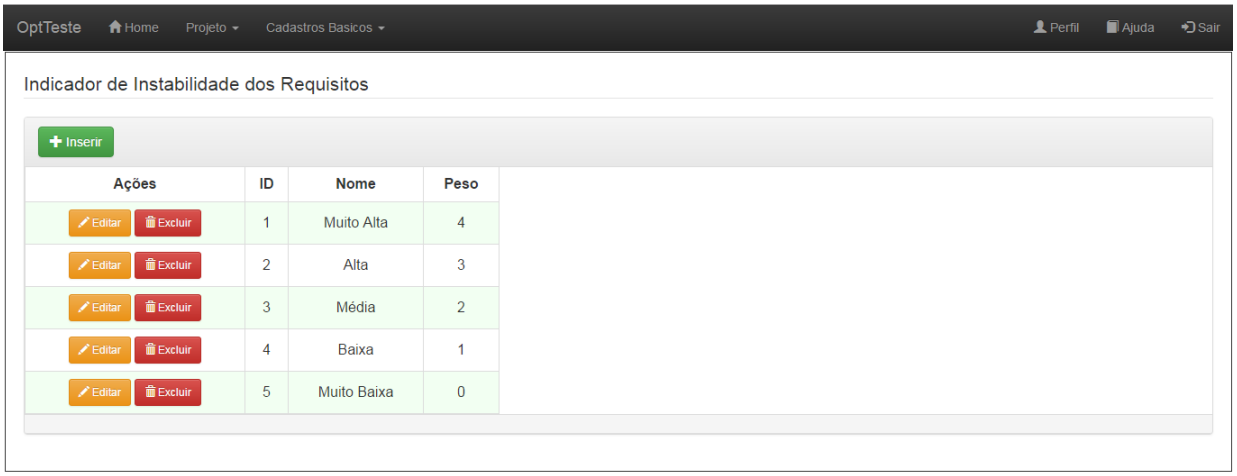

Figura 3.20: Tela inicial do Indicador de Instabilidade dos Requisitos

- Inclusão/Edição do Indicador de Instabilidade dos Requisitos;

Figura 3.21: Tela de Inclusão e Edição do Indicador de Instabilidade dos Requisitos

Cadastro de Faixa do Tamanho da equipe

- Tela da Faixa do Tamanho da equipe;

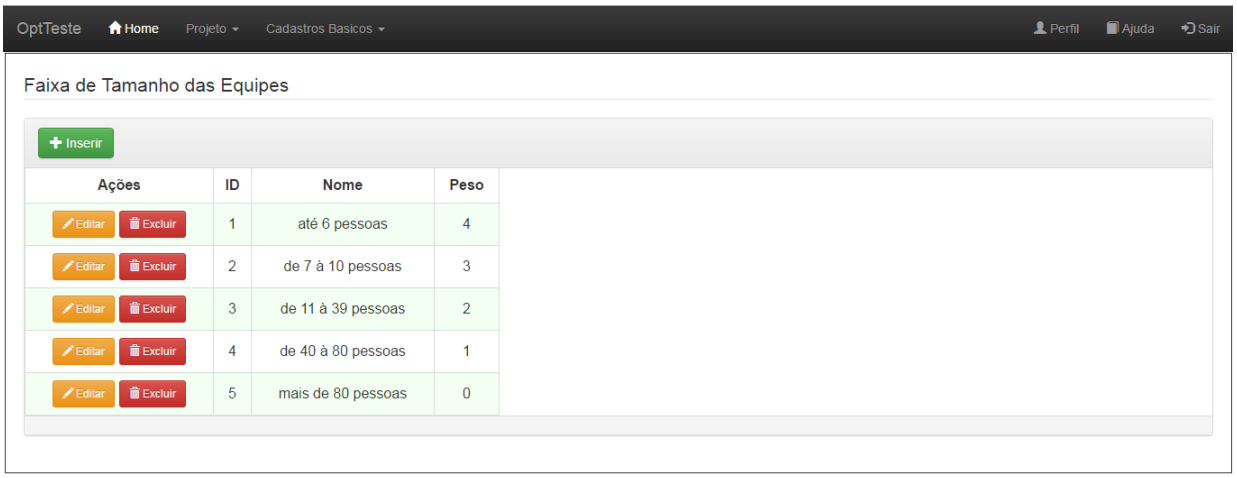

Figura 3.22: Tela inicial da Faixa do Tamanho da equipe

- Inclusão/Edição da Faixa do Tamanho da equipe;

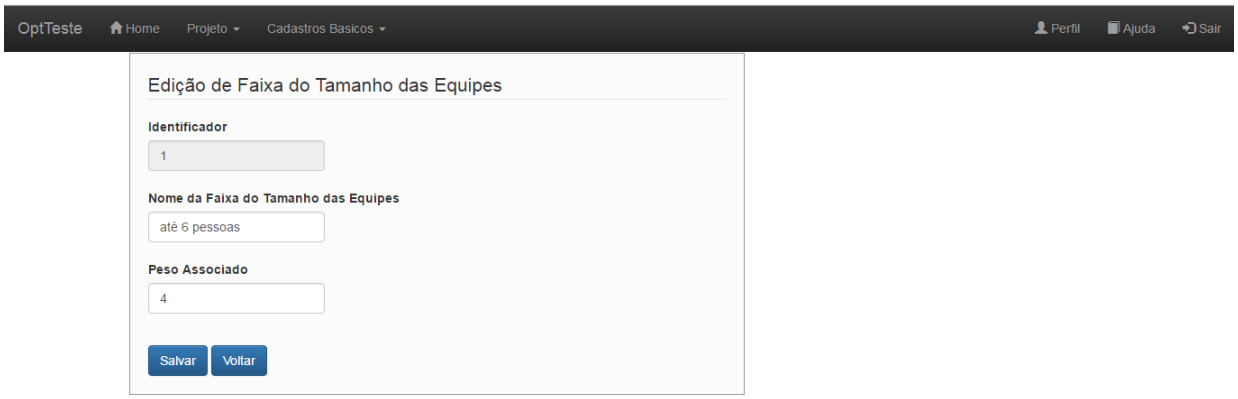

Figura 3.23: Tela de Inclusão e Edição da Faixa do Tamanho da equipe

Cadastro de Faixa de Duração do Projeto

- Tela da Faixa de Duração do Projeto;

Figura 3.24: Tela inicial da Faixa de Duração do Projeto

- Inclusão/Edição da Faixa de Duração do Projeto;

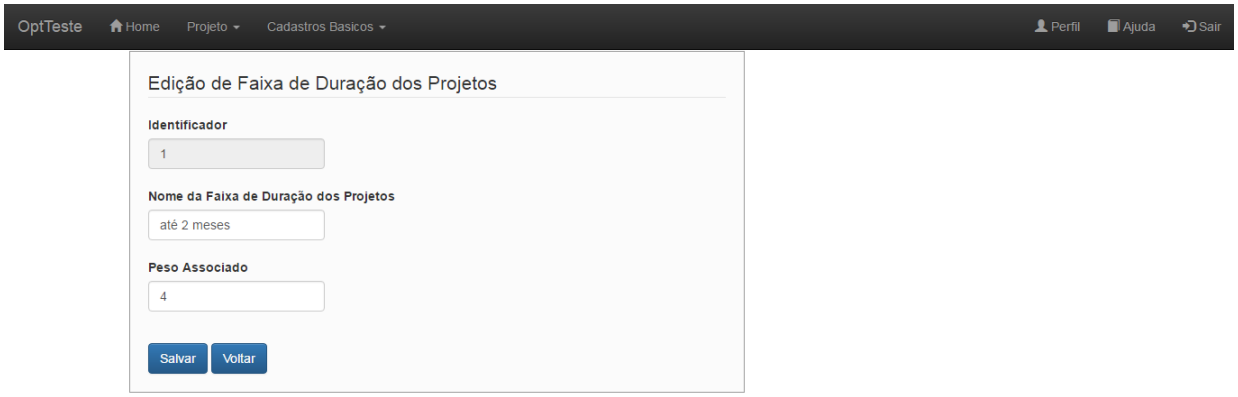

Figura 3.25: Tela de Inclusão e Edição da Faixa de Duração do Projeto

Cadastro de Criticidade do Projeto

- Tela da Criticidade do Projeto;

