**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**
**CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**MAICON GEAN GAVA**

# SISTEMA PARA DISTRIBUIÇÃO E CONTROLE DE SOLICITAÇÕES DE MODIFICAÇÕES EM SOFTWARE

**TRABALHO DE CONCLUSÃO DE CURSO 2**

**PATO BRANCO**
**2016**

**MAICON GEAN GAVA**

# SISTEMA PARA DISTRIBUIÇÃO E CONTROLE DE SOLICITAÇÕES DE MODIFICAÇÕES EM SOFTWARE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

**PATO BRANCO**
**2016**

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e
Desenvolvimento de Sistemas

# TERMO DE APROVAÇÃO

## TRABALHO DE CONCLUSÃO DE CURSO

## SISTEMA PARA DISTRIBUIÇÃO E CONTROLE DE SOLICITAÇÕES DE MODIFICAÇÕES EM SOFTWARE

**por**

**MAICON GEAN GAVA**

Este trabalho de conclusão de curso foi apresentado no dia 15 de agosto de 2016, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

**Banca examinadora:**

Profª. Drª. Beatriz Terezinha Borsoi
Orientador

Profª. Me. Andreia Scariot Beulke

Profª. Me. Eliane Maria de Bortoli Fávero

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª. Me. Soelaine Rodrigues Ascari
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

## RESUMO

GAVA, Maicon Gean. Sistema para distribuição e controle de solicitações de modificações em software. 2016. 70f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

É indiscutível a importância do acompanhamento, gerência e controle na distribuição de qualquer solicitação que visa modificar um software, independentemente da sua urgência ou complexidade. Esse acompanhamento é utilizado para a gerência de prazos e recursos envolvidos e para acompanhamento do trabalho sendo realizado. A empresa desenvolvedora e responsável pela manutenção do software deve responsabilizar-se em garantir que esse controle seja realizado da forma mais efetiva possível para que prazos e orçamento sejam respeitados. Para que isso possa ser realizado é importante contar com o auxílio de um sistema que faça esse controle específico para que possa ser utilizado na empresa por todas as pessoas envolvidas nessas solicitações de manutenção. Visando contribuir para o controle das solicitações de manutenção em software, neste trabalho é proposto o desenvolvimento de um software que visa auxiliar no controle e na distribuição dessas solicitações para empresas desenvolvedoras de software. Esse software visa otimizar pontos importantes no ciclo de vida das solicitações de manutenção, como a organização do trabalho, a distribuição dessas solicitações para os profissionais, a velocidade no desenvolvimento, a qualidade do resultado e a manutenção do histórico.

**Palavras-chave**: Solicitações de software. Controle de requisitos. Modificações em software. Distribuição de tarefas de desenvolvimento.

## ABSTRACT

GAVA, Maicon Gean. System for distribution and control of software modifications requests. 2016. 70f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2016.

It's unquestionable the importance of control and distribution of any request which aims to modify a software, regardless of the urgency or complexity. The company who is developer and responsible of the software maintenance shall be responsible for ensuring that this control be realized in the best possible way. To this end, it is highly recommended that a system with this particular control is installed and applied in the company. For this reason, it was viable in this work develop a software to help the control and distribution of these requests to software development companies. It will be a software that optimizes several important points, such as task organization, development speed, product quality, history, among others.



**Keywords**: Software requests. Requirements control. Software modifications. Development tasks distribution.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTAGENS DE CÓDIGO

## LISTA DE SIGLAS

| | |
|---|---|
| CRUD | *Create, Read, Update and Delete* |
| DAO | *Data Access Object* |
| JPA | *Java Persistence API* |
| MVC | *Model-View-Controller* |
| UML | *Unified Modeling language* |
| XML | *Extensible Markup Language* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa de realização do trabalho. Por fim está a apresentação dos capítulos subsequentes.

## 1.1 CONSIDERAÇÕES INICIAIS

A manutenção de software é caracterizada pelas modificações realizadas em produtos de software que são entregues ao cliente (INSTITUTE..., 1998). Essas modificações estão relacionadas aos erros, às melhorias em desempenho ou qualquer outro atributo, como inclusões de funcionalidades e adaptações para atender mudanças em termos de tecnologias, regras e conceitos de negócio e requisitos de legislação.

Pfleeger (2004) classifica os sistemas em três categorias considerando necessidades e tipos de manutenções realizadas:

a) Software construído com base em especificação muito bem definida e rígida em termos de alterações que possa sofrer e que possuem os resultados esperados bem conhecidos. Esses são os aplicativos para realizar operações com matrizes e técnicas de grafos e árvores, por exemplo, ou mesmo algoritmos de busca e ordenação, de reconhecimento de padrões em imagens e que possuem uma forte conceituação matemática. Uma vez que tenham sido construídos, devidamente testados e considerada correta a implementação do método, não haverá necessidade de manutenção.

b) Software como resultado de implementações aproximadas para problemas reais. Exemplos dessas soluções são aquelas para problemas conhecidos por terem uma solução difícil para a ciência da computação, com aplicações como logísticas de transporte e distribuição, jogos como xadrez (impossibilidade de calcular todos os próximos passos possíveis pela explosão combinatória de possibilidades, por exemplo), entre outros. Nesses casos, as soluções completas podem ser obtidas somente na teoria e as implementações apresentam soluções que podem ser aproximadas da melhor solução.

c) Software que considera mudanças no ambiente de utilização. São os aplicativos sujeitos às mudanças em tempo de execução como os de provisões

econômicas, bolsas de valores, investimentos, monitoramento e controle de eventos ambientais e na área de medicina para tratamento que dependem da evolução da doença.

Além das categorias indicadas por Pfleeger (2004) estão as manutenções decorrentes de ajustes e correções resultantes de erros e falhas nos sistemas e de inclusão e melhorias de funcionalidades no software. Essas podem ser comuns especialmente para aplicativos desenvolvidos sob demanda, que possuam contrato de manutenção ou que são personalizados para clientes específicos ou categorias de clientes.

Nas empresas de desenvolvimento de software que realizam manutenções corretivas, preventivas, preditivas e outras, é importante que a implementação seja realizada com rapidez e segurança. A rapidez está relacionada ao menor tempo para que a solicitação de manutenção seja implementada, ocasionando redução de custos para empresa e rápido atendimento ao cliente. E a segurança vincula-se à qualidade da manutenção realizada para que seja assegurada a não existência de erros e inconsistências com outras funcionalidades já implementadas no sistema. Assegurar esses dois requisitos está, entre outros, relacionado à escolha do profissional mais adequado para realizar a manutenção. A adequação está relacionada ao conhecimento e à experiência do profissional nas tecnologias utilizadas e tipo de tarefa a ser realizada no tempo disponível que esse profissional possui.

Considerando esse contexto e nele inserindo a possibilidade de ferramentas computacionais que possam auxiliar na escolha do profissional mais indicado para implementar determinada solicitação de manutenção, este trabalho propõe o desenvolvimento de um aplicativo que visa auxiliar na realização dessa tarefa de controle do atendimento de solicitações de manutenção. O aplicativo terá como principal vantagem auxiliar a indicar, a partir de informações armazenadas sobre os profissionais (desenvolvedores e testadores) e requisitos da solicitação de manutenção, o programador mais adequado para implementar a referida solicitação.

## 1.2 OBJETIVOS

A seguir são apresentados os objetivos pretendidos com a realização deste trabalho.

### 1.2.1 Objetivo Geral

Desenvolver um aplicativo para auxiliar na distribuição de solicitações de modificações em software.

### 1.2.2 Objetivos Específicos

- Auxiliar no gerenciamento de atendimentos de solicitações de alterações e inclusões em software;
- Proporcionar o controle informatizado de solicitações de manutenção de software, permitindo acompanhar o estado de cada solicitação e o tempo de implementação e testes;
- Facilitar a distribuição de solicitações de manutenção de software para os membros de uma equipe de desenvolvimento, visando otimizar o tempo e o custo e maximizar a qualidade da implementação e dos testes realizados.

## 1.3 JUSTIFICATIVA

O desenvolvimento de um aplicativo computacional que visa auxiliar o controle de solicitações de software, e principalmente a distribuição mais adequada de solicitações de manutenção se justifica pela necessidade de reduzir o tempo de implementação e de aumentar a qualidade dessa implementação. Existe a possibilidade do programador ou testador mais adequado não estarem disponíveis no momento em que a solicitação ocorre. Assim, a adequação está em que a

solicitação seja atribuída ao profissional que implementará de forma mais rápida e eficiente a solução para o problema, considerando a criticidade da implementação, a sua urgência e os conhecimentos necessários das tecnologias e das regras de negócio envolvidas.

Para que se possa realizar a distribuição de solicitações de manutenção considerando outros critérios que não exclusivamente o fato de o programador ou testador estar disponível, é importante considerar o auxílio de um aplicativo computacional. Dessa forma, dados como habilidades, conhecimento e experiências podem ser armazenados e utilizados como critérios para indicação do profissional mais capacitado para resolver determinado problema juntamente com sua disponibilidade. Esses critérios não serão avaliados em conjunto, mas paralelamente, delegando a decisão ao gerente de projetos, analista ou outro papel incumbido da decisão.

O aplicativo instrumenta o decisor no processo de tomada de decisão para a escolha do profissional mais adequado para cada demanda. A decisão, também estará vinculada à quantidade, qualidade e atualidade dos dados tanto dos profissionais quanto da demanda que serão utilizados para gerar o *ranking* (posição ordenada) de profissionais de acordo com as características da demanda.

## 1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. Este é o primeiro e apresenta as considerações iniciais com o contexto do sistema que foi desenvolvido, os objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico centrado em manutenção de software. No Capítulo 3 estão as ferramentas e tecnologias utilizadas na modelagem do sistema e que foram utilizadas na implementação subsequente do sistema. No Capítulo 4 é apresentado o resultado da realização do trabalho, ou seja, a modelagem e a implementação do sistema. Por fim estão as considerações finais seguidas pelas referências.

# 2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho que está relacionado à manutenção de software.

## 2.1 MANUTENÇÃO DE SOFTWARE

O sucesso de um software está vinculado ao atendimento qualitativo, na fase de desenvolvimento, das necessidades do usuário e do seu negócio, levando em consideração prazos e custos adequados aos requisitos desenvolvidos (ARAKAKI; SOUZA, 2006) e atendendo padrões de qualidade.

Mudanças em regras de negócio, necessidade de incremento de funcionalidades e ajustes em decorrência de erros são comuns em software e determinam a necessidade de manutenção desses produtos. Linger e Moore (2001) ressaltam que o processo de manutenção de software mal realizado em um sistema em produção pode levar à sua indisponibilidade temporária, proporcionando prejuízos de negócio e a redução do tempo de vida do software devido à inserção de novos defeitos. Com o intuito de minimizar esses impactos, Fiutem *et al*. (1996) propôs que a cada atividade de manutenção de um sistema é necessário identificar os elementos de software que a compõem e os seus relacionamentos.

A manutenibilidade de software pode ser definida qualitativamente como a facilidade com que um software pode ser entendido, corrigido, adaptado e/ou aumentado (PRESSMAN, 2006). A manutenção de sistemas está relacionada à realização de atividades para atualizar a corrigir um software já liberado para o cliente. A necessidade de manutenção ocorre pela identificação de problemas de funcionamento ou desempenho do software, inclusão ou ajuste de funcionalidades que está em uso, em produção.

Sommerville (2007) destaca que a manutenção de software engloba três atividades: manutenção corretiva (reparo de defeitos no software), manutenção adaptativa (adaptação do software a um ambiente operacional diferente), manutenção evolutiva (manutenção para adicionar funcionalidades ao software ou modificá-la).

Pressman (2006) acrescenta a manutenção preventiva ou reengenharia. Os tipos de manutenção definidos por Pressman podem ser estruturados como apresentado pela Figura 1, que exemplifica esta estrutura a partir de uma requisição de modificação, que é desdobrada em correção e melhoria e essas são divididas nas quatro categorias de modificação: corretiva, preventiva, adaptativa e perfectiva.

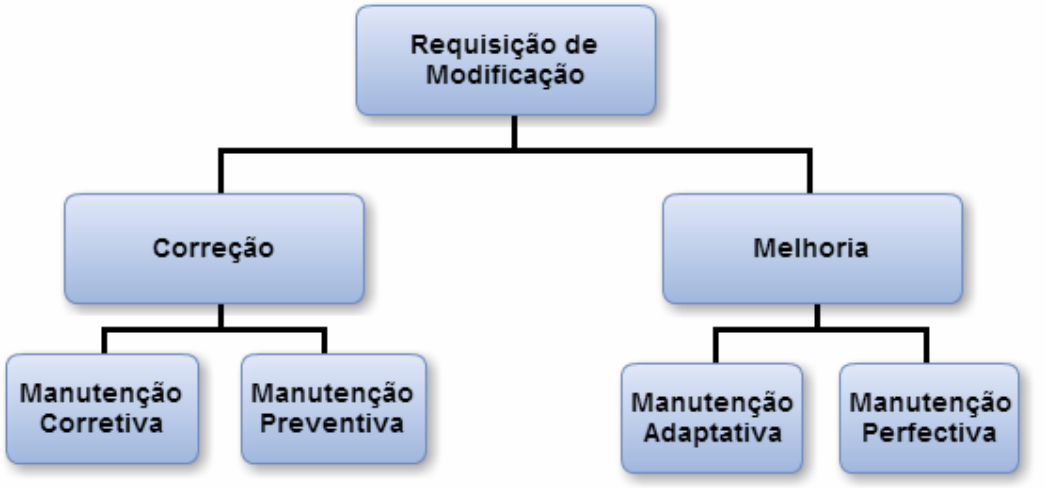


**Figura 1 – Categorização da manutenção de software**
**Fonte: composto a partir de Pressman (2006).**

Pressman (2006) adota a mesma classificação do padrão IEEE 1219 (IEEE, 1999). Outros autores como Calazans e Oliveira (2005) também utilizam essa classificação:

a) Manutenções adaptativas - representam as manutenções executadas com o objetivo de adaptar o software às mudanças do ambiente, como por exemplo, alterações nas configurações de hardware e de sistema operacional.

b) Manutenções corretivas - contemplam as mudanças acarretadas por falhas no processamento das funcionalidades ou devido a ausência de atributos especificados, como por exemplo, desempenho ou segurança e que comprometem o uso do sistema. Estão, geralmente, relacionadas a erros do sistema, sejam de lógica, codificação, projeto ou outros (SILVA *et al.*, 2010).

c) Manutenções perfectivas - envolvem as manutenções responsáveis por inserir ou modificar os requisitos funcionais e não funcionais do software, assim como as manutenções executadas para melhorar a manutenibilidade do sistema, reduzindo o esforço necessário para realização das manutenções futuras. Nessa categoria incluem-se as alterações que almejam melhorar a compreensão sobre o software, reduzir a complexidade do código ou eliminar a degradação inserida por

outras atividades de manutenção. Esse tipo de manutenção é conhecida, também, como evolutiva, no sentido que o software tenha seus requisitos funcionais aprimorados, partir de observação e participação dos desenvolvedores do software e dos usuários, melhorando a eficiência do sistema (SILVA *et al*., 2010).

d) Manutenções preventivas - abrangem as correções diagnosticadas e concluídas antes que se tornem um problema real. Esse tipo de manutenção é parte de uma política de gerenciamento pró-ativo da fase de manutenção. A manutenção preventiva caracteriza modificações que deterioram o software quanto a correção, adaptação ou melhoria ao longo do ciclo de vida (SILVA *et al*., 2010). É conhecida, também, como reengenharia e está relacionada às alterações no software buscando melhorar a confiabilidade ou oferecer uma estrutura melhor para futuras manutenções.

Engholm Junior (2013), com base nas classificações de manutenção cita que as manutenções estão associadas à: remoção de defeitos; adição de funcionalidades no sistema; alteração de funcionalidades; evolução das funcionalidades; ajustes; atualização e melhora de desempenho.

A norma ABNT NBR ISO/IEC 12207 (ASSOCIAÇÃO..., 2009, p. 53) define um processo de manutenção de software caracterizando-o como processo técnico e com o propósito de fornecer suporte com boa relação custo-benefício ao produto de software entregue.

A manutenção está associada com a fase de pós-entrega do software, quando o mesmo já está em uso no cliente. Webster, Oliveira e Anquetil (2005), mencionam que durante a fase de desenvolvimento inicial os processos são governados pelos requisitos elucidados e na fase de manutenção os processos são dirigidos por eventos, representados pelas solicitações de manutenção registradas pelos usuários. Consequentemente, a equipe de manutenção e os usuários passam a usufruir de outros mecanismos para elucidar, negociar, planejar, executar e validar as solicitações de manutenção. Mens et al. (2005) apontam o gerenciamento e a evolução desse conjunto de mecanismos como um dos principais desafios para o desenvolvimento da manutenção de software.