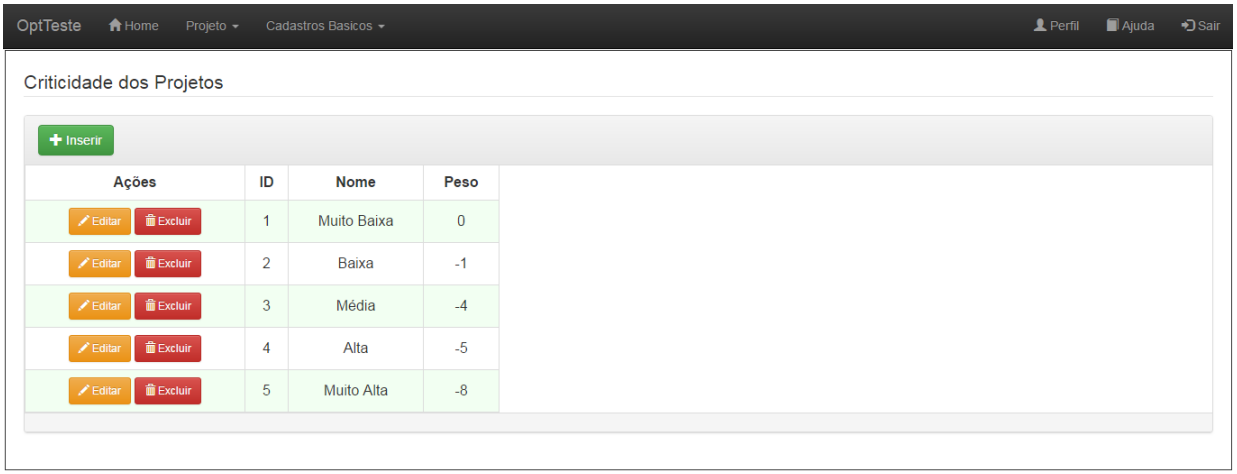

Figura 3.26: Tela inicial da Criticidade do Projeto

- Inclusão/Edição da Criticidade do Projeto;

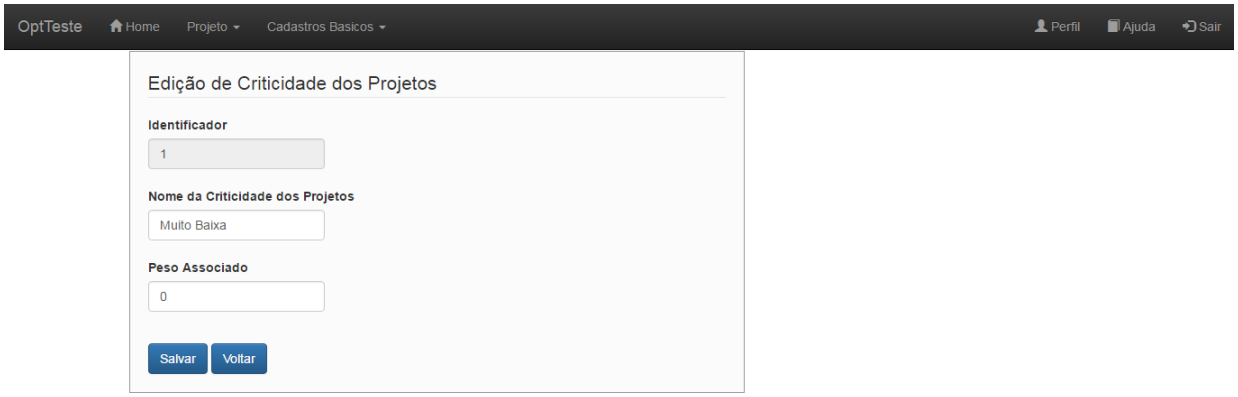

Figura 3.27: Tela de Inclusão e Edição da Criticidade do Projeto

Cadastro de Plataforma de Execução

- Tela da Plataforma de Execução;

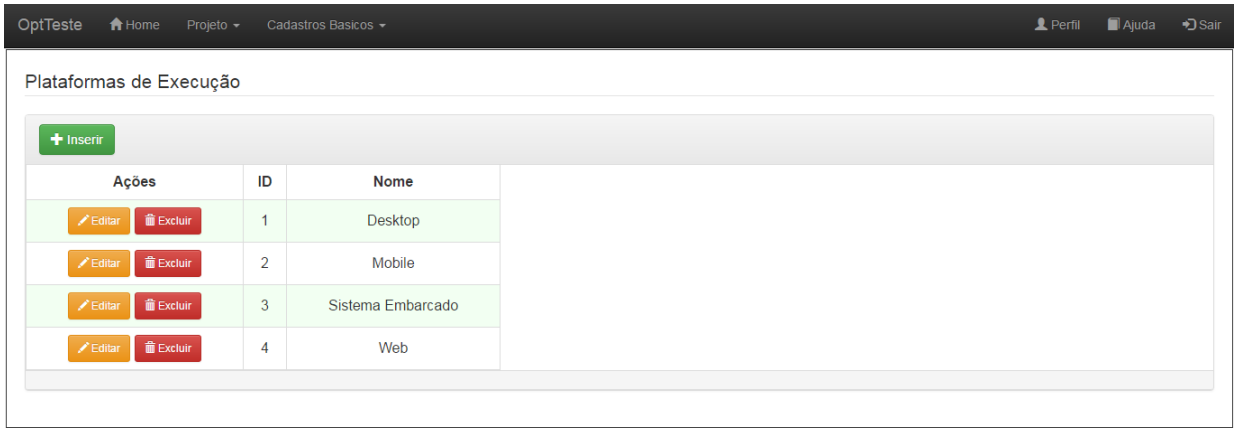

Figura 3.28: Tela inicial da Plataforma de Execução

- Inclusão/Edição da Plataforma de Execução;

Figura 3.29: Tela de Inclusão e Edição da Plataforma de Execução

Cadastro de Domínio da Aplicação

- Tela do Domínio da Aplicação;

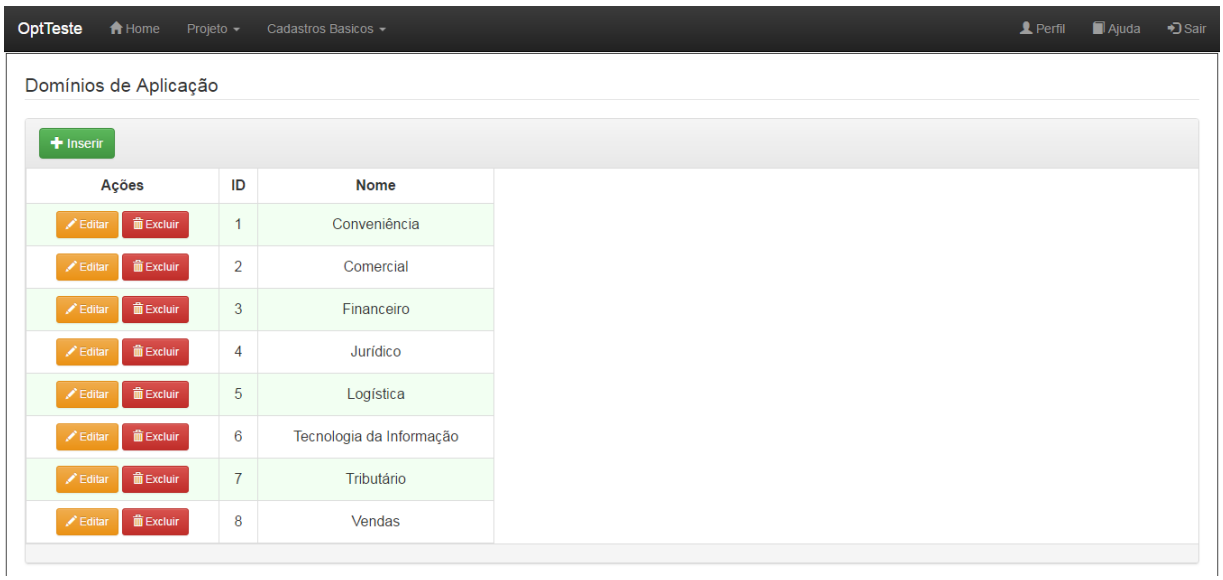

Figura 3.30: Tela inicial do Domínio da Aplicação

- Inclusão/Edição do Domínio da Aplicação;

Figura 3.31: Tela de Inclusão e Edição do Domínio da Aplicação

3.6.3 Inclusão/Edição de novos projetos

Após acessar a opção de Abrir Projeto, o usuário pode manter (Inserir, Editar e excluir) um projeto.