A fase de manutenção possui outras características que representam riscos aos projetos e influenciam diretamente a especificação dos processos executados nessa fase do ciclo de vida de software. Conforme relatam Basili *et al*. (1996), durante a fase de manutenção os colaboradores gastam a maior parte dotempo

isolando e analisando as solicitações que efetivamente implementando as modificações. Os produtos de software oferecidos a uma grande quantidade de clientes também tendem a ter ciclos de manutenção específicos. Rajlich e Bennett (2000) salientam que nesse contexto, a manutenção rotineira dos sistemas é muitas vezes inexequível, devido à ampla base de clientes.

Atividades de manutenção de software são caracterizadas por intervenções no produto de software de forma a evitar sua deterioração, no sentido de os objetivos de suas funcionalidades deixarem de se adequar ao ambiente externo (PADUELLI, 2007).

# 3 MATERIAIS E MÉTODO

A seguir estão os materiais e o método utilizados para a modelagem e a implementação do sistema obtido como resultado deste trabalho.

## 3.1 MATERIAIS

O Quadro 1 apresenta as tecnologias e as ferramentas utilizadas no desenvolvimento do trabalho.

| Ferramenta | Versão | Link | Funcionalidade |
|---|---|---|---|
| Visual Paradigm | 12.2 | http://www.visual-paradigm.com/ | Modelagem do diagrama de casos de uso e do diagrama de entidades e relacionamentos do banco de dados |
| Microsoft Visio | 2003 | https://products.office.com/pt-br/visio/flowchart-software | Modelagem do fluxo principal do processo |
| NetBeans | 8.0.2 | https://netbeans.org/ | Ambiente integrado de desenvolvimento. |
| Java | 7.0 | http://www.oracle.com | Linguagem de programação |
| Java Persistence API (JPA) | 2.0 | http://www.oracle.com | Modelo de persistência para mapeamento de objeto relacional. |
| PostgresSQL | 9.4 | http://www.postgresql.org/ | Banco de dados. |

**Quadro 1 – Tecnologias e ferramentas utilizadas**

## 3.2 MÉTODO

O método consiste nas atividades de levantamento de requisitos, análise, projeto, e desenvolvimento. A seguir estão descritas, de forma sucinta, as etapas desenvolvidas para a realização deste trabalho.

**Levantamento de requisitos**

O levantamento de requisitos foi realizado com base nas necessidades percebidas no cotidiano profissional do autor deste trabalho que desempenha a função de atendimento a clientes em uma empresa de desenvolvimento de software.

Essa empresa desenvolve software sob demanda e personaliza soluções para clientes.

Para o levantamento dos requisitos foi analisado o processo envolvido no atendimento das solicitações de mudança que ocorrem dentro da empresa, desde o pedido do cliente até o atendimento da solicitação pelos desenvolvedores e a realização dos testes.

A observação do processo, agregada pela opinião das pessoas que realizam atividade de atendimento a clientes na empresa, auxiliou a definir os requisitos do sistema, que foram organizados em funcionais e não funcionais.

**Análise e projeto**

Os requisitos levantados foram modelados sob a forma de casos de uso e de diagrama de entidades e relacionamentos do banco de dados. Assim, foi definida uma visão mais ampla e clara do sistema.

**Desenvolvimento**

A implementação foi realizada utilizando a linguagem Java e testes de código foram realizados à medida que as funcionalidades foram implementadas. Os testes visaram identificar erros de implementação, validações de campos, consistência com dados armazenados e excluídos e o atendimento das funcionalidades definidas.

# 4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho. O resultado é o desenvolvimento de um sistema para acompanhamento de solicitações de modificações em software.

## 4.1 ESCOPO DO SISTEMA

O aplicativo descrito neste trabalho é um sistema *desktop* implementado com a linguagem Java. Esse aplicativo tem como principal objetivo gerenciar o acompanhamento da implementação de solicitações de modificações em um software realizadas por uma equipe de desenvolvimento. O aplicativo está centrado no cadastro, na distribuição e no controle dessas solicitações. Essas tarefas são realizadas pelos funcionários, considerados usuários do sistema.

Inicialmente, o administrador realizará todos os cadastros gerais, como funcionários, níveis, áreas de conhecimento, entre outros.

Após o sistema estar estruturado e com todos os cadastros necessários, analistas poderão incluir solicitações, seja de um cliente, funcionário ou mediante análise própria, quando o desenvolvimento de algum item relacionado a um produto de software da empresa seja considerado necessário.

A principal funcionalidade do aplicativo está na análise inicial da solicitação. O aplicativo fornecerá uma espécie de classificação dos funcionários de acordo com requisitos definidos, visando auxiliar o analista na distribuição das solicitações para as pessoas da equipe, possibilitando uma melhor divisão de tarefas. Os funcionários serão categorizados em níveis em cada área de conhecimento requerida na solicitação e em termos de disponibilidade de tempo. Esses índices serão baseados em cadastros prévios definidos no perfil de cada funcionário.

Todas as informações modificadas nas solicitações pelos funcionários são importantes para o bom funcionamento do sistema, organização e atualização de informações para os interessados.

Após a definição dos funcionários relacionados à solicitação, elas ficarão disponíveis na tela de solicitações e cada funcionário é responsável por filtrar as tarefas de sua responsabilidade ou seu interesse e modificá-las, caso necessário.

Todo o processo será acompanhado pelo analista e por outros interessados, exceto o cliente. Neste caso, algum responsável manterá o cliente informado do andamento da solicitação.

A rotina de trabalho consiste, principalmente, em modificar as situações das solicitações e adicionar comentários para manter o processo de desenvolvimento documentado e atualizado. Ao serem adicionados comentários será, automaticamente, enviada mensagem eletrônica via *e-mail* para os funcionários envolvidos na referida solicitação para que todos tenham conhecimento do andamento do processo.

Todos esses controles visam uma melhor organização do processo e fornecer facilidade de uso por parte da equipe, auxiliando na análise, desenvolvimento e testes das solicitações.

## 4.2 MODELAGEM DO SISTEMA

Esta seção apresenta os requisitos definidos para o sistema e os diagramas usados para detalhar os processos e a estrutura do software desenvolvido.

A Figura 2 apresenta uma visão geral do fluxo da funcionalidade principal do sistema que é a avaliação para a distribuição de solicitações (requisições para manutenções) provenientes de clientes externos. A solicitação, como descrita pelo usuário, é recebida pelo suporte, que a partir dessa descrição elabora uma descrição técnica, no sentido de o problema ser descrito de maneira que possa ser melhor compreendido pela equipe técnica (analista, desenvolvedor, testador). A distribuição da solicitação é realizada por meio de análise do perfil técnico do analista, desenvolvedor ou testador e da disponibilidade de tempo que o mesmo possui.

A parte inferior da Figura 2 apresenta a descrição do processo contido na atividade “Escolha desenvolvedor/testador” da parte superior dessa figura.

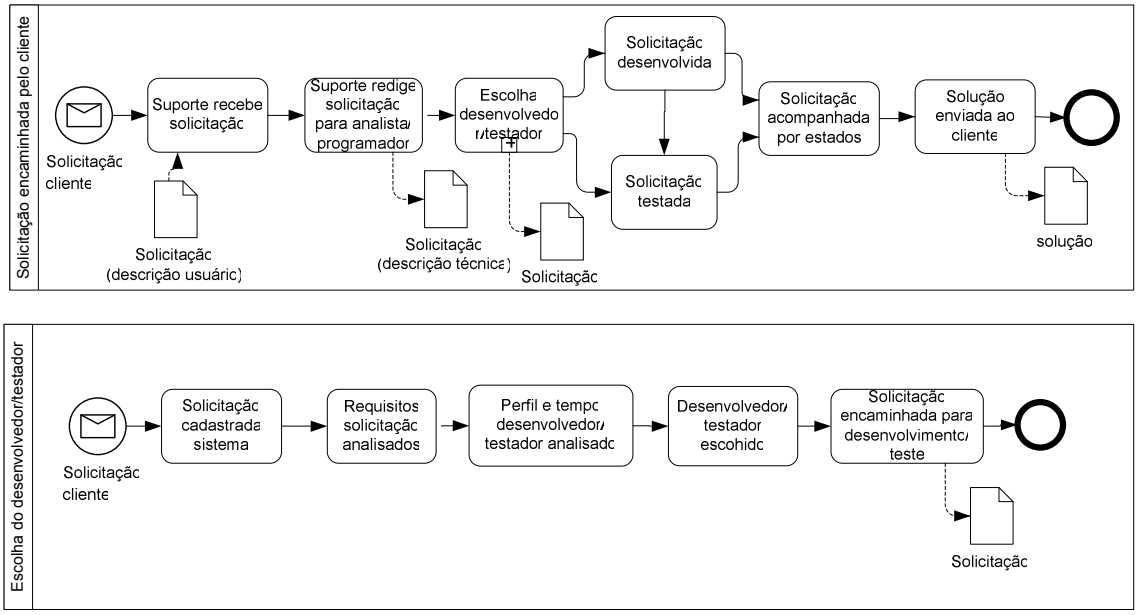


**Figura 2 – Fluxo da funcionalidade principal do sistema**

O levantamento de requisitos está categorizado em requisitos funcionais, que representam funções do sistema que serão realizados pelos atores e requisitos não funcionais, que são comportamentos automáticos do sistema visando à qualidade do seu funcionamento.

Nos quadros a seguir os requisitos funcionais estão representados com a sigla “RF” e os requisitos não funcionais com a sigla “RNF”, cada um com seu nome e descrição. Define-se manutenção como sendo as modificações realizadas em um software já em uso por clientes. Essas manutenções podem referir-se a consertos de erros, acréscimos, complementos ou ajustes de funcionalidades, atendimento a legislação, entre outros. No Quadro 1 estão os requisitos funcionais do sistema.

| **Identificação** | **Nome** | **Descrição** |
|---|---|---|
| RF1 | Cadastrar funcionário | Manutenção de funcionários com todas suas especificações e, também, *login* e senha para acesso. |
| RF2 | Cadastrar áreas de conhecimento | Manutenção das áreas de conhecimento necessárias para o desenvolvimento das solicitações. As áreas de conhecimento são utilizadas, juntamente com os níveis, para compor o perfil do funcionário. Esse perfil, juntamente com a disponibilidade de tempo, é utilizado para auxiliar na distribuição das solicitações entre os desenvolvedores. |
| RF3 | Cadastrar níveis | Manutenção dos níveis de experiência para classificar as áreas de conhecimento de cada funcionário. |
| RF4 | Cadastrar produtos | Manutenção de produtos desenvolvidos pela empresa. |
| RF5 | Cadastrar versões | Manutenção das versões dos produtos. |
| RF6 | Cadastrar clientes | Manutenção de clientes. |
| RF7 | Cadastrar | Manutenção das cidades dos clientes e funcionários. |

| | | |
|---|---|---|
| | cidades | |
| RF8 | Cadastrar tipos de solicitações | Manutenção de tipos de solicitações (Correção, sugestão, entre outros) |
| RF9 | Cadastrar solicitações | Manutenção do cadastro das solicitações de modificação, com suas respectivas especificações. |
| RF10 | Emitir Relatórios | Emissão de relatórios pré-definidos, com as informações provenientes do banco de dados. |

**Quadro 2 – Requisitos funcionais**

No Quadro 3 estão os requisitos não funcionais do sistema e estão relacionados ao acesso ao sistema, a categorização dos funcionários para a distribuição das solicitações e a interface do sistema.

| **Identificação** | **Nome** | **Descrição** |
|---|---|---|
| RNF1 | Efetuar login | O sistema validará *login* e senha para conceder acesso ao sistema. |
| RNF2 | Enviar e-mail | O sistema enviará automaticamente um e-mail para todos os funcionários interessados nas solicitações a cada comentário inserido. |
| RNF3 | Categorizar funcionários | O sistema irá categorizar os funcionários em níveis de áreas de conhecimento e tempo disponível, em uma lista decrescente de acordo com os níveis constantes no cadastro de perfil de cada funcionário. |
| RNF4 | Interface | O sistema irá operar em uma interface *desktop*. |

**Quadro 3 – Requisitos não funcionais**

A Figura 3 representa o diagrama de casos de uso do sistema desenvolvido utilizando o software Visual Paradigm. Neste diagrama, o único ator é apresentado com suas atividades específicas (casos de uso) que representam funcionalidades do sistema.

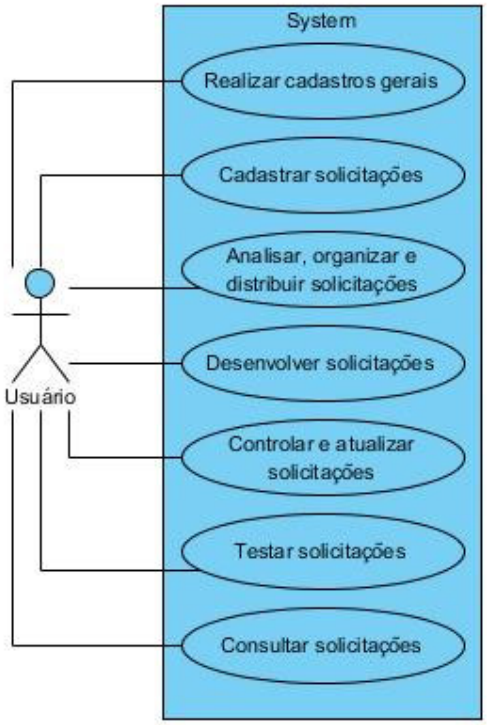


**Figura 3 – Diagrama de casos de uso**

Os Quadros 4 a 7 apresentam a descrição do caso de uso "realizar cadastros gerais" (cidades, cargos, produtos, versões, áreas de conhecimento, níveis e tipos de solicitação) e "cadastrar solicitações". A descrição está organizada por funcionalidade desses dois casos de uso e pela funcionalidade de consulta do caso de uso "consultar solicitações". A operação de inclusão está descrita no Quadro 4.

**Caso de uso:**
Incluir (operação de inclusão de registro).
**Descrição**:
Inclusão dos dados cadastrais de um registro no sistema.
**Evento Iniciador**:
O usuário solicita a inclusão de um registro no sistema.
**Atores**:
Todos.
**Pré-condição**:
Não há.
**Sequência de Eventos**:
1. Ator acessa a tela para cadastro incluindo as informações necessárias.
2. O sistema insere os dados no banco de dados e informa o usuário que o referido cadastro foi realizado com sucesso.
**Pós-Condição**:
Registro inserido no banco de dados.
**Extensões**:
Informações provenientes de outros cadastros como, por exemplo, cidade não cadastrada.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| 1 Cadastro de cidades. | 1.1 O ator acessa a tela do sistema para cadastrar cidades e incluir as informações solicitadas.<br>1.2 O sistema inclui informações no banco de dados. |

**Quadro 4 – Operação "incluir" dos casos de uso de cadastro**

No Quadro 5 é apresentada a expansão da operação excluir dos casos de uso de cadastro.

**Caso de uso:**
Excluir (operação de exclusão de registros).
**Descrição**:
Exclusão de dados cadastrais de um registro no sistema.
**Evento Iniciador**:
O usuário solicita a exclusão de um registro no sistema.
**Atores**:
Todos.
**Pré-condição**:
Não há.
**Sequência de Eventos**:
1. Ator acessa a tela para exclusão de cadastros.
2. Ator seleciona registro a ser excluído.
3. O sistema exclui o registro indicado e informa ao usuário que a referida exclusão foi realizada com sucesso.
**Pós-Condição**:
Registro excluído no banco de dados.
**Extensões**:
Registro sendo utilizado em outros cadastros.

| Nome do fluxo alternativo (extensão) | Descrição |
|---|---|
| 1 Exclusão de registro com vínculos | 1.1 O sistema informa que o registro não pode ser excluído porque possui registros vinculados. |

**Quadro 5 – Operação "excluir" dos casos de uso de cadastro**

A operação de atualização de dados já cadastrados é apresentada no Quadro 6 como a expansão do respectivo caso de uso.

**Caso de uso:**
Atualizar (operação de atualização de dados).
**Descrição**:
Atualização dos dados cadastrais de um registro no sistema.
**Evento Iniciador**:
O usuário solicita a atualização de um registro no sistema.
**Atores**:
Todos.
**Pré-condição**:
Não há.
**Sequência de Eventos**:
1. Ator acessa a tela para visualizar o conteúdo de um registro.
2. O sistema apresenta o registro selecionado para edição.

3. O usuário altera os dados e solicita alteração do registro.
4. O sistema inclui os dados alterados e informa ao usuário que a operação foi realizada conforme solicitado.

**Pós-Condição**:
Dados do registro alterados no banco de dados.

**Quadro 6 – Operação "atualizar" dos casos de uso de cadastro**

No Quadro 7 é apresentada a operação de consulta de dados de cadastros, como expansão de caso de uso.

**Caso de uso:**
Consultar (operação de consulta de dados).
**Descrição**:
Consulta dos dados cadastrais de um registro no sistema.
**Evento Iniciador**:
O usuário solicita a consulta de um registro no sistema.
**Atores**:
Todos.
**Pré-condição**:
Não há.
**Sequência de Eventos**:
1. Ator acessa a tela para consultar o conteúdo de um registro.
2. O ator indica o parâmetro de consulta.
3. O sistema apresenta os resultados da consulta.

**Pós-Condição**:
Dados da consulta apresentados ao usuário.

**Quadro 7 – Operação "consultar" dos casos de uso de cadastro**

A seguir, nos Quadros 8 a 11, está a descrição dos casos e uso que não se referem especificamente operações de *Create, Read, Update and Delete (*CRUD). No Quadro 8 é apresentada a expansão do caso de uso que se refere à realização de testes em solicitações.

**Caso de uso:**
Testar solicitações.
**Descrição**:
Teste de uma solicitação cadastrada no sistema.
**Evento Iniciador**:
Uma solicitação é desenvolvida e fica na fila para testes do sistema.
**Atores**:
Testador.
**Pré-condição**:
1. Necessita de uma solicitação desenvolvida.
**Sequência de Eventos**:
1. O ator consulta as solicitações desenvolvidas, sua descrição, e seu funcionamento previsto.
2. O ator testa, verifica se está funcionando como esperado e atualiza o registro da solicitação com seus resultados.

**Pós-Condição**:
O cadastro da solicitação é atualizado no sistema com os resultados dos testes.

**Quadro 8 – Caso de uso testar solicitações**

No Quadro 9 é apresentada a expansão do caso de uso desenvolver solicitações, como expansão de caso de uso.

**Caso de uso:**
Desenvolver solicitações.
**Descrição**:
Desenvolvimento de uma solicitação pendente no sistema.
**Evento Iniciador**:
Uma solicitação é encaminhada pelo analista e fica na fila para desenvolvimento.
**Atores**:
Desenvolvedor e Analista.
**Pré-condição**:
1. Necessita da solicitação cadastrada e a análise de viabilidade feita.
**Sequência de Eventos**:
1. O desenvolvedor consulta as solicitações pendentes para desenvolvimento.
2. O desenvolvedor, com ajuda do analista, analisa como será feito o desenvolvimento.
3. O ator desenvolve e atualiza o registro da solicitação com seus resultados.
**Pós-Condição**:
O cadastro da solicitação é atualizada no sistema com os resultados do desenvolvimento.

**Quadro 9 – Caso de uso desenvolver solicitações**

A expansão do caso de uso analisar, organizar e distribuir solicitações é apresentada no Quadro 10.

**Caso de uso:**
Analisar, organizar e distribuir solicitações.
**Descrição**:
Análise, organização e distribuição de novas solicitações.
**Evento Iniciador**:
Uma solicitação é incluída no sistema, e precisa ter seu cronograma definido, quantidade de horas, desenvolvedores, entre outros.
**Atores**:
Analista.
**Pré-condição**:
1. Necessita da solicitação incluída no sistema.
**Sequência de Eventos**:
1. O ator consulta as solicitações pendentes de análise.
2. O ator organiza as solicitações, atualizando algum dado se preciso, como recusar a solicitação, deixar em espera, definir previsão, entre outros.
3. O ator analisa as solicitações e distribui aos desenvolvedores e testadores que vão ser responsáveis de concluir a solicitação.
**Pós-Condição**:
O cadastro da solicitação é atualizada no sistema com os dados inseridos pelo analista, e ficam disponíveis na lista de afazeres dos funcionários.

**Quadro 10 – Caso de uso analisar, organizar e distribuir solicitações**

No Quadro 11 é apresentada a expansão do caso de uso controlar e atualizar solicitações.

| |
|---|
| **Caso de uso:**<br>Controlar e atualizar solicitações.<br>**Descrição**:<br>Controle e atualização de solicitações durante o processo de desenvolvimento ou de testes.<br>**Evento Iniciador**:<br>Uma solicitação é aprovada e encaminhada como tarefa à equipe.<br>**Atores**:<br>Analista, desenvolvedor, testador.<br>**Pré-condição**:<br>1. Necessita da aprovação da solicitação.<br>**Sequência de Eventos**:<br>1. Os atores controlam diariamente suas tarefas, e atualizam as mesmas com os resultados obtidos de seus processos de desenvolvimento e testes.<br>**Pós-Condição**:<br>O cadastro da solicitação é atualizado no sistema. |

**Quadro 11 – Caso de uso controlar e atualizar solicitações**

A Figura 4 apresenta o diagrama de entidades e relacionamento do sistema. Nele são detalhadas as 19 tabelas que compõe o banco de dados do sistema.

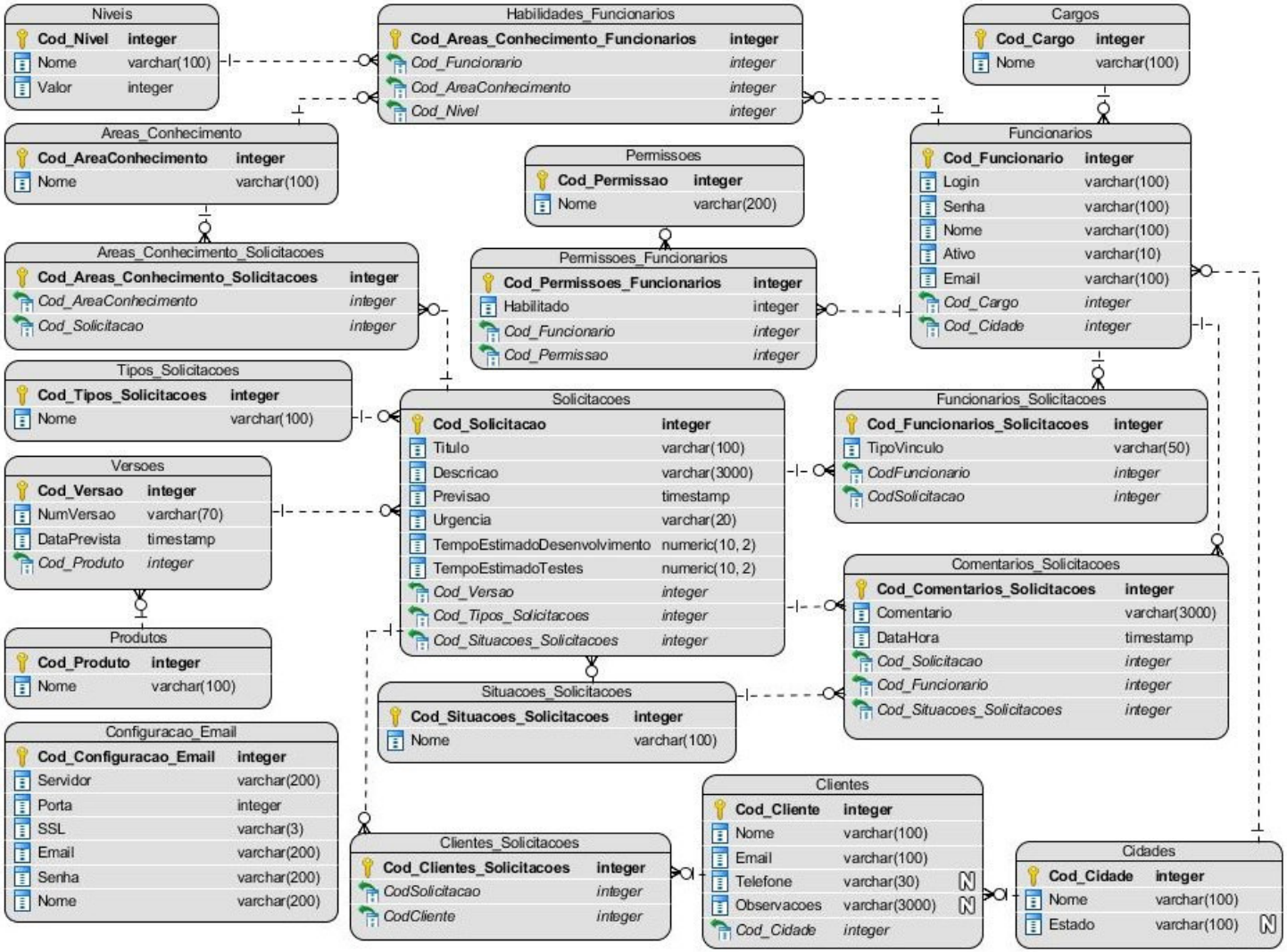


**Figura 4 – Diagrama de entidades e relacionamentos**

O sistema está centrado nas tabelas de solicitações e funcionários, as quais dependem das tabelas de cadastros gerais para seu adequado funcionamento.

Os funcionários possuirão apenas informações importantes, como cargo e situação de ativo, visto que ele é uma parte crucial do sistema. Essa mesma tabela também será usada como usuário, por isso possuirá também um campo de *login* e senha. Seus principais relacionamentos serão com os cargos e com a definição de habilidades, indispensável na análise e distribuição de solicitações.

A tabela de solicitações possui relação com quase todas as outras tabelas. Além de campos básicos como previsão e urgência, poderá possuir vários clientes e funcionários relacionados, será classificada em tipos e situações, possuirá comentários, áreas de conhecimento necessárias para o desenvolvimento e vínculo com uma versão de um determinado produto. Exceto a tabela de situações de solicitações, que possui dados padrão e não poderá ser alterada, é imprescindível que todas as outras tabelas relacionadas possuam dados cadastrados corretos e atualizados. Eles são necessários para o bom funcionamento do sistema, devido ao grande número de relacionamentos e dependências entre as tabelas.

## 4.3 APRESENTAÇÃO DO SISTEMA

Ao acessar o sistema é apresentada a tela de *login* na qual o funcionário deve inserir seus dados de *login* e senha. Os dados desses campos são criptografados e comparados com os dados dos funcionários ativos do banco de dados. Caso eles sejam correspondentes aos armazenados, o *login* será realizado e a tela inicial do sistema será exibida.

Na tela inicial do sistema (Figura 5) está localizada a barra de menus, usada para a navegação geral no sistema. Os menus e sub-menus permitirão ao usuário acessar os cadastros e funcionalidades do sistema, sempre abrindo uma nova janela quando um clique é realizado.

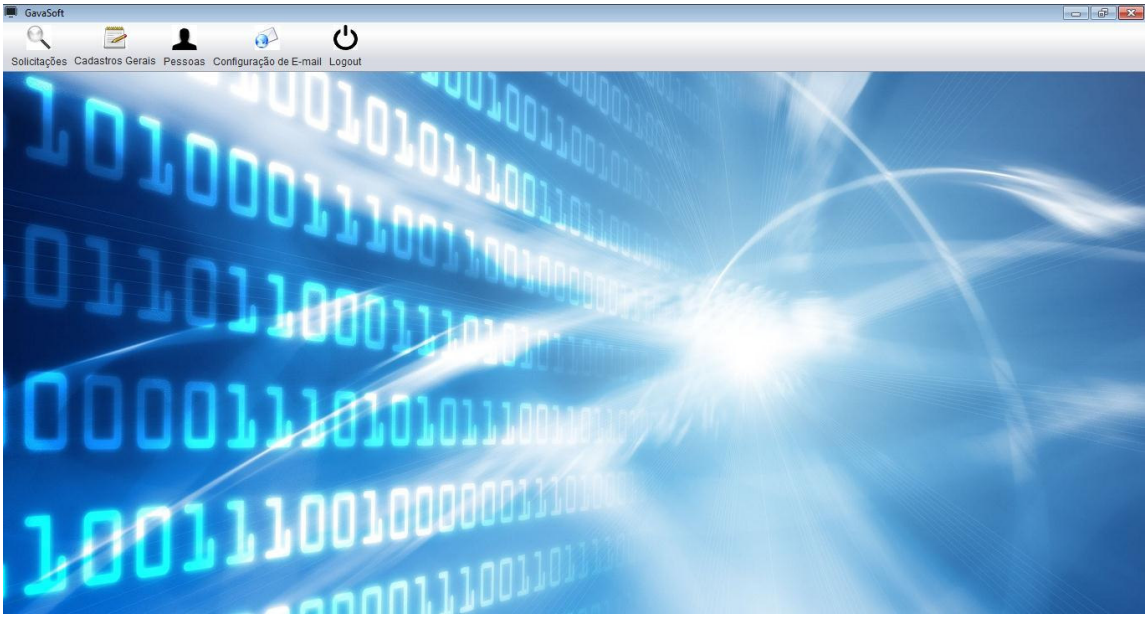


**Figura 5 – Tela inicial do sistema**

Todas as telas pertencentes aos menus principais de cadastros gerais e de pessoas (Figura 6) possuem *layout* e funcionamento semelhante. Elas possuem filtros específicos, ícones adicionados dinamicamente, uma tabela para exibição dos dados relevantes, e botões para pesquisar, incluir, editar e excluir registros, todos com ícones atribuídos dinamicamente. As únicas exceções, neste caso, são as telas de Versões e Funcionários, que possuem também um botão de relatório.

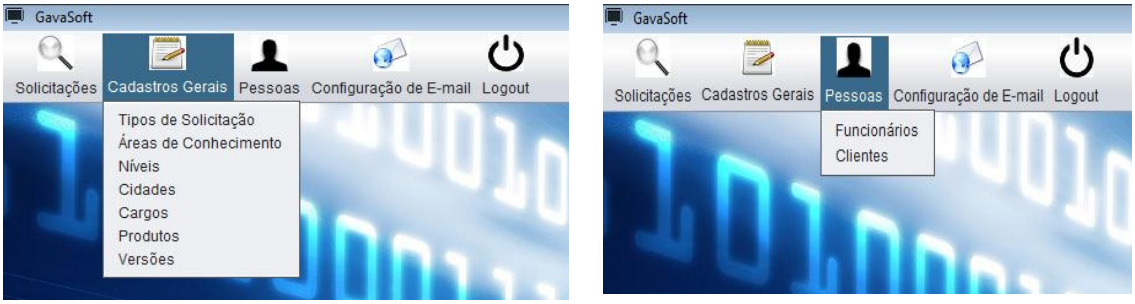


**Figura 6 – Cadastros gerais e de pessoas**

Nas telas de cadastros, todos os campos de texto nos filtros estão vinculados ao pressionamento da tecla Enter que ativará a função de Pesquisar. Além disso, em todas as tabelas é implementado o evento de duplo clique, que ativará a função de Editar para o registro da tabela selecionado nesse evento.

Em todas essas telas, ao clicar na função de Incluir será aberta uma nova janela modal do cadastro correspondente com os campos em branco. Nela, haverá

atributos específicos correspondentes ao cadastro, um botão de Cancelar para fechar a tela, e um botão de Salvar. Todos os campos de texto no cadastro da tela estão vinculados ao pressionamento da tecla Enter, que ativará a função de Salvar. Ao usar a função de Salvar é realizada a verificação dos campos obrigatórios vazios. Caso o salvamento seja realizado com sucesso, a janela modal será fechada automaticamente, os dados serão cadastrados na tabela correspondente no banco de dados e o novo registro será incluído na lista atualizada.

Ao clicar no ícone de editar será validado se existe um registro selecionado e uma mensagem será exibida. Ao selecionar um registro na tabela e usar o botão de editar, ou dois cliques em um registro da tabela, será aberta a mesma janela modal citada anteriormente, porém, com o cadastro já completo com os dados do registro selecionado na tabela, incluindo o código do registro exibido em uma *label* não editável.

Ao clicar no ícone de excluir também será validado se existe um registro selecionado e uma mensagem correspondente será mostrada. Caso a exclusão seja possível, o registro será excluído do banco de dados e da lista na tela, caso contrário será exibida uma mensagem informando que não foi possível realizar a exclusão..

A seguir são apresentadas as telas presentes em ambos os menus que contém os padrões acima, com detalhes de suas especificações não padrões:

A Figura 7 apresenta a tela de filtro de tipos de solicitações, acessada por meio do menu Cadastros Gerais -> Tipos de Solicitações. Ela mostrará na tabela inicialmente todos os tipos de solicitações cadastradas, segundo plano da Figura 6. A modal de cadastro, exibida em primeiro plano na Figura 6, possui apenas o campo de nome. O restante dos atributos de ambas as telas são iguais aos padrões citados anteriormente.


**Figura 7 – Tela de cadastro e consulta de tipos de solicitações**

Os tipos das solicitações serão vinculados posteriormente às solicitações e são usados para classificá-las, por exemplo, em correções, sugestões, implementações, entre outros.