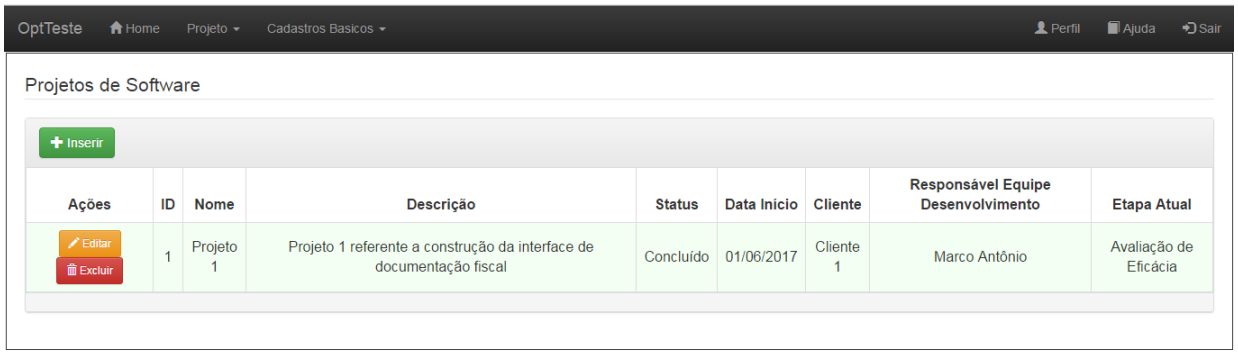

Figura 3.32: Tela inicial com os Projetos Cadastrados

Na sequência serão citadas cada uma das etapas de um projeto.

Etapa I - Dados Básicos do Projeto

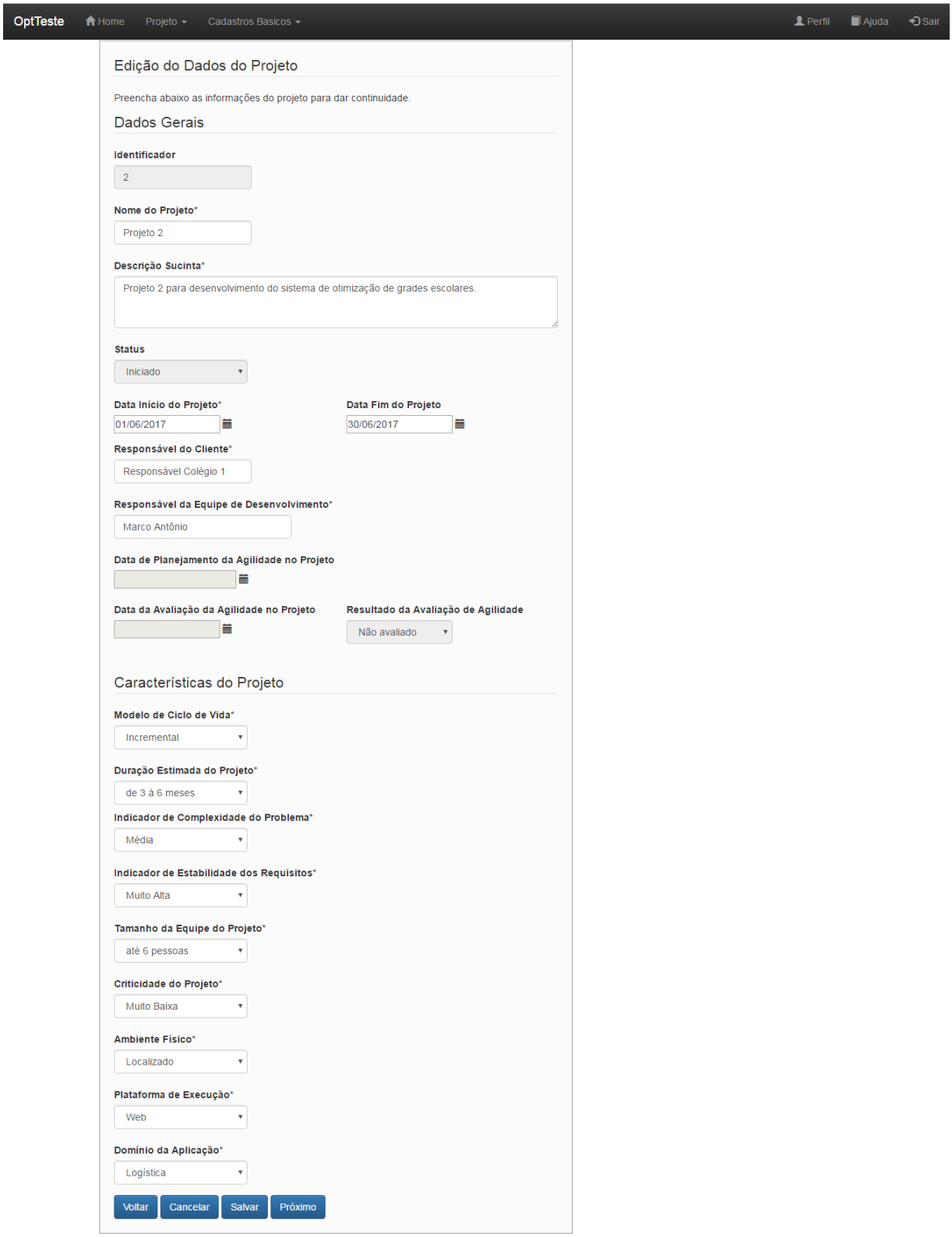

Figura 3.33: Informações Básicas do Projeto

Etapa II - Adequação do Projeto às abordagens Ágeis

A sugestão do sistema pode ser positiva ou negativa, dependendo dos parâmetros informados anteriormente.

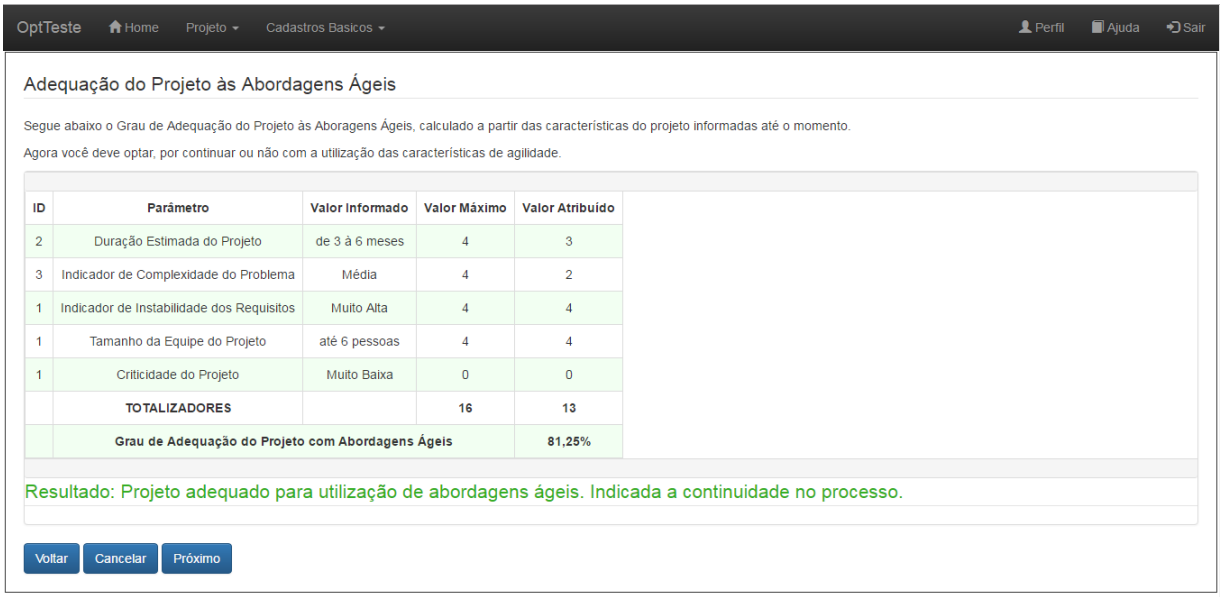

Figura 3.34: Tela de sugestão da Adequação do projeto às abordagens ágeis a partir das características gerais do mesmo. Sendo considerado adequado caso obtenha valor acima de 50%

Escolha das Características de Agilidade e Exibição das Práticas Associadas

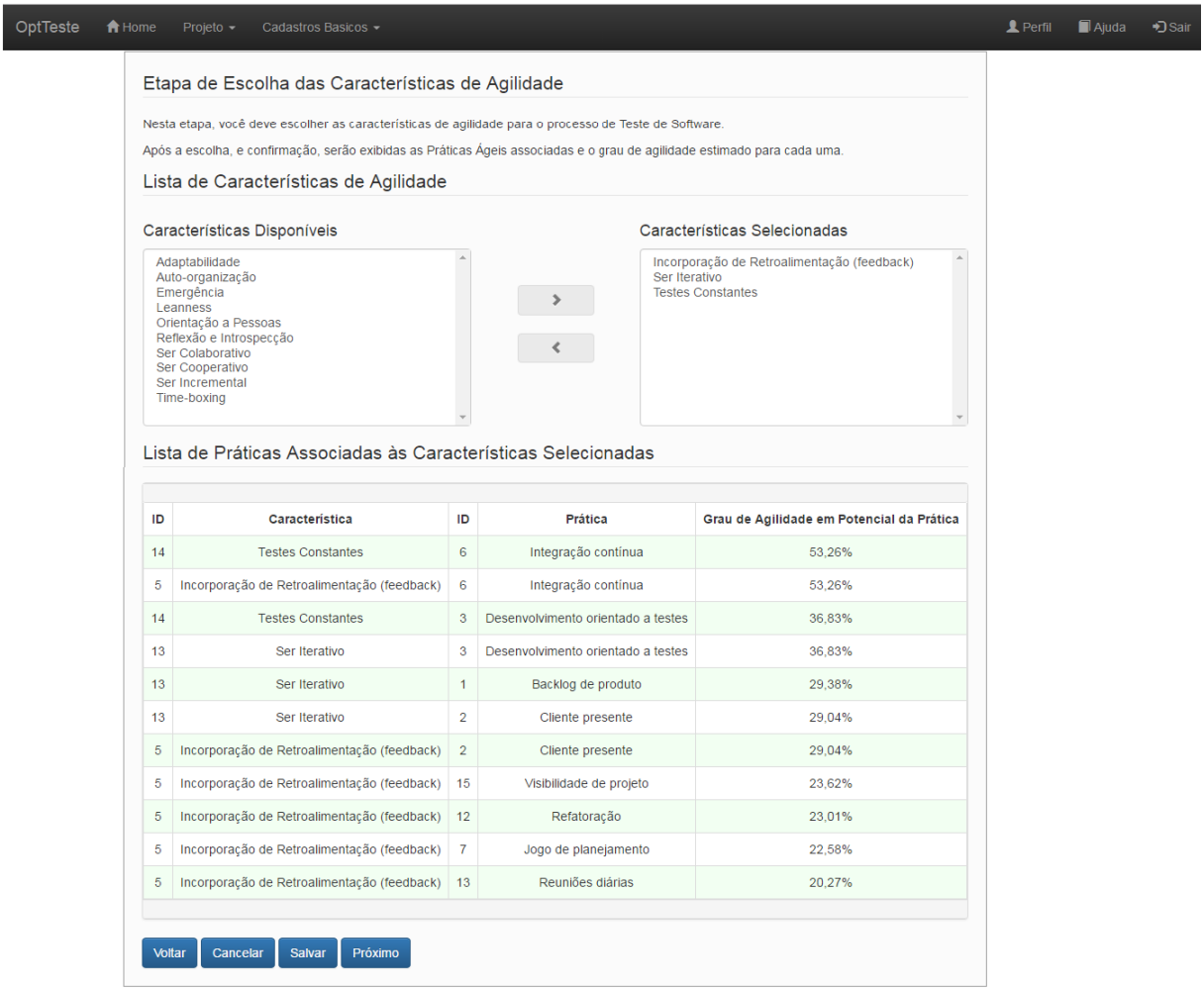

Figura 3.35: Tela de escolha das características de agilidade

Práticas Ágeis x Etapas de Teste de Software

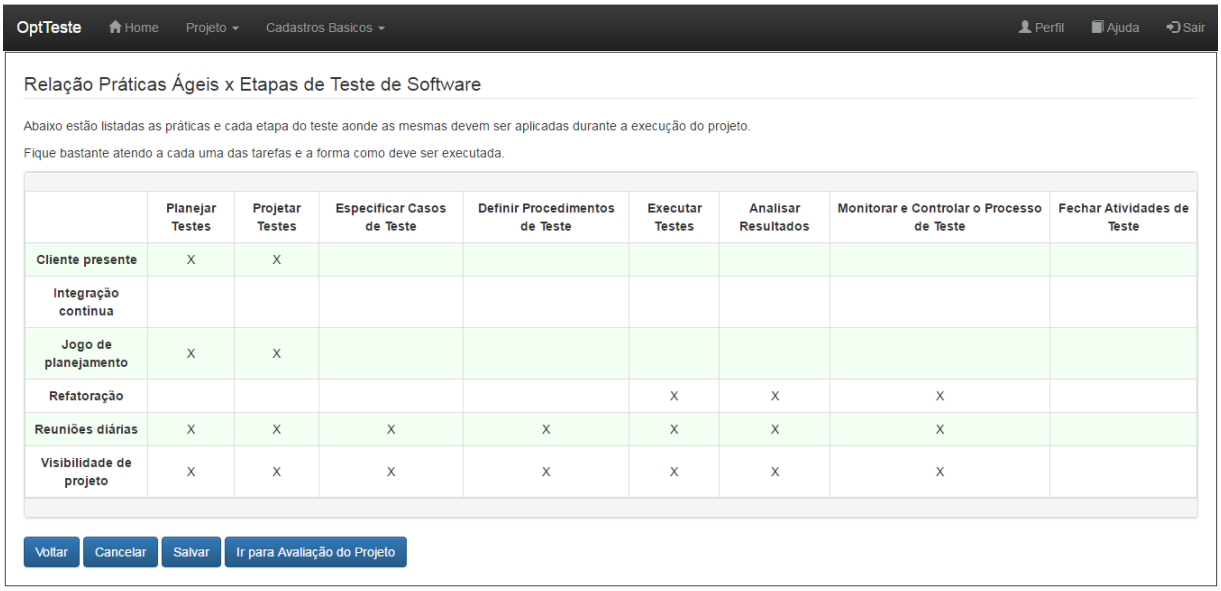

Figura 3.36: Tela de associação das Práticas Ágeis com as Etapas de Teste de Software

3.6.4 Avaliação de Eficácia

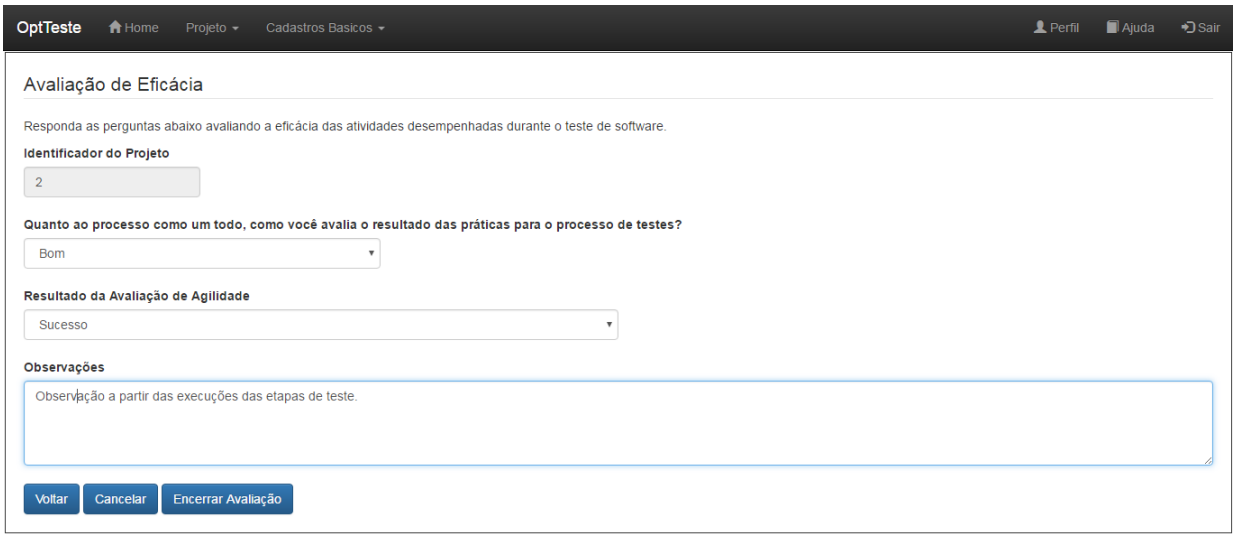

Figura 3.37: Tela de avaliação final do projeto

3.7 Conclusão

Neste capítulo foi explicado todo o detalhamento da estratégia, estruturação técnica, funcionalidades e uma apresentação das telas do sistema, seguindo a sequência usual da

ferramenta, com o objetivo de servir como guia básico e para ilustrar a utilização da mesma. Assim, podendo servir de apoio aos usuários.