A Figura 8 apresenta a tela de cadastro e filtro de áreas de conhecimento, acessada pelo menu Cadastros Gerais -> Áreas de Conhecimento. Ela mostrará na tabela inicialmente todas as áreas de conhecimento cadastradas (plano de fundo). Sua tela modal de cadastro acessada por meio dessa (primeiro plano na figura), possuirá apenas o campo de nome, bem semelhante às telas de filtro e cadastro de Tipos de Solicitações, a diferença é a impossibilidade de cadastrar duas áreas com o mesmo nome, com principal objetivo de evitar problemas e confusões na distribuição das solicitações. O restante dos atributos de ambas as telas são iguais aos padrões citados anteriormente.

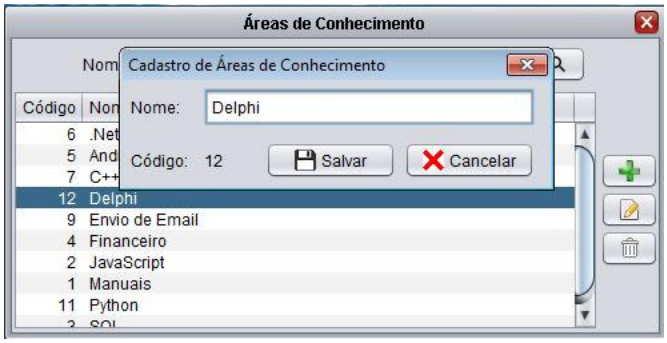


**Figura 8 – Tela de cadastro e consulta de áreas de conhecimento**

As áreas de conhecimento podem ser usadas de forma bem abrangente e possuem um papel importante no controle das solicitações. Uma ou mais áreas poderão ser vinculadas às solicitações, para definir com quais áreas de conhecimento as tarefas da solicitação em questão são relacionadas. Todas as áreas terão vínculos também com as habilidades dos funcionários. Cada funcionário possuirá um nível de conhecimento em cada habilidade, que será essencial na otimização da distribuição de tarefas para desenvolvedores e testadores. Alguns exemplos de áreas de conhecimento são: SQL, Relatórios, Financeiro, NFS-e, *Extensible Markup Language* (XML), entre outros.

A Figura 9 apresenta a tela de cadastro e filtro de níveis, acessada pelo menu Cadastros Gerais -> Níveis. Ao abrir a tela são apresentados, inicialmente, todos os níveis cadastrados. A tela modal de cadastro acessada (primeiro plano na Figura 8) possuirá o campo de nome e valor, que disponibilizará para escolha em um

componente do tipo ComboBox valores padrões de 1 a 10, usado para ordenar os níveis do maior ao menor, bem semelhante às telas de filtro e cadastro de tipos de solicitações A diferença está na impossibilidade de cadastrar dois níveis com o mesmo valor. O restante dos atributos de ambas as telas são iguais aos padrões citados anteriormente.

Os níveis são usados na definição das habilidades do funcionário. Cada nível será vinculado à habilidade de determinado funcionário em determinada área de conhecimento, e como a área de conhecimento, também será essencial na otimização da distribuição de tarefas para desenvolvedores e testadores.


**Figura 9 - Tela de cadastro e consulta de níveis de conhecimento**

A tela de cadastro e filtro de cidades, acessada por meio do menu Cadastros Gerais -> Cidades, é apresentada na Figura 10. Inicialmente são apresentadas as cidades já cadastradas. A tela modal de cadastro possuirá apenas o campo de nome, como ocorre nas telas de filtro e cadastro tipos de solicitações. Os demais atributos são semelhantes aos padrões já citados anteriormente.


**Figura 10 - Tela de cadastro e consulta de cidades**

Na Figura 11 está a tela de filtro e de cadastro de cargos que é acessada por meio do menu Cadastros Gerais -> Cargos. O funcionamento é semelhante às demais telas de cadastro apresentadas. Os cargos de Administrador, Analista,

Desenvolvedor e Testador já vêm cadastrados e não podem ser excluídos pelo fato de serem usados em diversos funcionamentos no software.

Os cargos são vinculados ao cadastro de funcionários e serve para definir qual seu cargo na empresa. Para desenvolvedores e testadores, será necessário usar os cadastros padrões para o funcionamento correto do sistema.

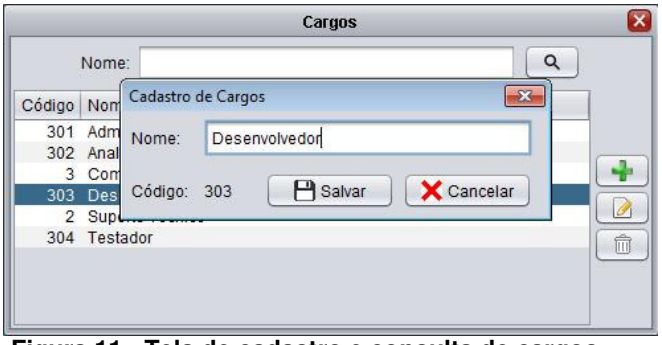


**Figura 11 - Tela de cadastro e consulta de cargos**

A Figura 12 apresenta a tela de filtro e cadastro de produtos, acessada pelo menu Cadastros Gerais -> Produtos. O funcionamento dessa tela é semelhante às demais de cadastro apresentadas. Os produtos correspondem aos produtos comercializados pela empresa usuária deste software. Eles possuirão versões que serão vinculadas posteriormente às solicitações, para possibilitar o controle de versões dos produtos da empresa.


**Figura 12 - Tela de cadastro e consulta de produtos**

*A* tela de filtro de versões, acessada pelo menu Cadastros Gerais -> Versões, é apresentada na Figura 13. Sua tela modal de cadastro possuirá o campo de número da versão, data prevista e produto, que carregará em um componente do tipo ComboBox todos os produtos cadastrados no banco de dados. O restante dos atributos de ambas as telas são iguais aos padrões citados anteriormente.

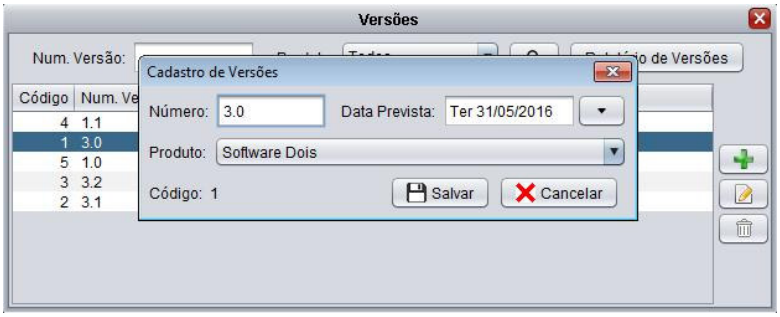


**Figura 13 - Tela de cadastro e consulta de versões**

A tela exibida na Figura 14 mostrará, inicialmente, todas as versões cadastradas, mas diferente das outras telas de filtro desse menu, possui também um botão de relatório, que exibirá um relatório desenvolvido com a ferramenta Jasper Studio de todas as versões agrupadas por produto por meio do JasperViewer, apresentado na Figura 13.

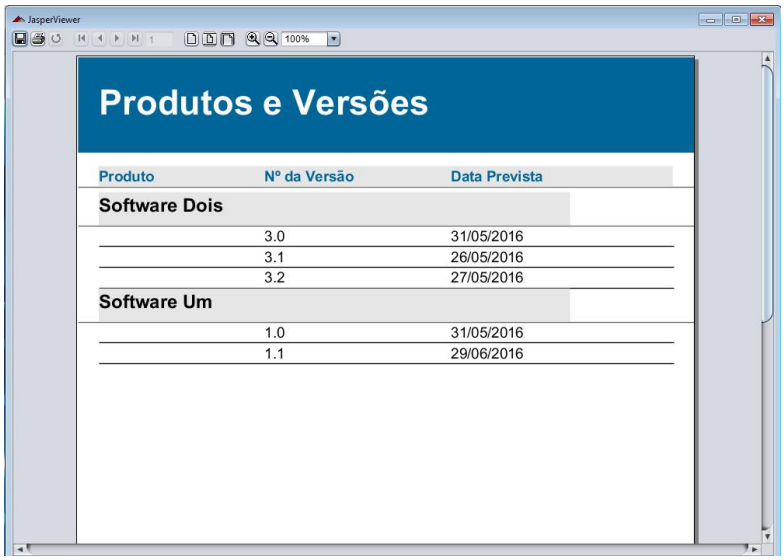


**Figura 14 – Tela de relatório de produtos e versões**

As versões são agrupadas por produto e são vinculadas diretamente às solicitações, para possibilitar o controle de versões dos produtos da empresa.

A Figura 15 apresenta a tela de filtro de clientes, acessada por meio do menu Pessoas -> Clientes. Ela mostrará na tabela inicialmente todos os clientes cadastrados.

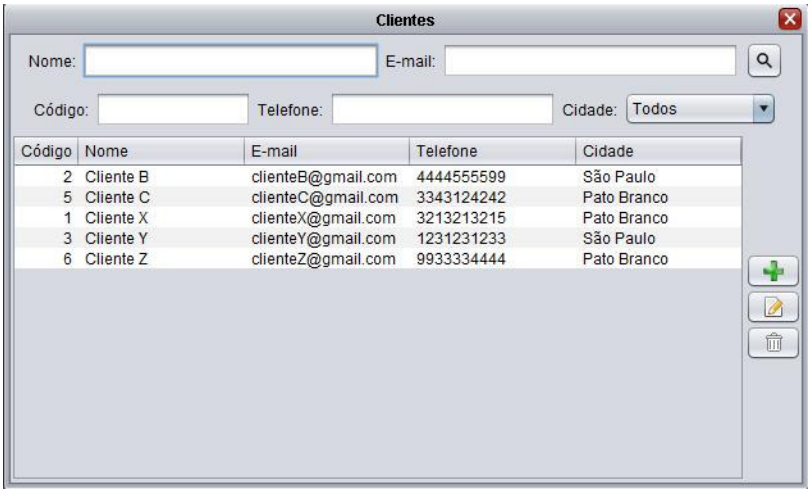


**Figura 15 – Tela de consulta de clientes**

A tela modal de cadastro de clientes (Figura 16) possui alguns campos distintos das outras telas, são eles: nome, campo obrigatório, sem nenhuma restrição; e-mail, para questão de comunicação da empresa, o qual precisa obrigatoriamente possuir uma arroba (@) e um ponto final (.) em seu conteúdo para ser válido; telefone, não obrigatório, para comunicação da empresa; observações, mostrado em um formato diferente que possibilita a inserção de um vasto texto com espaços, exibida em um componente do tipo TextArea e; cidade, que carregará em um componente ComboBox todas as cidades cadastradas. O restante dos atributos de ambas as telas são iguais aos padrões citados anteriormente.

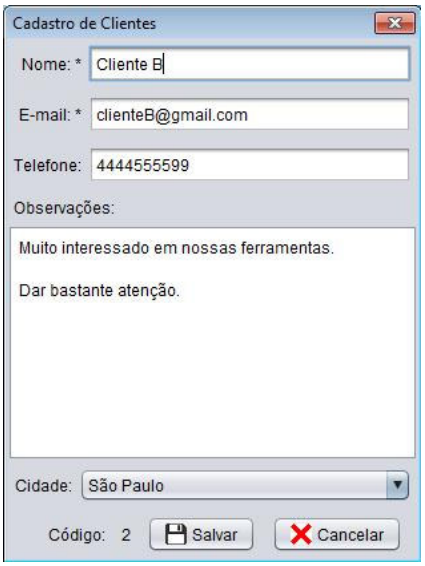


**Figura 16 – Tela de cadastro de clientes**

Os clientes servirão principalmente para serem vinculados às solicitações, a fim de registrar quais clientes estão necessitando da implementação da solicitação em questão. Esse controle também poderá ser usado para comunicação da empresa usuária do software com os clientes.

A Figura 17 apresenta a tela de filtro de funcionários, acessada pelo menu Pessoas -> Funcionários. Ela mostrará na tabela inicialmente todos os funcionários cadastrados e possui, também, um botão de relatório, que exibirá um relatório desenvolvido com a ferramenta Jasper Studio de todas as habilidades dos funcionários, ou seja, todos os níveis em todas as áreas de conhecimento agrupadas por funcionário, através do JasperViewer, apresentado na Figura 18.

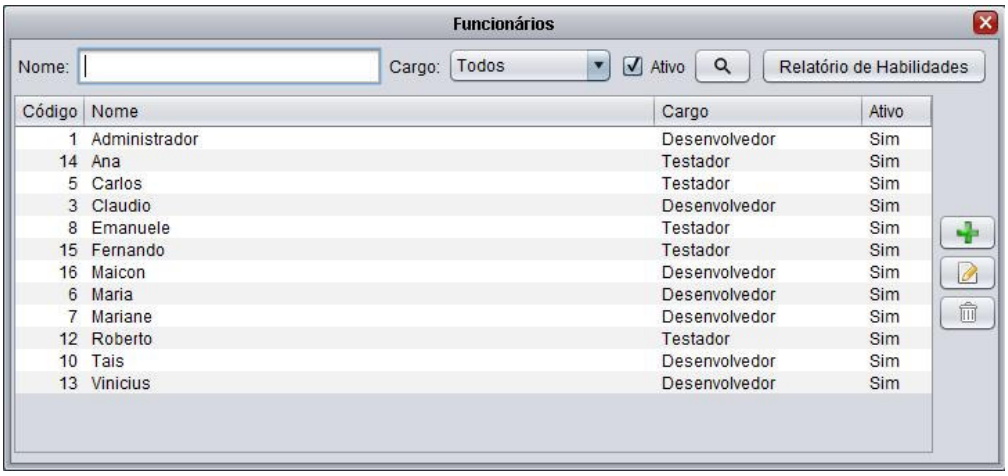


**Figura 17 – Tela de consulta de funcionários**

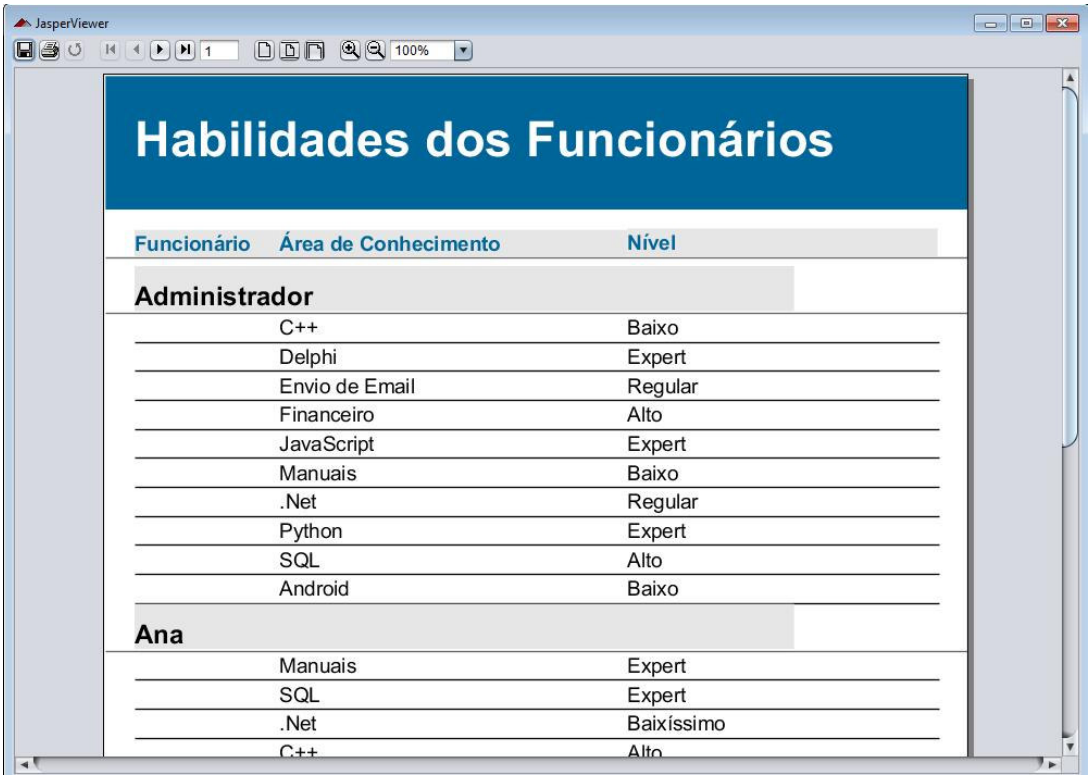


**Figura 18 – Relatório de habilidades de funcionários**

A tela modal de cadastro de funcionários apresentada na Figura 19, possui vários campos e funções distintas das outras telas.