4 Considerações Finais

Neste capítulo é apresentada uma avaliação do projeto, resultados, as contribuições da pesquisa, suas limitações, sugestões de trabalhos futuros para continuidade e melhoria da mesma.

4.1 Avaliação da Projeto

Analisando o projeto como um todo é possível observar características positivas, mesmo tendo consciência das limitações e possibilidades de crescimento (detalhados melhor adiante). Podemos citar, por exemplo:

- Desenvolvimento da ferramenta para apoiar a decisão de quais práticas podem ser utilizadas para acrescentar maior agilidade no processo e consequentemente o cumprimento do objetivo principal proposto;
- A estratégia abordada nos indica que é possível embutir características de agilidade e práticas ágeis no processo de teste, mas sendo necessário o acompanhamento prático para observar o desempenho diante de soluções ad-hoc (comumente utilizadas). E por se tratar da aplicação de uma técnica já existente, foi desconsiderada uma avaliação/medição formal (como: entrevistas, avaliações de especialistas e aplicação de caso em disciplina), pois não traria um ganho considerável;
- A ferramenta desenvolvida leva em consideração que o processo é muito aberto e que a própria área de teste ainda pode sofrer mudanças e desta forma o mesmo foi pensado de forma a ser o mais parametrizado possível para evitar o engessamento do processo;
- Tanto a abordagem, como o projeto em si pode apoiar e/ou servir de hipótese para o surgimento de novos trabalhos correlacionados ao tema, aprofundando o conhecimento quanto à agilidade no processo de desenvolvimento;

4.2 Limitações e Trabalhos Futuros

Considerando interessante a perspectiva de melhoria constante e após a construção e avaliação do projeto, foram levantados alguns pontos que podem ser levados em consideração tanto para aprimorar este trabalho, como em outros estudos, buscando inserir agilidade diretamente ao processo de teste de software e/ou outra etapa do processo de software. Seguem abaixo os itens:

- Aplicação e acompanhamento do projeto desenvolvido dentro de organizações para uma real medição do ganho estimado e acréscimo de uma maior validade da pesquisa;
- Comparação com projetos históricos: o registro das informações durante as etapas de teste de software abrem caminho para uma análise histórica mais profunda, permitindo auxiliar na tomada de decisão a partir de outros projetos que apresentem o mesmo cenário;
- Exemplificação de como utilizar as práticas ágeis: uma forma de reduzir a complexidade no uso do projeto e evitar a necessidade de o usuário possuir um grande conhecimento técnico sobre as práticas cadastradas é a existência de uma integração ou plugin para a execução das práticas ágeis no projeto;
- Adicionar os tipos de teste de software que podem ser utilizados em cada prática ou etapa de teste e possíveis integrações com softwares que já executam os testes para facilitar a experiência de uso do usuário. Este tópico demanda um estudo e relacionamento entre os tipos de teste e as prática ou etapa de teste;
- No cadastro de Faixa de Tamanho da Equipe é válida a especificação do mesmo, permitindo o detalhamento em dois aspectos (que podem contribuir muito para decisão de continuidade ou não do uso das características e práticas no projeto):
    1. Experiência média da equipe ou de cada participante com abordagens ágeis;
    2. Experiência média da equipe ou dos participantes nas tecnologias envolvidas, como por exemplo: Analista Júnior, Analista Pleno e Analista Sênior);

- No cadastro de Plataforma de Execução ou então em um novo cadastro associado a este, é válido acrescentar às tecnologias e linguagens de desenvolvimento que serão utilizadas, sendo possível a relação de experiência com cada membro da equipe (item anterior) e que trás mais um nível de detalhamento para auxílio na decisão de continuidade ou não do projeto e inclusive de comparações com projetos históricos;

- Integrações com ferramentas de Gestão de Projetos (por exemplo o RedMine) e/ou diretamente ligadas ao rastreamento e histórico de defeitos (por exemplo o Bugzilla), pois as integrações são importantes para evitar entradas manuais, o que reduz o subjetivismo, desburocratiza o processo e ocupa menos tempo que já é escasso dentro do projeto;

- Avaliar a possibilidade de estender o projeto para as demais etapas do processo de desenvolvimento de software, sendo facilitada a análise a partir do acompanhamento do uso da ferramenta;

4.3 Conclusões

Embora o tema proposto seja em uma área que ainda está em franco crescimento e que a mesma não possuí estrutura determinística para comprovar todos os casos em que são válidos ou não o uso de agilidade, são exatamente estes os pontos que podem trazer grandes ganhos para a área da computação e um consequente reflexo positivo na indústria, e por isso a importância da abordagem de temas similares. Com este projeto, conseguimos observar que a área de Teste de Software têm a possibilidade de conciliar agilidade no seu processo e que pode obter resultados satisfatórios, mesmo que ainda sejam necessárias adequações ao projeto e até estudos de caso para observar o desempenho da estratégia descrita. O trabalho foi uma ótima oportunidade para buscar maior conhecimento da área e aplicar os já adquiridos durante a formação de Bacharel em Ciência da Computação, e deixando portas abertas para a continuidade/expansão do mesmo.

Referências Bibliográficas

Abrantes, J. F. **Estudos experimentais sobre agilidade no desenvolvimento de software e sua útil utilização no processo de teste**. Tese de Doutorado, 2012.

BOEHM, B. e TURNER, R. **Balancing agility and discipline: A guide for the perplexed**. 2004.

Bastos, A.; Rios, E.; Cristalli, R. ; Moreira, T. **Base de conhecimento em teste de software**. p. 32. Alta Books, 2012.

Boehm, B.; Basili, V. **Software defect reduction top 10 list**. 2001.

Boehm, B. **Making a difference in the software century**. IEEE computer society, 2008.

Borjesson, A.; Mathissen, L. **Improving software organizations: agility challenges and implications**. volume 18 de **n.4**, p. 359–382, 2005.

Dijkstra, E. **Notes on structured programming**. circulated privately, Abril, 1970.

IEEE. **Ieee standard 610-1990: Ieee standard glossary of software engineering terminology**. IEEE Press, 1990.

Kongsli, V. **Agile security in web applications**. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, p. 805–808, 2006.

MNKANDLA, E. **A selection framework for agile methodology practices: A family of methodologies approach**. PhD Thesis, 2008.

de Macêdo, A. B. L.; Spínola, R. O. **Ciclos de vida do software**. p. 21–28, 2011.

Rios, E.; Moreira, T. **Teste de software**. Alta Books, 2003.

Myers, G. J. **The art of software testing**. John Wiley & Sons, 2004.

Neto, A. D. **Introdução a testes de software**. p. 54–59. IEEE Press, 2007.

de Pádua, W. **Engenharia de software: fundamentos, métodos e padrões**. 2000.

Pressman, R. S. **Engenharia de software**. Mc Graw Hill, 2010.

QUMER, A. HENDERSON-SELLERS, B. **An evaluation of the degree of agility in six agile methods and its applicability for method engineering**. p. 280–295, 2008.

5 Apêndice

Este capítulo é reservado aos documentos extraídos de outros projetos e que são relevantes e/ou servem de apoio.

5.1 Identificando a Pertinência das Características de Agilidade e das Práticas Ágeis

Segue abaixo explicação quanto a montagem dos campos, retirada integralmente de Abrantes (2012):

- Identificando a Pertinência das Características de Agilidade e das Práticas Ágeis

  Uma vez que o conjunto de participantes para avaliar as características de agilidade e as práticas ágeis é o mesmo, os cálculos para definir quais são as características e práticas pertinentes também são os mesmos, dependendo apenas das diferentes respostas dos participantes para os dois conjuntos de características e práticas.

  Para definir quais características de agilidade e práticas ágeis são pertinentes no contexto das abordagens ágeis de software, é necessário primeiramente computar as respostas de cada participante e considerar seu respectivo peso, tanto para as características quanto para as práticas. A computação das respostas com os respectivos pesos, bem como o patamar de corte adotado foi efetuada através das fórmulas a seguir apresentadas, adaptadas do trabalho de FARIAS (2002) sobre planejamento de riscos em ambientes de desenvolvimento de software:

  $Pertinence(j) = \sum_{i=1}^{m}(Answer(i,j) * S(i))$ onde:

  - Pertinence(j) é o valor total das respostas de todos os participantes

(multiplicadas por seus respectivos pesos) sobre a pertinência da característica/prática j;

- Answer(i, j) é o indicador de pertinência (1) ou não pertinência (0) definido pelo participante i para a característica/prática j.
- S(i) é o peso atribuído ao participante i;
- m é o total de participantes que responderam à pesquisa de opinião.

A definição se uma característica de agilidade ou prática ágil é pertinente ou não pertinente deve estar baseada em um ponto de corte, isto é, um patamar indicando se a característica ou prática está incluída no conjunto final (valor maior ou igual ao patamar). O patamar adotado para este estudo é 50% do valor máximo que poderia ser obtido para a característica/prática j na variável Pertinence(j) se todos os participantes respondessem sim com relação à sua pertinência no contexto das abordagens ágeis de software.

$Threshold = 0,5 * \sum_{i=1}^{m} S(i)$ onde:

- S(i) é o peso atribuído ao participante i
- m é o total de participantes que responderam à pesquisa

Portanto o critério fica assim:

- Se Pertinence(j) $<$ Treshold então a característica/prática j é classificada como não pertinente e deve ser removida do conjunto inicial.
- Se Pertinence(j) $\geq$ Treshold então a característica/prática j é classificada como pertinente e deve ser mantida no conjunto inicial.

Ao conjunto obtido a partir do conjunto inicial de características de agilidade ou prática ágil, de acordo com o patamar estabelecido, devem ser adicionadas as características ou práticas indicadas pelos participantes para completar o conjunto inicial, desde que as características e/ou práticas indicadas tenham sido descritas quanto ao seu significado. As características e práticas incluídas pelos participantes serão consideradas como pertinentes.

- Identificando a Ordem de Relevância das Características de Agilidade e das Práticas Ágeis

Para definir o nível de relevância de uma característica de agilidade ou prática ágil classificada previamente como pertinente, é necessário primeiramente somar as respostas de cada particimante (multiplicadas por seus respectivos pesos).

$RLevel(j) = \sum_{i=1}^{m}(Scale(i,j) * S(i))$ onde:

- RLevel(j) é o valor total das respostas de todos os participantes (multiplicadas por seus respectivos pesos) para cada característica/prática j
- m é o total de participantes que responderam à pesquisa
- Scale(i, j) é a escala de nível de relevância (0-3) definida pelo participante i para a característica/prática j
- S(i) é o peso atribuído ao participante i

Após este passo, as características de agilidade e práticas ágeis serão ordenadas por seus respectivos RLevel(j). A característica e prática mais relevante será aquela com maior valor de RLevel(j).

5.2 Lista de Características de Agilidade

Tabela 5.1: Lista de Características de Agilidade

Característica	**Descrição**	**Pertinência**	**Relevância**
Ser Colaborativo	Esta é uma atitude por parte dos membros da equipe de desenvolvimento, dentre os quais a comunicação é encorajada para disseminar informação e apoiar integração rápida de incrementos de software.	94,10%	84,70%
Continua na próxima página			

Tabela 5.1 – Continuação da página anterior

Característica	Descrição	Pertinência	Relevância
Ser Cooperativo	Interação aberta e proximidade entre todos as partes interessadas (especialmente entre clientes e desenvolvedores); o cliente deve ser um elemento ativo no processo de desenvolvimento e deve prover retroalimentação (feedback) de modo regular e frequente.	90,90%	79,30%
Ser Incremental	Não tentar construir o sistema todo de uma só vez; o sistema deve ser particionado emincrementos (pequenas liberações com novas funcionalidades) desenvolvidos em ciclos rápidos e paralelos; quando o incremento estiver completo e testado, ele é integrado ao sistema.	94,10%	83,30%
Adaptabilidade	Habilidade e capacidade de adaptar rapidamente o processo para atender e reagir a mudanças de última hora nos requisitos e/ou mudanças de ambiente, bem como atender e reagir a riscos ou situações não previstas.	100,00%	88,70%
Auto-organização	As equipes definem as melhores maneiras de se trabalhar; elas são autônomas e podem se auto-organizar de acordo para completar os itens de trabalho.	76,30%	57,20%
Ser Iterativo	Usar vários ciclos curtos guiados por funcionalidades do produto, nos quais certo conjunto de atividades é completado em poucas semanas; estes ciclos são repetidos muitas vezes pra refinar as entregas.	92,50%	84,80%

Continua na próxima página

Tabela 5.1 – Continuação da página anterior

Característica	Descrição	Pertinência	Relevância
Testes Constantes	Para prevenir a degradação da qualidade por causa de programas de liberações frequentes, um alto grau de ênfase é colocado no teste do produto ao longo de seu ciclo de vida; testes de integração devem ser automatizados com construções (builds) diárias bem como a execução de testes de regressão para garantir que todas as funcionalidades operem dequadamente.	93,80%	84,10%
Emergência	Os processos, princípios e estruturas de trabalho são reconhecidos durante o processo de execução, não sendo definidos a priori; é permitido e aceito que tecnologias e requisitos emerjam durante o ciclo de vida do produto.	74,10%	57,70%
Incorporação de Retroalimentação (feedback)	As equipes devem ser capazes de receber e procurar continuamente por etroalimentação de modo mais frequente e com mais rapidez.	79,70%	84,10%
Leanness	Esta característica está relacionada com a eliminação de perdas e com a habilidade derealizar mais trabalho com menos esforço; é uma característica de processos ágeis querequer o mínimo necessário de atividades para mitigar riscos e alcançar metas; todas as atividades que não são necessárias devem ser removidas do processo de desenvolvimento.	61,30%	56,50%

Continua na próxima página

Tabela 5.1 – Continuação da página anterior

Característica	Descrição	Pertinência	Relevância
Modularidade	Esta característica permite que um processo seja particionado em componentes chamados de atividades, tornando viável adicioná-los ou removê-los de um processo quando necessário.	64,40%	54,00%
Orientação aPessoas	Privilegiar pessoas em detrimento de processos e tecnologias; desenvolvedores são fortalecidos e encorajados a aumentar sua produtividade, qualidade e desempenho; comunicação e cooperação são fundamentais e necessárias; reuniões em pé e oficinas (workshops) de reflexão dão às pessoas a chance de expor suas preocupações.	98,40%	89,10%
Reflexão e Introspecção	Acontecem reuniões no final de cada subprojeto ou iteração, nas quais cada membro da equipe pode discutir o que está sendo bem feito e o que precisa ser melhorado.	98,40%	80,80%
Equipes pequenas	Equipes pequenas, e pequeno número de equipes por projeto, são necessários para promover um ambiente colaborativo e requer menos planejamento para coordenar as atividades dos membros das equipes.	71,60%	43,20%

Continua na próxima página

Tabela 5.1 – Continuação da página anterior

Característica	Descrição	Pertinência	Relevância
Time-boxing	É o estabelecimento de limite ou fatias de tempo para cada iteração programada. Grandes esforços de desenvolvimento são divididos em múltiplas entregas desenvolvidas de modo incremental e concorrente, de maneira previsível.	93,10%	63,50%
Transparência	O método ou processo ágil deve ser fácil de aprender e de ser modificado, além de estar adequadamente documentado.	61,60%	62,80%