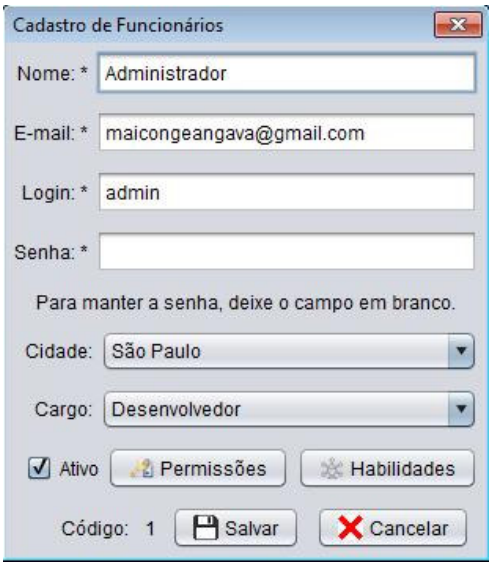


**Figura 19 – Tela de cadastro de funcionários**

Os campos disponíveis na tela de cadastro de funcionários são os seguintes:

a) Nome, campo obrigatório, sem nenhuma restrição.

b) E-mail, importante para receber as atualizações de comentários nas solicitações quando o funcionário é interessado na mesma. Precisa, obrigatoriamente, possuir uma arroba (@) e um ponto final (.) em seu conteúdo para ser válido.

c) Login, usado para acessar o sistema. Aqui existe a validação de duplicidade, sendo impossível cadastrar um login igual ao login de outro funcionário.

d) Senha, usada também para acessar o sistema. Ela precisa possuir mais de 8 caracteres e ao salvar é criptografada no banco de dados. Esse campo em específico não é carregado ao Editar um funcionário, ele fica em branco, porém, a senha se mantém a mesma, caso o campo continue em branco ao salvar novamente depois da edição. Na label abaixo do campo são mostradas instruções sobre a senha.

f) Cidade, que carregará em um componente ComboBox todas as cidades cadastradas.

g) Cargo, que carregará em um componente ComboBox todos os cargos cadastrados.

h) Ativo, para marcar se o funcionário é ativo ou não. Caso não seja, seu Login não funcionará mesmo se estiver correto.

i) O botão de Permissões abrirá uma nova tela modal com vários componentes CheckBox, com todas as permissões do sistema. Ao salvar, as permissões marcadas serão salvas na tabela de permissões de funcionários como habilitadas, e as desmarcadas como desabilitadas.

As permissões salvas para os funcionários serão validadas quando uma tela, cuja exibição dependa de alguma permissão, é aberta. Caso não possua a permissão, a operação relacionada será bloqueada, por bloqueio ou ocultação do componente.

A tela de permissões que podem ser atribuídas a usuários do sistema é apresentada na Figura 20.

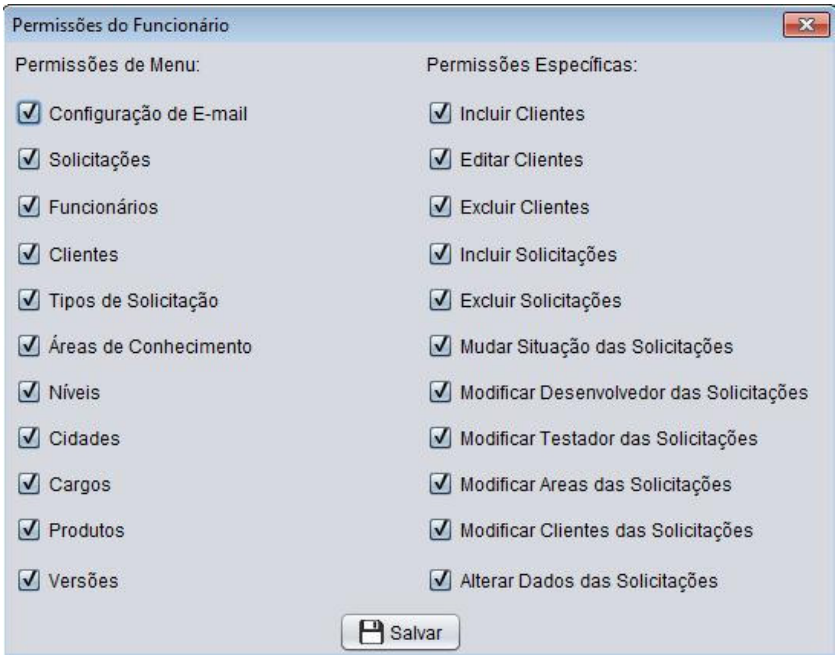


**Figura 20 – Tela de permissões de usuários do sistema**

O botão de Habilidades (tela apresentada na Figura 21) abrirá uma nova tela modal que irá incluir dinamicamente vários componentes, resultando em uma listagem de todas as áreas de conhecimento cadastradas, sendo possível selecionar em um componente ComboBox um nível dentre os cadastrados para cada uma das áreas de conhecimento. Será gerado, também dinamicamente, um botão de Salvar que percorre todos os componentes da tela e salva um registro na tabela de habilidades para cada uma das áreas de conhecimento e exclui os antigos a fim de substituir as habilidades cadastradas anteriormente.

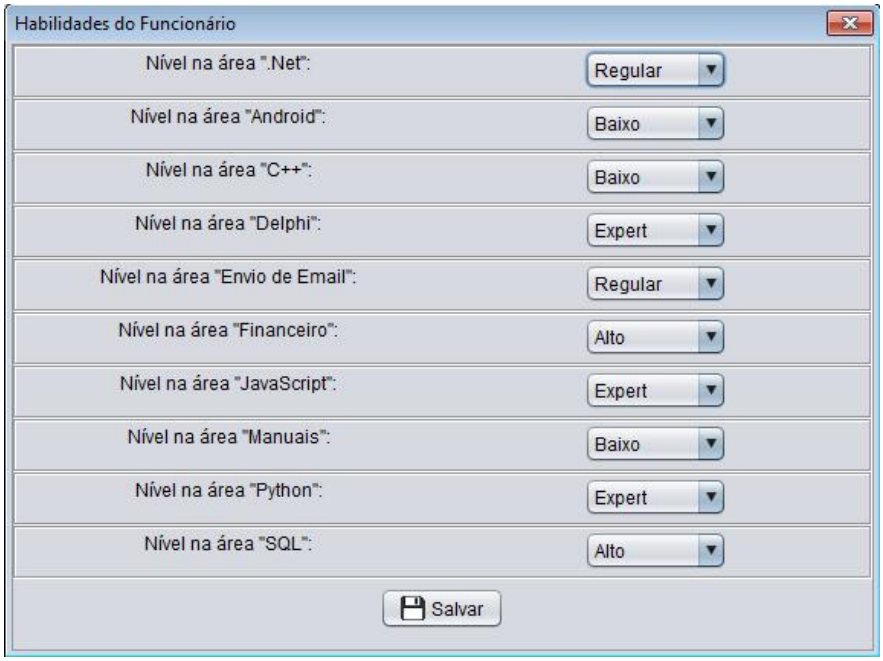


**Figura 21 – Tela para atribuição de habilidades a usuários do sistema**

A exibição dessa tela de cadastro de funcionário possui um funcionamento um pouco diferente do padrão. Ao incluir um novo funcionário os botões de permissões e habilidades são bloqueados, porém, após salvar a inclusão ambas as janelas serão abertas automaticamente e não podem ser fechadas, obrigando o usuário definir permissões e habilidades. Quando isso acontece, por padrão as permissões aparecem todas marcadas na janela e as habilidades definidas com o menor nível cadastrado.

Ao editar um funcionário, os botões são habilitados, e ao clicar neles, as permissões e habilidades dos funcionários são carregados nas telas.

A Figura 22 apresenta a tela de configuração de e-mail, acessada através do menu Configuração de E-mail. Este menu não possui sub-menus. Na tela serão apresentados campos para uma configuração de envio de e-mail, a qual será usada para enviar os e-mails aos interessados das solicitações quando um comentário for adicionado.

Ao usar a função de Salvar, a configuração será salva no banco de dados, substituindo a configuração anterior. Essa configuração será validada quando do envio de e-mails.

Por questão de segurança, o campo de senha nunca é carregado, e é sempre exibido em branco mesmo que a senha exista de fato.

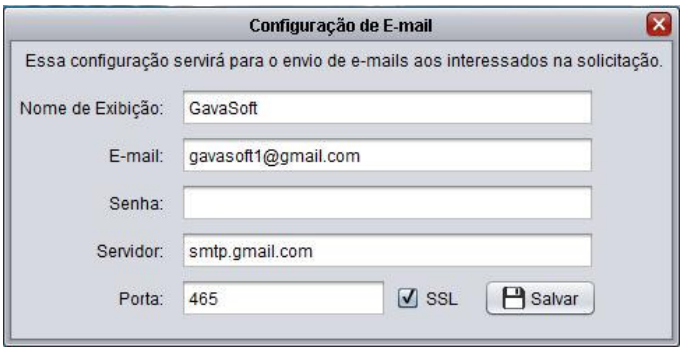


**Figura 22 – Formulário de configurações de e-mail**

Ao clicar no botão de Logout, a tela principal do sistema será fechada juntamente com todas as telas abertas e a tela de Login será mostrada novamente.

A Figura 23 apresenta a tela de filtro de solicitações, acessada pelo menu Solicitações. Essa tela é uma das principais do sistema, nela, é possível buscar as solicitações. Dentre alguns exemplos de pesquisas que podem ser realizadas são: pesquisar solicitações concluídas de determinada versão de produto, para produção de manuais para a versão ou algo do gênero; pesquisar solicitações em determinada situação de determinado desenvolvedor ou testador, para verificar as tarefas em aberto ou concluídas; buscar solicitações com maior urgência.

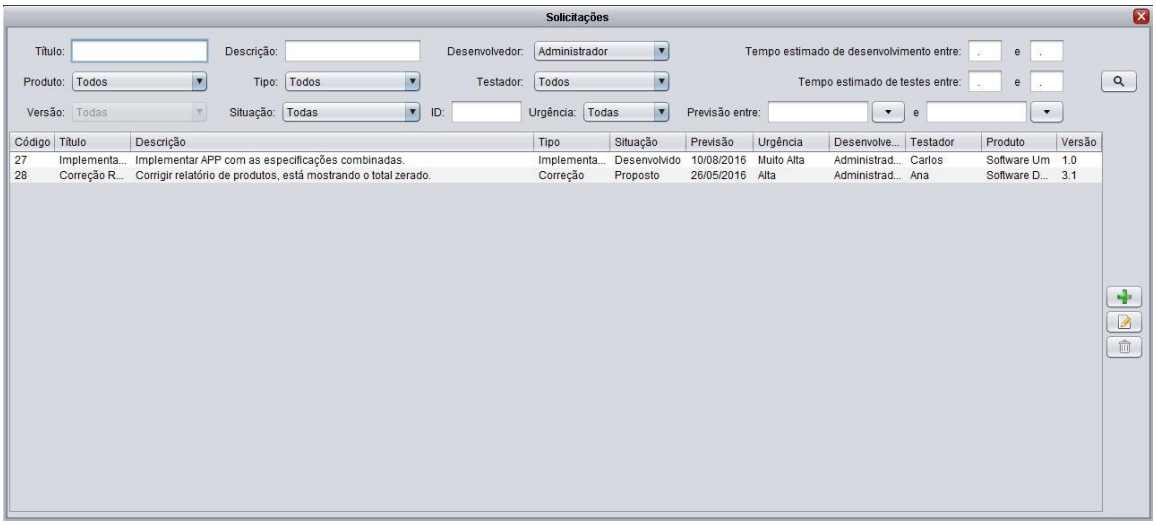


**Figura 23 – Tela de filtro de solicitações**

A tela modal de cadastro de solicitações (Figura 24) possui vínculos com quase todos os cadastros do sistema. Ela possui vários componentes e funções. Ao Incluir uma nova solicitação, muitos componentes são desbloqueados após salvar a solicitação.

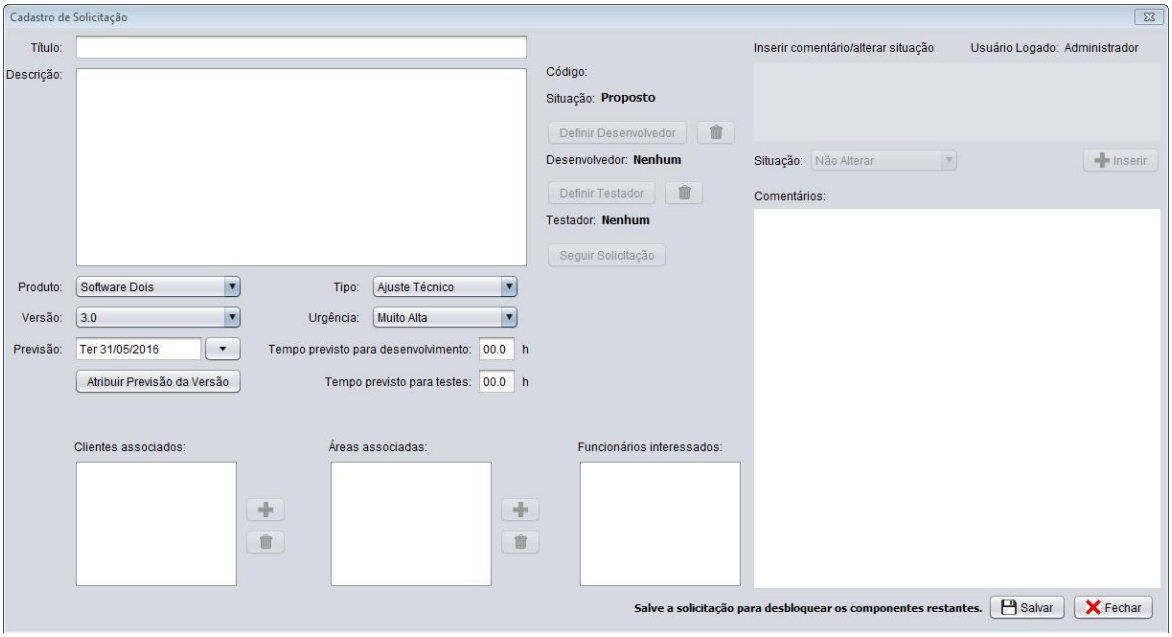


**Figura 24 – Tela de cadastro de solicitações (a)**

Todos os componentes disponíveis e não bloqueados na inclusão são obrigatórios e devem ser informados antes de salvar, como é informado na mensagem destacada na parte inferior da tela. São os seguintes:

a) Título, para resumir o assunto da solicitação, por exemplo: ajuste no relatório de caixa;

b) Descrição, para detalhar tudo o que for relevante à solicitação, como detalhes da mesma, cuidados a serem tomados, entre outros;

c) Produto e Versão, para vincular a solicitação a uma versão disponibilizada pela empresa referente a algum produto. O componente ComboBox de produtos carregará todos os produtos cadastrados e ao selecionar algum deles, o ComboBox de versões é carregado com todas as versões do produto selecionado;

d) Previsão, para definir qual a previsão de conclusão da solicitação, em conjunto com um botão que possibilita atribuir a previsão vinculada à versão definida;

e) Tipo, para definir qual a classificação dessa solicitação. Carregará no componente ComboBox todos os tipos de solicitação cadastrados;

f) Urgência, para definir qual a urgência da solicitação. Carregará no componente ComboBox urgências padrões. São as seguintes: Muito Alta, Alta, Média, Baixa, Muito baixa;

g) Tempo previsto para desenvolvimento e Tempo previsto para testes, importante para controlar o tempo ocupado pelos desenvolvedores e testadores responsáveis por realizar a tarefa em questão. Essa informação será considerada na tela acessada pelos botões Definir Desenvolvedor e Definir Testador que será apresentada na sequência. Os campos possuem um formato padrão para informá-lo em formato decimal, e será carregado por padrão sempre com o valor zerado, no caso de empresas que não desejam controlar o tempo ocupado pelos funcionários.

Após completar todos os campos é possível, então, salvar a solicitação. Ao salvar, algumas operações específicas serão feitas automaticamente pelo sistema, são elas:

a) Diferente das outras telas, essa tela modal não será fechada, com objetivo de continuar os cadastros dos outros componentes que agora serão desbloqueados;

b) A solicitação será salva automaticamente na situação "Proposto".

c) O componente Label que mostra o código da solicitação será atualizado e mostrado na tela.

d) Um comentário será salvo no banco de dados automaticamente e exibido na tela. Ele será salvo com data/hora atual, usuário logado, situação da solicitação proposto, e com a descrição padrão: " Solicitação criada pelo funcionário <usuário logado>". Isso tem como objetivo registrar o momento da criação da solicitação e quem é o responsável por essa operação.

Após a solicitação ser salva e a tela ser atualizada, será possível também trabalhar com os outros componentes que antes eram bloqueados. O mesmo vale ao editar uma solicitação. A tela de edição de solicitação com os componentes desbloqueados é apresentada na Figura 25.