5.3 Lista de Práticas Ágeis

Tabela 5.2: Lista de Práticas Ágeis

Prática	**Descrição**	**Pertinência**	**Relevância**
Backlog de produto	Esta prática inclui tarefas para criação de uma lista de backlog de produto, e seu controle durante o processo de inserção, remoção, atualização e priorização dos itens da lista. A lista de backlog de produto define tudo o que é necessário para o produto final baseado no conhecimento atual que dele se tem. O backlog de produto define o trabalho a ser feito no projeto, incluindo uma priorização e constante atualização da lista de requisitos para o sistema sendo construído ou melhorado. Itens de backlog podem incluir, por exemplo, funcionalidades, correção de erros, defeitos, requisições de melhorias, atualizações de tecnologia, etc. Questões que requeiram solução antes que outros itens de backlog sejam trabalhados também podem estar na lista.	95,30%	82,70%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Cliente presente	Em termos práticos, isso significa colocar o cliente fisicamente próximo aos desenvolvedores ou mover os desenvolvedores para próximo do cliente. Esta prática indica que o cliente deve fazer parte da equipe de desenvolvimento. Para esclarecer e validar requisitos e estabelecer prioridades, um representante do cliente trabalha junto da equipe todo o tempo. Assim o cliente pode responder a perguntas, resolver questões, estabelecer prioridades, fazer testes de aceitação e assegurar que o desenvolvimento tenha o progresso esperado. Quando surgem questionamentos, os programadores podem resolver imediatamente com o cliente, ao invés de tentar imaginar quais seriam suas preferências. Esta prática também leva o cliente a mudar mais prontamente os requisitos, ajudando a equipe a mudar o foco dos esforços de desenvolvimento para as necessidades mais prementes.	50,60%	37,30%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Desenvolvimento orientado a testes	Todo programador escreve casos de teste antes de escrever o código.Os testes devem ser escritos antes da implementação e conter o que for necessário para verificar se o código se comporta de acordo com os requisitos do usuário.O desenvolvedor ou testador deve escrever os casos de teste assim que os requisitos estiverem definidos.Além de escrever os testes de unidade antes da codificação, os desenvolvedores devem encorajar os usuários a escrever os testes de aceitação.Ao ser executado pela primeira vez, o caso de teste irá falhar, uma vez que o desenvolvimento do código que implementa a condição sendo testada ainda não foi gerado.Então, escreve-se código apenas o suficiente para o caso de teste passar, seguido de refatoração para remover duplicações e melhorar a legibilidade do código.Escrever drivers de teste antes da codificação força o desenvolvedor a pensar no problema antes da programação.Esta prática aplicada corretamente garante uma suíte de testes para fins de teste de regressão, além de prover documentação para o código implementado, servindo de casos de uso para este mesmo código.Deve-se ter o cliente escrevendo testes de aceitação, que podem se tornar casos de teste de um plano geral de testes para o sistema.Antes de o programador integrar o seu código à base de código, ele deve ter todos os seus próprios testes passando, além de todos aqueles testes já escritos e já incorporados à base, que também deverão passar.Isto assegura que o novo código sendo integrado para implementar uma nova funcionalidade não quebre o código de alguém que já havia sido incorporado à base de código.	59,30%	46,20%

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Design simples	A ênfase desta prática está em projetar a solução mais simples possível e que seja viável no momento. Complexidade desnecessária e código extra devem ser removidos assim que reconhecidos. Não se deve incluir aspectos adicionais aos artefatos sem uma boa justificativa para tal. A prática do design simples requer que a equipe não projete para satisfazer necessidades futuras, as quais os desenvolvedores não devem tentar prever. A abordagem de desenvolvimento mais efetiva em termos de custo deve focar em resolver os problemas de hoje ao invés de projetar para mudanças futuras que não se sabe se realmente ocorrerão. Deve-se trabalhar a solução mais simples que possa funcionar.	78,30%	62,90%
Equipe completa	Refere-se à prática de incluir todos os perfis e perspectivas necessários na equipe para que ela possa ter bom desempenho, enfatizando o espírito de equipe, com todos os seus membros compartilhando um propósito e apoiando-se mutuamente. Clientes, usuários e demais interessados devem ter um envolvimento direto no projeto, a fim de possibilitar entender o comportamento do sistema mais cedo no ciclo de vida.	51,00%	34,90%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Integração contínua	Na integração contínua, os membros da equipe devem integrar seu trabalho frequentemente, toda vez que novas mudanças ou uma tarefa for completada, para revelar problemas de integração e detectar falhas de sistema o mais cedo possível. Cada pessoa deve fazer a integração pelo menos uma vez por dia. Isto garante que sempre esteja disponível uma versão executável do sistema, contendo todas as novas funcionalidades e modificações, podendo servir de baseline para o trabalho. Depois da integração, todos os testes devem passar, ou o novo código deve ser descartado. Os integrantes da equipe podem fazer a integração sempre que desejarem, a não ser em curtos períodos de tempo em que o código é congelado. Cada integração deve ser verificada imediatamente ou à noite através de uma build automática com execução de todos os testes de integração. Esta prática é um exemplo de uma técnica dinâmica de garantia de qualidade.	92,10%	92,10%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Jogo de planejamento	Juntos desenvolvedores e clientes atuam no jogo de planejamento no qual o cliente escolhe as histórias de usuário que incluem os requisitos mais importantes a serem incluídos em uma entrega curta e incremental. Cada incremento curto implementado é experimentado pelo cliente. As histórias remanescentes são reavaliadas em termos de requisitos e prioridades, sendo o jogo de planejamento jogado novamente para definir o próximo incremento a ser implementado. A meta do jogo de planejamento é balancear os interesses do cliente com a capacidade da equipe. O planejamento é contínuo e progressivo. Os desenvolvedores estimam o custo das funcionalidades candidatas e o cliente as prioriza com base no custo e no valor agregado para o negócio. Uma das grandes vantagens do jogo de planejamento é a participação ativa do cliente e da equipe, com o processo de desenvolvimento sendo conhecido por todos. Diretrizes que levam a decisões relacionadas com liberações ou iterações específicas ficam claras para todos, pois cliente e equipe as definem juntos. Após as histórias de usuário terem sido definidas, a equipe de desenvolvimento fornece ao cliente uma estimativa de tempo para implementar cada uma delas. O cliente então prioriza as histórias considerando estas estimativas. Posteriormente a equipe informa ao cliente o tempo que irão trabalhar no próximo incremento e baseado nisso o cliente seleciona as histórias que a equipe irá implementar em seguida. Os desenvolvedores então dividem as histórias em tarefas, mas sem envolver o cliente com detalhes de implementação.	85,00%	70,70%

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Metáfora	Esta prática consiste em apresentar uma estória simples e compartilhada que explica a essência de como o sistema funciona, para dar a desenvolvedores e cliente um entendimento comum sobre o projeto. De um certo modo, a metáfora serve como uma arquitetura de alto nível para o software. A metáfora serve para fazer a ligação de um domínio conhecido com um domínio com o qual não se está familiarizado. Pensando sobre uma metáfora apropriada, os desenvolvedores podem expandir suas perspectivas de análise da aplicação sendo desenvolvida. Há dois propósitos principais para a metáfora. Um é a comunicação. A metáfora preenche a lacuna entre desenvolvedores e usuários assegurando facilidades na discussão sobre o software e no fornecimento de exemplos. O segundo propósito da metáfora é a contribuição para que a equipe desenvolva uma arquitetura apropriada para o software.	54,50%	34,60%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Padrões de código	Pelo fato de os desenvolvedores programarem diferentes partes do sistema com vários membros da equipe, a adoção de padrões de código é bastante interessante. Eles facilitam o entendimento do código, aumentam a legibilidade e melhoram a consistência entre membros da equipe. Os padrões devem ser fáceis de serem seguidos e devem ser adotados voluntariamente. Deve ser acordado pela equipe, assegurando que a comunicação possa ser feita via código, além de levar os desenvolvedores a entender facilmente o código de seus colegas. Esta prática libera o programador de tomar decisões com respeito a um estilo, torna mais fácil a adoção da prática de programação em par, além de apoiar a prática de propriedade coletiva de código.	78,30%	55,70%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Propriedade coletiva do código	O repositório de código deve ser de livre acesso para todos os programadores e permitir mudanças sempre que necessário. Uma vez na base de código, qualquer membro da equipe tem a posse sobre todo código. Todos são encorajados a fazer mudanças no código, em qualquer parte e a qualquer tempo que sintam a necessidade de fazê-las, sem ter que pedir permissão para quem quer que seja. Esta prática pode remover o gargalo de software que normalmente está relacionada com a posse individual do código. Qualquer par de programadores que veja uma oportunidade de adicionar valor a qualquer parte do código pode fazê-lo a qualquer tempo. A propriedade coletiva do código, além de ajudar na melhoria do código, dá mais flexibilidade aos programadores para se ausentar em casos de necessidade ou para saírem de férias. A disponibilidade de drivers de teste automatizados faz com que os desenvolvedores tenham mais liberdade para modificar o código sem maiores receios de eventuais repercussões que possam causar. Ao poder examinar o código escrito por outros, os programadores podem refletir e considerar as razões que levaram os colegas a tomar determinadas decisões específicas, ou podem melhorar o entendimento de seu próprio código e de suas interfaces com as demais partes da codificação do software.	71,50%	51,70%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Refatoração	O software se deteriora com o tempo. Um projeto que inicialmente parece enxuto e/ou perfeito pode torna-se progressivamente sobrecarregado e/ou deteriorado a cada modificação nele introduzida. Quando o software tem uma documentação mínima, o código fonte deve permanecer simples e fácil de entender. A refatoração é uma técnica disciplinada para se reestruturar um corpo de código existente ou para constantemente melhorar sua inteligibilidade e manutenibilidade, bem como o seu projeto, alterando sua estrutura interna sem mudar seu comportamento externo nem a funcionalidade do sistema. O foco é obter código simples, limpo e não repetitivo, que pode ser modificado facilmente. A essência da prática é uma série de pequenas transformações que preservam comportamento. Pelo fato das transformações serem pequenas, a possibilidade de algo dar errado é também pequena, com o sistema permanecendo totalmente funcional após cada refatoração. Durante a refatoração os desenvolvedores reconstroem o código e isto já provê inspeção da sua funcionalidade. As diferentes formas de refatoração podem envolver a simplificação de declarações complexas, a abstração de soluções comuns para fins de reuso e a remoção de código duplicado. Esta prática reduz a probabilidade de geração de erros durante o desenvolvimento. Entretanto, a prática de refatoração é altamente dependente do conjunto de casos de teste de unidade automatizados. Quando o código é refatorado, ele deve ainda passar por todos os casos de teste de unidade armazenados. Se algum caso de teste falhar	86,60%	80,20%