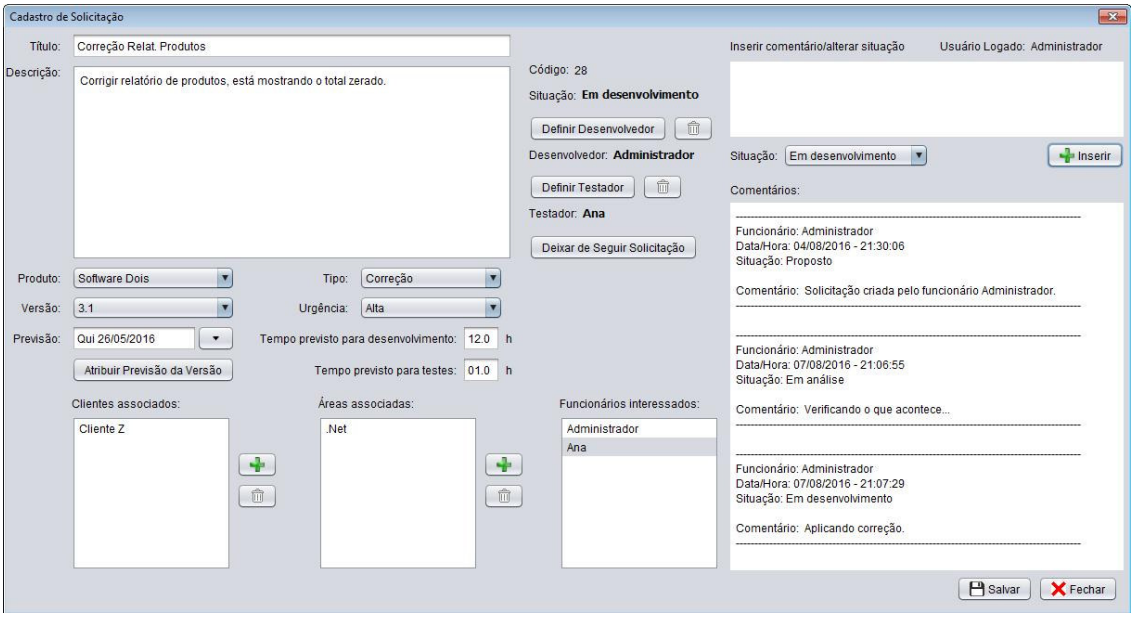


**Figura 25 – Tela de cadastro de solicitações (b)**

Os componentes agora desbloqueados que não foram anteriormente citados, são os seguintes:

a) Clientes associados - mostra em uma lista todos os clientes vinculados à solicitação. Ao lado do quadro, haverá um botão de Incluir e Excluir. Ao clicar em Excluir, será validado se existe um registro selecionado na lista. Se existir, o registro será excluído do banco de dados e da lista. Ao clicar em Incluir será aberta uma tela de pesquisa de clientes, mostrada na Figura 26, contendo os mesmos filtros da tela de filtros de clientes comum, porém, com apenas a função de Adicionar. A função poderá ser usada selecionando um registro na tabela e clicando no botão, ou então dando duplo clique em um registro na tabela.

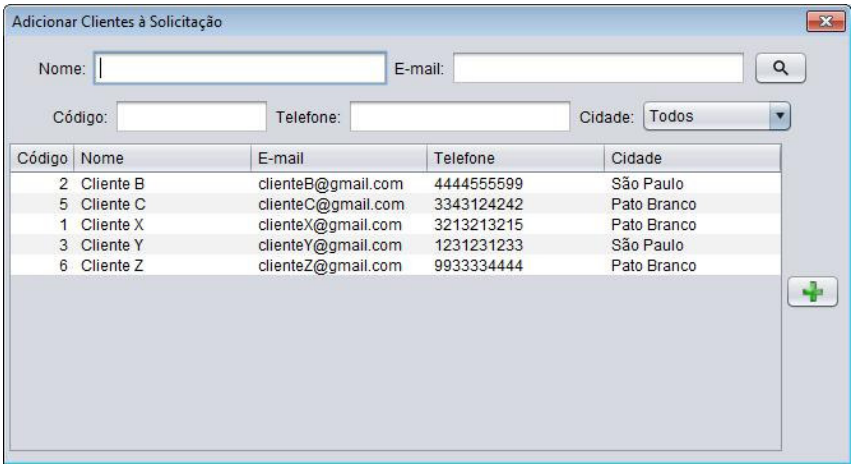


**Figura 26 – Adicionar clientes à solicitação**

Ao tentar adicionar um cliente será validado se o cliente já não foi vinculado à solicitação, caso não tenha sido, ele será adicionado à solicitação e salvo no banco de dados. Ao fechar essa tela, a lista de clientes vinculados será atualizada.

b) Áreas associadas - mostrará todas as áreas de conhecimento vinculadas à solicitação. Possui funcionamento semelhante à função de adicionar clientes. O objetivo dessa função é definir as áreas de conhecimento necessárias para a realização da solicitação. A Figura 27 apresenta a tela para vincular áreas de conhecimento à solicitação.

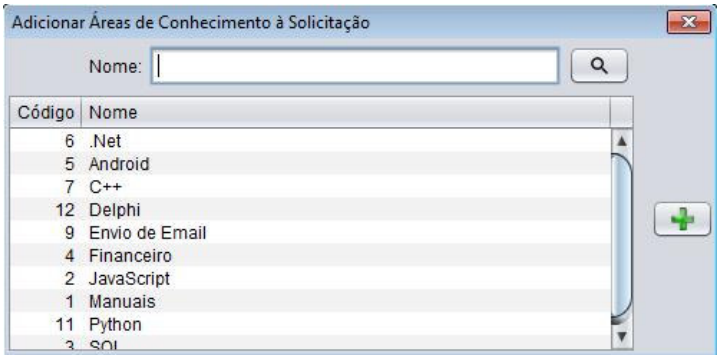


**Figura 27 – Vincular áreas de conhecimento às solicitações**

c) Seguir Solicitação/Deixar de Seguir Solicitação/Funcionários Interessados- ao clicar em "Seguir Solicitação" o usuário logado será adicionado automaticamente na lista de funcionários interessados. Isso servirá para receber atualizações no e-mail quando algum comentário for incluído. Ao editar a solicitação, caso o funcionário logado já esteja na lista de funcionários interessados, o botão mudará seu texto para "Deixar de Seguir Solicitação" e ao clicar no mesmo o funcionário será tirado da lista.

d) Inserir comentário/Alterar situação - ao digitar o texto no campo, será obrigatório também selecionar qual situação será escolhida. No componente ComboBox de situação serão carregadas todas as situações padrões, que são: Não Alterar, Proposto, Em Análise, Em Desenvolvimento, Desenvolvido, Em Teste, Recusado, Testado-Passou, Testado-Falhou e Concluído/Liberado.

Consequentemente, para mudar a situação da solicitação, é obrigatório incluir um comentário, visando um melhor registro das modificações. Caso o funcionário deseje somente comentar sem alterar a situação da solicitação basta escolher a opção "Não Alterar".

Ao clicar em Inserir comentário, será enviado um *e-mail* de atualização para os funcionários interessados com todos os dados do comentário. Durante o envio será mostrada uma tela de carregamento e caso o envio falhe para algum destinatário será mostrada uma mensagem informando a falha e quais e-mails não receberam o aviso, como é mostrado na Figura 28.

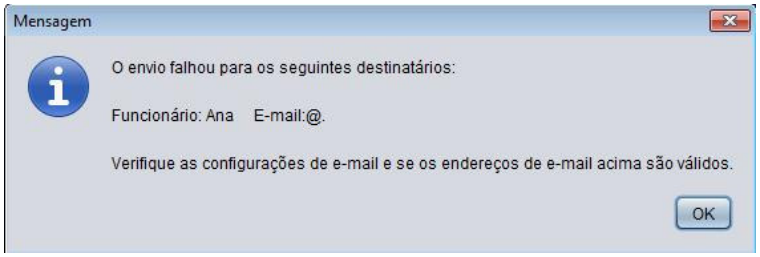


**Figura 28 – Mensagem de falha de envio de e-mail**

Após isso, o comentário será salvo no banco de dados e exibido no quadro de comentários.

e) Definir Desenvolvedor/Definir Testador - ambos os botões servirão para vincular um funcionário à solicitação como desenvolvedor ou testador. Para usar essas funções, a solicitação precisa de pelo menos uma área de conhecimento vinculada. Eles abrirão a mesma tela, porém, com descrições diferentes de acordo com o cargo, como mostrado nas Figuras 29 e 30.


**Figura 29 – Tela para vincular desenvolvedor à solicitação**

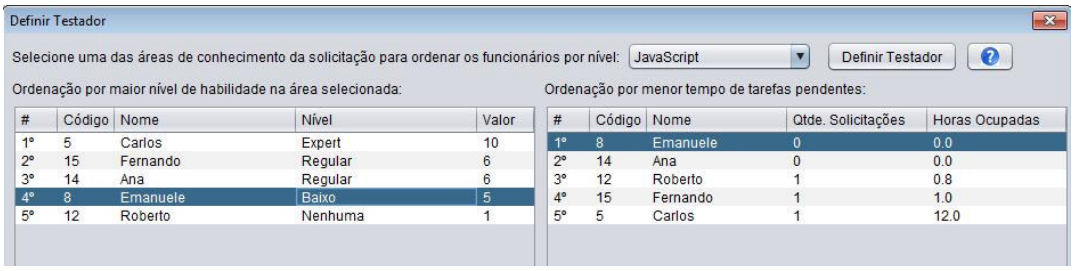


**Figura 30 – Tela para vincular testador à solicitação**

Essas telas possuirão o seguinte funcionamento:

Ao abrir a tela serão mostrados os dados das tabelas dos funcionários de acordo com o botão clicado (testador ou desenvolvedor). Se o botão Definir Desenvolvedor foi usado, serão mostrados todos os funcionários com o cargo padrão de Desenvolvedor. Se usado o botão Definir testador, serão mostrados todos os funcionários com o cargo padrão de Testador.

Na tela também será exibido um componente ComboBox com todas as áreas de conhecimento vinculadas à solicitação. Ele servirá para definir os dados da primeira tabela, que mostrarão todas as habilidades dos funcionários na área selecionada, ordenada de maneira decrescente.

Na segunda tabela, os funcionários serão mostrados com as informações de quantidade de solicitações em aberto e quantidade de horas ocupadas, baseado nas informações da solicitação.

Na parte superior haverá um botão de Ajuda explicando quais situações de solicitações são consideradas na hora de atribuir os valores para a segunda tabela, como mostra a Figura 31.

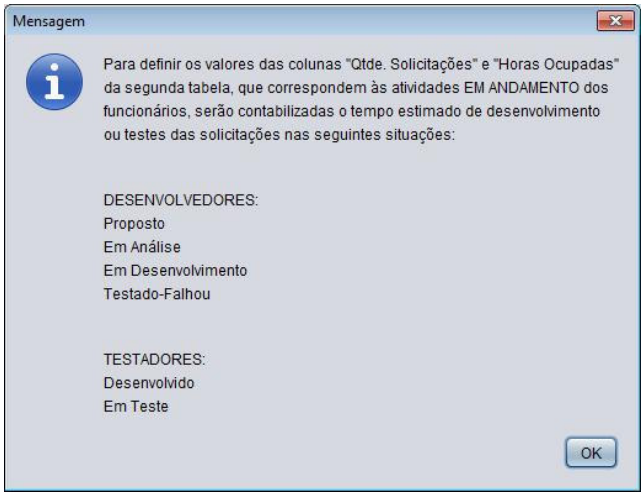


**Figura 31 – Tela de ajuda para vincular desenvolvedor ou testador à solicitação**

A ordenação de ambas as tabelas é muito importante para fornecer uma visão mais abrangente ao analista no momento de distribuir as tarefas. Na primeira tabela, é possível visualizar de forma ordenada quais são os funcionários mais experientes na área em questão e comparar com a segunda tabela, para verificar se o funcionário pretendido na primeira tabela tem um tempo considerável disponível ou não.

Para ajudar nessa visão, ao selecionar um registro em uma tabela, com o mouse ou alternar o registro com as setas do teclado, o mesmo registro na outra tabela também é selecionado. Caso a área de conhecimento seja trocada, o funcionário continua selecionado na tabela.

O funcionário será vinculado ao ser selecionado e clicar no botão de definir, ou então dar dois cliques no registro da tabela. Ao fazer isso, o sistema irá validar se o mesmo já não está vinculado à solicitação, ou se já existe outro funcionário do mesmo cargo vinculado. Caso o mesmo já esteja vinculado será mostrada uma mensagem de aviso. Caso exista outro funcionário vinculado, será possível substituí-lo. Isso serve para deixar vinculado sempre apenas um desenvolvedor ou testador à solicitação.

Ao definir o desenvolvedor ou testador, o mesmo também será automaticamente incluído na lista dos interessados na solicitação, para que recebam as atualizações de comentários e mudanças de situação no e-mail. Isso auxilia o analista na distribuição das tarefas, já avisando os funcionários de suas tarefas pendentes diretamente pelo comentário.

Ao fechar a tela, o funcionário escolhido será exibido na Label e poderá ser removido sem ser substituído pelo botão de exclusão. Também será exibido na lista de funcionários interessados.

## 4.4 DESENVOLVIMENTO DO SISTEMA

O sistema foi desenvolvido utilizando a tecnologia de desenvolvimento Java Desktop, em conjunto com a JPA como *framework* de persistência de dados. Utilizou-se o padrão *Model-View-Controller* (MVC), e quase que completamente orientado a objetos.

O código do software é bem vasto e existem tratamentos diferentes em cada caso específico. Em seguida são exibidos alguns exemplos de códigos com padrões comuns no software.

***a) Estrutura JPA - Banco de Dados***

A Listagem 1 mostra o código contido na principal classe de persistência do projeto que é essencial para o funcionamento correto do JPA. É por meio desse XML que parte da comunicação com o banco de dados é estruturada.

Nas classes são declaradas as pertencentes à integração e nas propriedades os dados de conexão com o banco.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence     version="2.1"     xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="TCCPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>TCC.Model.AreasConhecimento</class>
    <class>TCC.Model.AreasConhecimentoSolicitacoes</class>
    <class>TCC.Model.ClientesSolicitacoes</class>
    <class>TCC.Model.Solicitacoes</class>
    <class>TCC.Model.Cidades</class>
    <class>TCC.Model.TiposSolicitacoes</class>
    <class>TCC.Model.Cargos</class>
    <class>TCC.Model.SituacoesSolicitacoes</class>
    <class>TCC.Model.Produtos</class>
    <class>TCC.Model.Versoes</class>
    <class>TCC.Model.Funcionarios</class>
    <class>TCC.Model.ComentariosSolicitacoes</class>
    <class>TCC.Model.Clientes</class>
    <class>TCC.Model.Niveis</class>
    <class>TCC.Model.LogSituacoes</class>
    <class>TCC.Model.FuncionariosSolicitacoes</class>
    <class>TCC.Model.HabilidadesFuncionarios</class>
    <class>TCC.Model.InteressadosSolicitacoes</class>
    <class>TCC.Model.ConfiguracaoEmail</class>
    <class>TCC.Model.Permissoes</class>
    <class>TCC.Model.PermissoesFuncionarios</class>
    <properties>
      <property                                    name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/tcc"/>
      <property name="javax.persistence.jdbc.password" value="123456"/>
      <property                                 name="javax.persistence.jdbc.driver"
value="org.postgresql.Driver"/>
      <property name="javax.persistence.jdbc.user" value="postgres"/>
    </properties>
  </persistence-unit>
</persistence>
```

**Listagem 1 – Estrutura JPA do banco de dados**

***b) Instância do Banco de Dados***

A Listagem 2 mostra o código da classe DbConnection, responsável por fazer a conexão com o XML de persistência por meio da classe EntityManager. Ele usa o padrão de projeto Singleton, como é possível ver no método getInstance().

```
public class DbConnection {

    private static DbConnection dbConn;
    private final EntityManager entityManager;

    private DbConnection() throws Exception{
        EntityManagerFactory factory =
Persistence.createEntityManagerFactory("TCCPU");
        entityManager = factory.createEntityManager();
    }

    public static EntityManager getInstance()throws Exception{
        if (dbConn == null){
            dbConn = new DbConnection();
            return dbConn.entityManager;
        }else {
            return dbConn.entityManager;
        }
    }
}
```

**Listagem 2 – Instância do banco de dados**

***c) Exemplo de código de uma classe do pacote Model***

A Listagem 3 mostra parte de um código de exemplo de uma classe do pacote Model. As classes são definidas com seus atributos e métodos, juntamente com anotações padrões do JPA responsáveis por fazerem a integração correta com as tabelas do Banco de Dados.

As anotações possuem um funcionamento padrão e são sinalizadas pelo caractere "@" antes das mesmas. Por exemplo: a anotação @Column vincula um determinado atributo a uma tabela no banco. As funções get, set e toString são importantes para as chamadas e exibição de atributos nas telas, e principalmente pelo encapsulamento, já que os atributos são declarados, por segurança, com a identificação private.

```
@Entity
@Table(name = "produtos")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Produtos.findAll", query = "SELECT p FROM Produtos p ORDER
BY p.nome")
public class Produtos extends AbstractEntityImpl {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "cod_produto")
    private Integer codProduto;
    @Column(name = "nome")
    private String nome;
    @OneToMany(mappedBy = "codProduto")
```

```
    private List<Versoes> versoesList;

    public Produtos() {
        super();
    }

    public Produtos(Integer codProduto) {
        this.codProduto = codProduto;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    @XmlTransient
    public List<Versoes> getVersoesList() {
        return versoesList;
    }

    public void setVersoesList(List<Versoes> versoesList) {
        this.versoesList = versoesList;
    }

    @Override
    public String toString() {
        return nome;
    }

    @Override
    public Integer getId() {
         return this.codProduto;
    }

    @Override
    public void setId(Integer id) {
        this.codProduto = id;
    }
```

**Listagem 3 – Classe do pacote modal**

***d) Exemplo de código da classe abstrata do pacote DAO***

A Listagem 4 mostra parte de um código de uma classe abstrata do pacote *Data Access Object* (DAO). Ela possui métodos padrões para serem usados em comandos relacionados ao banco de dados em qualquer uma das classes. Ela é estendida por todas as outras classes do mesmo pacote.

Essa classe faz, basicamente, todo o vínculo do banco de dados com as classes citadas anteriormente e a migração dos funcionamentos básicos como salvar ou excluir para todas as classes que a estenderem. Esse funcionamento é

dado principalmente ao uso da classe EntityManager e a substituição de objetos de classe, mostrada no código com o caracter T.

```
public T salvar(T obj) throws Exception{
        if (obj.isPersistente()){
            entity.merge(obj);
        }else {
            entity.persist(obj);
        }
        entity.flush();
        return obj;
    }

    public abstract T getById(Integer id)throws Exception;

    public Boolean excluir(T obj)throws Exception{
        obj = getById(obj.getId());
        entity.remove(obj);
        entity.flush();
        return true;
    }

    public abstract List<T> getTodos()throws Exception;
```

**Listagem 4 – Classe abstrata do pacote DAO**

***e) Exemplo de código de uma classe do pacote DAO***

A Listagem 5 apresenta parte de um código de exemplo de uma classe do pacote DAO. Nessa classe são declaradas as consultas e os tratamentos relacionados ao banco de dados. Ela estende a classe abstrata mostrada na Listagem 4, por isso não é necessário declarar novamente métodos como salvar ou excluir. A seguir são mostradas as formas mais comuns de chamadas ao banco de dados:

a) O primeiro método getById busca a classe pelo ID diretamente no entity, atributo estendido da classe abstrata, que contém a instância do banco de dados.

b) O segundo método apenas chamada uma NamedQuery declarada na classe do Model, como mostrado no código da Listagem 4.

c) O terceiro método possui uma chamada personalizada, chamada com parâmetros (String num) que são substituídos no local das variáveis (nesse exemplo o :num) na consulta usando o setParameter.

```
@Override
    public Versoes getById(Integer id) throws Exception {
        return entity.find(Versoes.class, id);
    }

    @Override
    public List<Versoes> getTodos() throws Exception {
        Query query = entity.createNamedQuery("Versoes.findAll");
```

```
        return query.getResultList();
    }

    public List<Versoes> getByNumVersao(String num) throws Exception {
        String sql = "Select v From Versoes v where v.numVersao = :num ORDER BY
v.dataPrevista DESC";
        Query query = entity.createQuery(sql);
        query.setParameter("num", num);
        return query.getResultList();
    }
```

**Listagem 5 – Classe do pacote DAO**

### *f) Exemplo de código de uma classe do pacote Controller*

Na Listagem 6 está parte de um código de exemplo de uma classe do pacote Controller, responsável por fazer a comunicação entre o View e o DAO, com as chamadas de métodos de consulta registrados nela. Essa classe contém as declarações dos métodos correspondentes das classes do pacote DAO e poderão ser chamadas nas classes do pacote View.

```
public List<Versoes> getFiltroModificado(String comandosql, Produtos codProduto)
throws Exception {
        return versoesDAO.getFiltroModificado(comandosql, codProduto);
    }

    public List<Versoes> getByProduto(Produtos codProduto) throws Exception {
        return versoesDAO.getByProduto(codProduto);
    }

    @Override
    public List<Versoes> getTodos() throws Exception{
        return versoesDAO.getTodos();
    }
```

**Listagem 6 – Classe do pacote controller**

### *g) Método de criptografia de senha*

A Listagem 7 apresenta o código usado para criptografar a senha dos funcionários no banco de dados, usando a função hash criptográfica MD5.

```
public static String criptografarMd5(String senha){
        String sen = "";
        MessageDigest md = null;
        try {
                md = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
        }
        BigInteger hash = new BigInteger(1, md.digest(senha.getBytes()));
        sen = hash.toString(16);
        return sen;
    }
```

**Listagem 7 – Método de criptografia de senha**

### *h) Método para verificação de login*

O código usado para validar o *login* criptografado do funcionário no banco de dados é apresentado na Listagem 8. É usada a classe de *controller* de funcionários (funcionariosCtrl) para acessar o método getTodos(), buscar todos os funcionários e inseri-los em uma lista de objetos da classe de funcionários (listaVerificacao).

Após isso, percorre toda a lista de funcionários, verifica se é ativo, se o login é igual, criptografa a senha e compara para ver se é a mesma, e então habilita o mesmo a entrar no sistema.

```
public void acessar(){
        habilitado= false;
        try {
            funcionariosCtrl = new FuncionariosCtrl();
            listaVerificacao = new ArrayList<>();
            String senha = new String(txtSenha.getPassword());
            listaVerificacao = funcionariosCtrl.getTodos();
            for(Funcionarios func : listaVerificacao){
                if(func.getAtivo().equals("Sim")){
                    if(func.getLogin().equals(txtLogin.getText())){
                        if(func.getSenha().equals(criptografarMd5(senha)))
                            habilitado=true;
                            usuarioLogado = func;
                            break;
                    }
                }
            }
            if(habilitado==false)
                JOptionPane.showMessageDialog(rootPane,     "Login     ou     senha
incorretos!");
            else{
               this.setVisible(false);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

**Listagem 8 – Método para verificação de login**

### *i) Exemplo de método incluir em uma tela de filtro*

Na Listagem 9 está parte de um código de uma classe do pacote View, mais especificamente uma tela de filtros. O código em questão é utilizado ao usar a função Incluir da tela. Ela irá abrir outra tela modal, como é o caso da variável dlg do código. Ao executar o método setVisible, a operação será realizada na outra janela, e quando essa for fechada, o código continuará. Na sequência, ele verifica se o registro da tela aberta foi salvo, e então usa a função de salvar do Controller para salvar o registro no banco de dados.

```
public void incluir(){
        DlgVersoes dlg = new DlgVersoes(frame, true);
        dlg.setLocationRelativeTo(null);
        dlg.setVisible(true);
        if (dlg.getOperacao() == 'S') {
            versao = dlg.getVersao();
            try {
                versao = versoesCtrl.salvar(versao);
                dlg.dispose();
                atualizarLista();
                txtFiltrarNumVersoes.setText("");
                cmbFiltrarProdutoVersoes.setSelectedIndex(0);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
```

**Listagem 9 – Método incluir em uma tela de filtro**

***j) Exemplo de método editar em uma tela de filtro***

Na Listagem 10 está parte do código de uma classe do pacote View, mais especificamente uma tela de filtros. Esse código é utilizado na função Incluir da tela.

Primeiro, ela verifica se existe um registro selecionado na tabela, se não existir, mostra um aviso e através do comando return, encerra o método. Se não for o caso, na sequência o registro selecionado é convertido em objeto. O sistema faz isso por meio do código contido na tabela, com esse código ele usa o método padrão da classe abstrata do pacote DAO chamada getById, e atribui à um objeto do mesmo tipo. Após isso, ele abre a tela em questão da mesma forma que no Incluir, porém neste caso usa o método *setVersao*, para atribuir o objeto à tela e completar os campos.

```
public void editar(){
        if (tblVersoes.getSelectedRow() == -1) {
            JOptionPane.showMessageDialog(rootPane,    "Favor    selecionar    um
registro.");
            return;
        }
        Integer row = tblVersoes.getSelectedRow();
        row = tblVersoes.convertRowIndexToModel(row);
        Integer codigo = Integer.parseInt(model.getValueAt(row, 0) + "");
        try {
            versao = versoesCtrl.getById(codigo);
            DlgVersoes dlg = new DlgVersoes(frame, true);
            dlg.setLocationRelativeTo(null);
            dlg.setVersao(versao);
            dlg.setVisible(true);
            if (dlg.getOperacao() == 'S') {
                versao = versoesCtrl.salvar(versao);
                pesquisar();
            }
        } catch (Exception ex) {
```

```
            ex.printStackTrace();
        }
    }
```

**Listagem 10 – Método editar em uma tela de filtro**

### *k) Exemplo de método excluir em uma tela de filtro*

Na Listagem 11 está parte do código de uma classe do pacote View, mais especificamente uma tela de filtros. Esse código é utilizado com a função Excluir da tela. Inicialmente ele faz um processo semelhante com o editar, porém, ao invés de abrir uma nova tela, ele exclui o registro do banco de dados e também na tabela.

```
public void excluir(){
       if (tblVersoes.getSelectedRow() <= -1) {
            JOptionPane.showMessageDialog(rootPane,    "Favor    selecionar    um
registro.");
            return;
         }
         Integer row = tblVersoes.getSelectedRow();
         row = tblVersoes.convertRowIndexToModel(row);
         Integer codigo = Integer.parseInt(model.getValueAt(row, 0) + "");
         try {
             versao = versoesCtrl.getById(codigo);
             if (versoesCtrl.excluir(versao)) {
                 model.removeRow(row);
             }
         } catch (Exception ex) {
             JOptionPane.showMessageDialog(rootPane, "Não foi possível excluir o
registro, pois o mesmo possui vínculos.");
         }
    }
```

**Listagem 11 – Método excluir em uma tela de filtro**

### *l) Exemplo de pesquisa em uma tela de filtro*

A Listagem 12 mostra o método usado para as pesquisas realizadas por meio dos filtros da tela. Ele funciona da seguinte forma: são passados três parâmetros, sendo a chave o primeiro. Se o primeiro parâmetro for uma condição verdadeira, é retornado o segundo parâmetro, e se for falso, o terceiro.

```
public static Object retornoCondicao(boolean expressao, Object ParteTRUE, Object
ParteFALSE){
        if (expressao)
         return ParteTRUE;
        else
         return ParteFALSE;
    }
```

**Listagem 12 – Método de pesquisa em uma tela de filtro**

A Listagem 13 mostra como esse método auxilia, na prática, ao realizar um filtro. Inicialmente é declarada uma *string* vazia. Para cada campo contido na tela,

será feita a verificação referente ao campo. Se o mesmo estiver filtrado, o método será usado verificando se a *string* está vazia ou não. Se estiver, é adicionado a parte do comando sql WHERE, senão adicionado AND.

Dessa forma, a primeira condição sempre será *where* e as próximas sempre serão AND, afinal, depois da primeira vez, a *string* não estará mais vazia e o método sempre vai retornar, nesse caso, AND.

Após fazer todas as verificações, é adicionado antes da *string* o comando básico no qual é selecionada a tabela em questão, neste exemplo "comandosql = "Select v From Versoes v " + comandosql".

Com toda a consulta definida, é usado um método específico de consulta contido no Controller correspondente à tela e depois os registros da tabela da tela são atualizados com o que foi filtrado.

```
public void pesquisar(){
        try {

            comandosql="";

            if(!txtFiltrarNumVersoes.getText().isEmpty()){
               comandosql = comandosql +
               MetodosUteis.retornoCondicao("".equals(comandosql), "where ", "and
") + "v.numVersao like '%" +
                        txtFiltrarNumVersoes.getText().toUpperCase() + "%' ";
            }


if(!"Todos".equals(cmbFiltrarProdutoVersoes.getSelectedItem().toString())){
                comandosql = comandosql +
                MetodosUteis.retornoCondicao("".equals(comandosql), "where ", "and
") + "v.codProduto = :codProduto";

                produto = (Produtos) cmbFiltrarProdutoVersoes.getSelectedItem();
            }

            comandosql = "Select v From Versoes v " + comandosql;

            listaVersoes = versoesCtrl.getFiltroModificado(comandosql, produto);

                for (int x = model.getRowCount()-1; x >= 0 ;x--){
                    model.removeRow(x);
                }
                for (Versoes versoes : listaVersoes){
                    addVersao(versoes);
                }
            }catch (Exception ex){
                ex.printStackTrace();
            }
        }
```

**Listagem 13 – Uso do método de pesquisa em uma tela de filtro**

**m) *Exemplo de chamada de relatório***

A Listagem 14 contém o método usado para chamar um relatório criado no Jasper Studio. Inicialmente é declarada a conexão com o banco de dados e depois é definido o caminho e o nome do relatório, para por fim mostrá-lo.

```
private void imprimirRelatorio() {
        try {

            Connection                                cn                                =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/tcc"
                    ,"postgres", "123456");

            JasperReport             report             =             (JasperReport)
JRLoader.loadObjectFromFile("versoeseprodutos.jasper");
            JasperPrint imp = JasperFillManager.fillReport(report, null, cn);
            JasperViewer viewer = new JasperViewer(imp, false);
            viewer.setVisible(true);

        }catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

**Listagem 14 – Método para imprimir relatório**

***n) Exemplo de adição de dados em tabelas***

Na Listagem 15 está o método usado para adicionar dados em determinada tabela. O método recebe parâmetros de dados e esses dados são adicionados em um vetor de objetos, por fim o vetor é adicionado como uma linha da tabela. Em muitos casos esse método é usado dentro de um laço de repetição, para adicionar várias linhas.

```
public void addVersao(Versoes versao) {
        Object[] vetor = new Object[4];
        vetor[0] = versao.getId();
        vetor[1] = versao.getNumversao();

        SimpleDateFormat dt = new SimpleDateFormat("dd-MM-yyyy");

        vetor[2] = dt.format(versao.getDataprevista());
        vetor[3] = versao.getCodProduto();
        model.addRow(vetor);
    }
```

**Listagem 15 – Método para armazenar dados de versão  na respectiva tabela**

***o) Exemplo de adição de dados em uma ComboBox***

Na Listagem 16 está o método usado para adicionar dados em um componente ComboBox.  O funcionamento é simples, por meio de um método do Controller são buscados os registros necessários, acrescentados em uma lista e

então a lista é percorrida e item a item são adicionados na ComboBox utilizando o método addItem.

```
private void addCmbProdutos() throws Exception {

        produtosCtrl = new ProdutosCtrl();

        produtos = produtosCtrl.getTodos();

        for(Produtos p : produtos){
            cmbFiltrarProdutoVersoes.addItem(p);
        }
    }
```

**Listagem 16 – Adicionar itens a uma lista**

***p) Exemplo de método para completar os campos do cadastro ao editar***

A Listagem 17 apresenta um exemplo de método usado para completar os campos de uma tela ao editar um registro. Ele recebe como parâmetro geralmente um objeto da classe correspondente com os dados, e eles são atribuídos aos componentes da tela.

```
public void setCliente(Clientes cliente){
        this.cliente = cliente;
        this.lblCodClientes.setText(String.valueOf(cliente.getId()));
        this.txtNomeClientes.setText(cliente.getNome());
        this.txtEmailClientes.setText(cliente.getEmail());
        this.txtTelefoneClientes.setText(String.valueOf(cliente.getTelefone()));

        String[] desc = cliente.getObservacoes().split("_-_");
        carregaObservacoes="";

        for(Integer i = 0;i < desc.length;i++){
                  carregaObservacoes = carregaObservacoes + desc[i] + "\n";
            }

        this.txtObservacoesClientes.setText(carregaObservacoes);

        cmbCidadeClientes.setSelectedItem(cliente.getCodCidade());
    }
```

**Listagem 17 – Método para completar campos do cadastro na sua edição**

***q)Exemplo de criação dinâmica de componentes***

Na Listagem 18 está o método usado para construir a tela de habilidades de funcionários, baseado nos registros contidos no banco de dados. A tela, inicialmente, contém um painel (JPanel). O método cria todos os componentes dinamicamente para cada área de conhecimento existente, criando e organizando painéis, descrições e uma combo para selecionar os níveis.