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Liberações frequentes (Releases curtas)	Esta prática visa maximizar o retorno dos projetos assegurando que o maior valor de negócio possível seja entregue ao final de cada release e que cada release tenha uma duração curta. Isso é feito através do processo contínuo de priorização que seleciona sempre as histórias de maior valor para serem implementadas primeiro. Além disso, procura antecipar o retorno entregando software rapidamente. Ciclos curtos reduzem os riscos, possibilitando ao cliente terminar rapidamente com projetos que não agreguem valor para o negócio. Além disso, ciclos de liberações frequentes ajudam a lidar com mudanças nos requisitos e reduzem o impacto de erros de planejamento. Ao final de cada release, o cliente revê todo o produto podendo identificar defeitos e fazer ajustes nos requisitos futuros.	92,10%	82,40%
Reuniões diárias	As reuniões diárias em pé são reuniões rápidas, geralmente de 15 minutos, organizadas para acompanhar o progresso do projeto, destacar questões importantes e organizar as atividades diárias. Cada membro da equipe relata rapidamente no que está trabalhando e o progresso já alcançado. Durante a reunião todos devem ficar de pé, para encorajar os participantes a serem objetivos, não ultrapassar o tempo previsto para a reunião, além de manter todos alertas e com atenção voltada para os assuntos tratados.	76,30%	57,70%

Continua na próxima página

Tabela 5.2 – Continuação da página anterior

Prática	Descrição	Pertinência	Relevância
Ritmo sustentável	Esta prática enfatiza que se deve trabalhar apenas a quantidade de horas que se possa manter produtividade de modo sustentável. Não trabalhar mais de 40 horas por semana é a regra, além de não mais de 8 horas por dia. Em períodos difíceis quando se trabalha além do tempo, os artefatos produzidos são pobres em qualidade. Os requisitos devem ser selecionados para cada iteração de modo que os desenvolvedores não precisem trabalhar fora de horário nem fazer horas-extras.	58,10%	42,30%
Visibilidade de projeto	Projetos ágeis por sua natureza estão continuamente mudando (planos, modelos, código e demais artefatos). Deve haver esforços para fornecer às equipes o status do projeto em que estão engajadas. A psicologia mostra que quanto mais imediata a retroalimentação, mais rapidamente as pessoas mudam o comportamento para se adequar a novas situações. Pode ser criado um painel na web para manter a qualquer tempo o status e as métricas relacionadas com o progresso do projeto. Múltiplos painéis podem ser usados para disponibilizar diferentes tipos de informação que atendam a todos os níveis organizacionais necessários. O avanço do projeto com relação às histórias de usuário que as equipes se comprometeram a entregar no final das iterações deve ser incluído. Modelos devem se tornar acessíveis para todas as equipes.	88,90%	75,80%

5.4 Lista das Etapas de Teste de Software

Tabela 5.3: Lista das Etapas de Teste de Software

Etapa de Teste	**Descrição Resumida**	**Produto Trabalho/Produzido**
Planejar Testes	Envolve o planejamento do processo de teste a ser seguido para um projeto específico, com estimativa de,custos, cronograma e recursos; são ainda definidos os itens a serem testados, as estratégias, métodos e técnicas de teste a serem adotadas, bem como é estabelecido um,critério para aceitação do software testado.	Plano de Teste
Projetar Testes	Esta atividade envolve a especificação detalhada das abordagens a serem seguidas durante a realização dos testes identificadas na atividade "Planejar Testes", para avaliar os itens de teste nela identificados. Nesta atividade são identificados conjuntos de casos e procedimentos de teste a serem executados para avaliação do software.	Especificação do Projeto de Teste

Continua na próxima página

Tabela 5.3 – Continuação da página anterior

Etapa de Teste	Descrição Resumida	Produto Trabalho/Produzido
Especificar Casos de Teste	Nessa atividade, devem ser especificados todos os casos de teste identificados na atividade anterior (Projetar Testes). Para cada caso de teste devem ser descritos os seus valores de entrada, resultados esperados, recursos necessários para a sua execução, suas restrições e dependências com outros casos de teste.	Especificação de Casos de Teste
Definir Procedimentos de Teste	Deve definir procedimentos descrevendo os passos necessários para a execução de um ou de um grupo de casos de teste. Um procedimento de teste precisa conter informações sobre o seu objetivo, requisitos para a sua execução, além dos passos a serem seguidos durante os testes.	Especificação de Procedimentos de Teste
Executar Testes	Esta atividade envolve executar os procedimentos de teste elaborados para um produto específico, devendo a execução ser devidamente documentada para permitir que outros testadores possam repeti-la de forma semelhante.	Histórico dos Testes, Relatório de Incidentes de Teste

Continua na próxima página

Tabela 5.3 – Continuação da página anterior

Etapa de Teste	Descrição Resumida	Produto Trabalho/Produzido
Analisar Resultados	Esta atividade envolve avaliar os resultados dos testes para saber se eles obtiveram sucesso. Na maioria das vezes, "sucesso" significa que o sistema funcionou conforme o planejado, e não apresentou resultados diferentes dos resultados esperados, conforme descrito nos casos de teste. Esta atividade também pode envolver a utilização de métricas de teste específicas, calculadas a partir dos resultados alcançados.	Relatório de Resumo dos Testes
Monitorar e Controlar o Processo de Teste	As atividades de monitoramento, controle e re-planejamento de testes tem como propósito manter o controle e fazer as correções necessárias no Plano de Teste,quando este não mais refletir a realidade na medida em que as atividades de teste são executadas e os testes progridem.	Registro das tarefas executadas, do andamento do processo e dos resultados obtidos para os testes

Continua na próxima página

Tabela 5.3 – Continuação da página anterior

Etapa de Teste	Descrição Resumida	Produto Trabalho/Produzido
Fechar Atividades de Teste	A atividade de fechamento do processo de teste tem como propósito descartar artefatos desnecessários após o término dos testes e guardar ativos físicos de valor, bem como informações que sejam relevantes para a organização. Os dados gerados ao longo dos testes são armazenados para permitir a qualquer momento a consulta a esses dados, como uma forma de apoio durante a realização de novas atividades de teste ou auditoria de execução dos testes	Registro de dados de execução e de resultados de teste, disponibilizados como fonte de consulta para apoiar planejamento de futuras instúncias de processos de teste.