```
    public boolean construirTela(){
        if(listaTodasAreasConhecimento.size()<1 || listaNiveis.size()<1){
            JOptionPane.showMessageDialog(rootPane, "Nenhuma área de conhecimento
ou nível cadastrado. "
                    + "\n\nRealize os cadastros necessários e "
                    + "defina as habilidades do funcionário posteriormente.");
            return false;
        }
        for(AreasConhecimento area : listaTodasAreasConhecimento){

            panel1 = new JPanel();
            panel2 = new JPanel();
            panel3 = new JPanel();
            label = new JLabel();
            comboNiveis = new JComboBox<>();
            label.setText("Nível na área \"" + area.getNome()+ "\":");
            panelPrincipal.add(panel1);
            panel1.setLayout(new GridLayout(1, 2, 1, 1));
            panel1.setBorder(javax.swing.BorderFactory.createEtchedBorder());
            panel2.add(label);
            panel2.setLayout(new FlowLayout(1, 1, 1));
            panel3.add(comboNiveis);
            panel3.setLayout(new FlowLayout(1, 1, 1));
            panel1.add(panel2);
            panel1.add(panel3);
            popularCombos();
            comboNiveis.setSelectedIndex(comboNiveis.getItemCount()-1);

            try {
                    listaHabilidadesFuncionarios                                    =
habilidadesFuncionarioCtrl.getByFuncionario(funcionario);
                    if(listaHabilidadesFuncionarios.size()>0){
                       for(HabilidadesFuncionarios                h                :
listaHabilidadesFuncionarios){
                        if(area == h.getCodAreaConhecimento()){
                            comboNiveis.setSelectedItem(h.getCodNivel());
                        }}}
catch (Exception ex) {
                    ex.printStackTrace();
                }
        }
        panel4 = new JPanel();
        panelPrincipal.add(panel4);
        Icon save = new ImageIcon("save.png");
        salvarHabilidades = new JButton();
        salvarHabilidades.setIcon(save);
        salvarHabilidades.setText("Salvar");
        salvarHabilidades.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          salvar();
        }
      });
        panel4.add(salvarHabilidades);
        return true;}
```

**Listagem 18 – Método para construir a tela de habilidades de funcionários**

**r) *Exemplo de método para salvar um cadastro***

Na Listagem 19 está um exemplo de método para salvar algum registro no banco de dados. Nesse método são, geralmente, feitas as verificações de campo na tela, e por fim, se tudo estiver correto é instanciado um novo objeto da classe em questão e definidos os valores através dos setters. Em alguns cadastros o método do controller salvar é usado na tela de pesquisa dependendo do retorno da variável "operacao", e não na própria tela de cadastro.

```
    public void salvar(){
        if (cliente == null){
           cliente = new Clientes();
       }

       if (!lblCodClientes.getText().trim().isEmpty()){
            Integer cod = Integer.parseInt(lblCodClientes.getText());
            cliente.setId(cod);
        }

       if (txtNomeClientes.getText().isEmpty() ||
            txtEmailClientes.getText().isEmpty() ||
           cmbCidadeClientes.getSelectedItem() == null)
           JOptionPane.showMessageDialog(rootPane,   "Complete   os   campos   para
salvar.");
       else            if(!txtEmailClientes.getText().contains("@")            ||
!txtEmailClientes.getText().contains(".")){
           JOptionPane.showMessageDialog(rootPane, "E-mail inválido");
       }else{
            cliente.setNome(txtNomeClientes.getText());
            cliente.setEmail(txtEmailClientes.getText());
            cliente.setTelefone(txtTelefoneClientes.getText());

            String[] obs = txtObservacoesClientes.getText().split("\n");
            registraObservacoes="";

            for(Integer i = 0;i < obs.length;i++){
                    registraObservacoes = registraObservacoes + obs[i] + "_-_";
            }

            cliente.setObservacoes(registraObservacoes);
            cliente.setCodCidade((Cidades)cmbCidadeClientes.getSelectedItem());

            operacao = 'S';
            setVisible(false);
       }
    }
```

**Listagem 19 – Método salvar**

***s) Exemplo de método mais detalhado para popular uma tabela***

Na Listagem 20 está um exemplo mais detalhado com vários tratamentos e verificações para popular uma tabela. Neste caso, é a tabela de desenvolvedores,

para vincular os mesmos à solicitações. São realizadas verificações de cargo, se é ativo ou não, e são ordenados e vinculados corretamente à tabela.

```
public void popularTabelaNivel(){

        while(model1.getRowCount()>0){
            model1.removeRow(0);
        }

        try {

            Cargos cargo;

            if("Desenvolvedor".equals(tipoFuncionario)){
                cargo = cargosCtrl.getById(303); //Cargo desenvolvedor
            }else{
                cargo = cargosCtrl.getById(304); //Cargo testador
            }

            listaFuncionarios = funcionariosCtrl.getByCargo(cargo);

        } catch (Exception ex) {

Logger.getLogger(DlgDefinirFuncionarioSolicitacoes.class.getName()).log(Level.SEVE
RE, null, ex);
        }

        for(Funcionarios func : listaFuncionarios){
                try {
                    listaHabilidadesFuncionarios                                    =
habilidadesFuncionariosCtrl.getByFuncionarioArea(func,
areaConhecimentoSelecionada);
                } catch (Exception ex) {

Logger.getLogger(DlgDefinirFuncionarioSolicitacoes.class.getName()).log(Level.SEVE
RE, null, ex);
                }
            for(HabilidadesFuncionarios h : listaHabilidadesFuncionarios){
                if(h.getCodFuncionario().getAtivo().equals("Sim")){
                    listaHabilidadesFuncionariosOrdenada.add(h);
                }
            }
        }

        Collections.sort(listaHabilidadesFuncionariosOrdenada);

        Integer x = 1;

        for(HabilidadesFuncionarios h : listaHabilidadesFuncionariosOrdenada){
           String colocacao = x + "º";
           addFuncionariosPorNivel(h, colocacao);
           x++;
        }

        listaHabilidadesFuncionarios.clear();
        listaHabilidadesFuncionariosOrdenada.clear();
        listaFuncionarios.clear();
    }
```

**Listagem 20 – Método para popular uma tabela**

### *t) Exemplo de método para verificar permissões*

A Listagem 21 contém um método de exemplo de verificação das permissões do funcionário logado que é executado ao acessar a tela, para bloquear componentes se necessário.

```
   public void verificaPermissoes(){
        try {
            listaPermissoesFuncionarios                                          =
permissoesFuncionariosCtrl.getByFuncionario(usuarioLogado);

            for(PermissoesFuncionarios p : listaPermissoesFuncionarios){

                if(p.getCodPermissao().getId()==17 && p.getHabilitado()==0){
                    cmbSituacoesSolicitacoes.setEnabled(false);
                }else if(p.getCodPermissao().getId()==18 && p.getHabilitado()==0){
                    btnDefinirDesenvolvedorSolicitacoes.setEnabled(false);
                    btnExcluirDesenvolvedorSolicitacoes.setEnabled(false);
                }else if(p.getCodPermissao().getId()==19 && p.getHabilitado()==0){
                    btnDefinirTestadorSolicitacoes.setEnabled(false);
                    btnExcluirTestadorSolicitacoes.setEnabled(false);
                }else if(p.getCodPermissao().getId()==20 && p.getHabilitado()==0){
                    btnIncluirAreasConhecimentoSolicitacoes.setEnabled(false);
                    btnExcluirAreasConhecimentoSolicitacoes.setEnabled(false);
                }else if(p.getCodPermissao().getId()==21 && p.getHabilitado()==0){
                    btnIncluirClientesSolicitacoes.setEnabled(false);
                    btnExcluirClientesSolicitacoes.setEnabled(false);
                }else if(p.getCodPermissao().getId()==22 && p.getHabilitado()==0){
                    btnSalvarSolicitacoes.setEnabled(false);
                }
            }
        } catch (Exception ex) {
            Logger.getLogger(FrmPrincipal.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
```

**Listagem 21 – Método para verificar permissões de acesso**

### *u) Método de envio de e-mail*

O método usado para enviar e-mail aos interessados na solicitação quando um comentário é adicionado é apresentado na Listagem 22. Esse método usa bibliotecas e métodos específicos, com alguns padrões de funcionamento.

```
public void enviaEmail(ComentariosSolicitacoes c, Solicitacoes s) throws
Exception{
        listaInteressadosSolicitacoes =
                funcionariosSolicitacoesCtrl.getBySolicitacaoTipoFuncionario(s,
"Interessado");

        listaConfiguracoes = configuracaoEmailCtrl.getTodos();

        Integer contador=0;
        String emailsComErro="";
        if(!listaConfiguracoes.isEmpty()){
            ConfiguracaoEmail conf = listaConfiguracoes.get(0);
```

```
            this.setVisible(true);
            for(FuncionariosSolicitacoes f : listaInteressadosSolicitacoes){

                try {
                    SimpleEmail email = new SimpleEmail();
                    if(conf.getSsl().equals("Sim"))
                        email.setSSLOnConnect(true);
                    else
                        email.setSSLOnConnect(false);
                    email.setHostName(conf.getServidor());
                    email.setSslSmtpPort(String.valueOf(conf.getPorta()));
                    email.setAuthenticator(new
DefaultAuthenticator(conf.getEmail(), conf.getSenha()));
                    email.setFrom(conf.getEmail(), conf.getNome());
                    email.setDebug(true);
                    email.setSubject("Comentário adicionado à solicitação ID " +
c.getCodSolicitacao().getId() + "!");

                    SimpleDateFormat           formatter           =           new
SimpleDateFormat("dd/MM/yyyy");
                    String dataFormatada = formatter.format(c.getDatahora());
                    SimpleDateFormat           formatter2          =           new
SimpleDateFormat("HH:mm:ss");
                    String horaFormatada = formatter2.format(c.getDatahora());
                    email.setMsg("Funcionário que inseriu o comentário: " +
                        c.getCodFuncionario() + "\n\nData-Hora: " + dataFormatada
+ " - " +
                            horaFormatada + "\n\nSituação: "
                        + c.getCodSituacoesSolicitacoes() + "\n\nComentário:\n\n"
+ c.getComentario());
                    email.addTo(f.getCodFuncionario().getEmail());
                    email.send();
                } catch (EmailException e) {
                    e.printStackTrace();
                    emailsComErro   =   emailsComErro   +   "Funcionário:   "   +
f.getCodFuncionario().getNome() +
                            "       E-mail:" + f.getCodFuncionario().getEmail() +
"\n";
                }
            }
        }else{
            JOptionPane.showMessageDialog(rootPane, "Os e-mails não serão enviados
aos interessados na solicitação "
                    + "pois não existe configuração de e-mail salva.\n\n Para
configurar o e-mail para envio, "
                    + "acesse o menu de Configuração de E-mail.");
        }

        if(!emailsComErro.equals("")){
            JOptionPane.showMessageDialog(rootPane,  "O  envio  falhou  para  os
seguintes destinatários:\n\n"
            + emailsComErro + "\nVerifique as configurações de e-mail e se os
endereços de e-mail acima são válidos.");
        }
    }
```

**Listagem 22 – Método para enviar email**

# 5 CONCLUSÃO

Baseado na realização deste projeto, utilizando os conhecimentos da Engenharia de Software, Banco de Dados, Programação Java e conceitos de Orientação a Objetos, foi possível perceber como todos os passos de um projeto de desenvolvimento de software são indispensáveis. Entre eles estão: o planejamento, a modelagem, o projeto, a definição de requisitos, o desenvolvimento e os testes.

Por meio da definição de requisitos e criação de diagramas de análise e projeto é possível definir uma estrutura clara e relativamente completa dos dados e requisitos envolvidos no sistema. Esses fatores fornecem o conhecimento prévio para todos os envolvidos, que consequentemente auxiliará em todo o projeto, desde a parte do levantamento de requisitos até a implementação e os testes.

A modelagem auxilia, também, na manutenção do sistema, pois tornam possível ter uma visão clara dos seus objetivos e como ele funcionará, facilitando o trabalho de todos os integrantes da equipe. É certo que muitos itens planejados podem mudar no decorrer do projeto por vários motivos, porém esses fatores mantêm uma linha a ser seguida pela equipe.

Mesmo não sendo uma prática realizada comumente em um projeto de desenvolvimento de software, concluiu-se que as atividaes de análise e de modelagem são indispensáveis. No desenvolvimento do projeto, o tempo gasto nelas, acabou facilitando todo o restante do processo, aumentando a qualidade e diminuindo o tempo e a dificuldade no desenvolvimento.

Por meio da imagem do sistema proveniente da análise que permite definir corretamente como todas as operações serão feitas, foi possível realizar o desenvolvimento sempre da melhor forma possível, procurando validar, baseado no planejado, todas as operações realizadas pelo usuário ou pelo próprio sistema para evitar problemas e até mau uso do software.

A realização do projeto como um todo possibilitou uma visão maior do esforço necessário para construir um produto de software, e o quanto a qualidade do projeto é importante para obtenção de *feedbacks* positivos.

Algumas dificuldades em relação aos componentes, versões, ou problemas específicos foram encontradas, mas todas foram solucionadas. Alguns exemplos são o envio de e-mail, tratamentos específicos de componentes e ordenação de listas.

Conclui-se que o sistema desenvolvido possui funcionalidades que atende as necessidades básicas do controle interno de solicitações em empresas de desenvolvimento de software. Além de proporcionar opções visando a melhor rotina possível de trabalho, melhorando o processo de um setor de desenvolvimento de uma empresa, e consequentemente gerando um melhor aproveitamento de cada funcionário e a melhoria da qualidade e do controle no desenvolvimento e manutenção de software.

Como implementações futuras, complementares ao que foi desenvolvido, destaca-se o registro do tempo efetivamente gasto para implementar a solução. Assim, esse tempo pode ser utilizado para auxiliar na distribuição das solicitações de manutenção.

## REFERÊNCIAS


ARAKAKI, Reginaldo; SOUZA, Alexandra A. **Processo de manutenção de software web apoiado pela modelagem IHC**. DataGramaZero - Revista de Ciência da Informação, v. 7, n. 6, dez/06, 2006. Disponível em: <http://www.dgz.org.br/dez06/Art_01.htm>. Acesso em: 15 out. 2015.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR ISO/IEC 12207: 2009. **Engenharia de sistemas e software – Processos de ciclo de vida de software**. Rio de Janeiro: ABNT, 2009.

BASILI, Victor et al. **Understanding and predicting the process of software maintenance release**. In: 8th IEEE International Conference on Software Engineering (ICSE). Berlim, Alemanha: IEEE Computer Society, 1996, p. 464-474.

CALAZANS, Angélica T. S.; OLIVEIRA, Marcelo A. L. **Avaliação de Estimativa de Tamanho para Projetos de Manutenção de Softwares.** In: Proc. of Argentine Symposium on Software Engineering, 2005.

ENGHOLM Junior, Hélio. **Engenharia de software na prática**. São Paulo: Novatec, 2013.

FIUTEM, Roberto et al. A Cliché-Based Environment to Support Architectural Reverse Engineering. **IEEE Computer Society**. Proceedings of the 1996 International Conference on Software Maintenance (ICSM '96), 1063-6773/96, 1996.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1219-1998 - IEEE Standard for Software Maintenance**, Software Engineering Standards Committee of the IEEE Computer Society, 1998.

LINGER, Richard; MOORE, Andrew. **Foundations for survivable system development: service traces, intrusion traces, and evaluation models**. Carnigie-Melon Software Engineering Institute, Pittsburgh, Tecnical Report, CMU/SEI-2001-TR-029, Outubro 2001.

MENS, Tom et al. **Challenges in software evolution**. In: 8th IEEE International Workshop on Principles of Software Evolution (IWPSE). Lisboa, Portugal: IEEE Computer Society, 2005, p. 13–22.

PADUELLI, Mateus M. **Manutençao de software: problemas típicos e diretrizes para uma disciplina específica.** Dissertação (mestrado) Instituto de Ciências Matemáticas e de Computação da USP, 2007.

PFLEEGER, Shari Lawrence. **Engenharia de software:** teoria e prática. 2 ed. São Paulo, SP: Prentice-Hall, 2004.

PRESSMAN, Roger S. **Engenharia de software**, 6 ed. McGraw-Hill, 2006.

RAJLICH, Václav T.; BENNETT, Keith H. A staged model for the software life cycle. **Computer**, Los Alamitos, EUA, v. 33, n. 7, p. 66-71, jul. 2000.

SILVA, Ronan Assumpção; MATOS, Simone Nasser; SOUZA, João Umberto Furquim de; MOURA, Louisi Francis. **Manutenção de software nas empresas. Congresso Internacional de Administração**. 2010, p. 1-9.

SOMMERVILLE, Ian. **Software engineering**. 8 ed. Reading, EUA: Addison Wesley, 2007.

WEBSTER, Kenia P. Batista; OLIVEIRA, Kathia M. de; ANQUETIL, Nicolas. **A risk taxonomy proposal for software maintenance**. In: 21th IEEE International Conference on Software Maintenance (ICSM). Budapeste, Hungria: IEEE Computer Society, 2005. p. 453-461.