

FACULDADES INTEGRADAS DE ARARAQUARA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

RAFAEL STAIN CASSAU

SISTEMA DE GERENCIAMENTO DE OCORRÊNCIAS PARA O
DESENVOLVIMENTO DE APLICAÇÕES CORPORATIVAS - BUGSYS

Araraquara - SP

Novembro 2013

RAFAEL STAIN CASSAU

SISTEMA DE GERENCIAMENTO DE OCORRÊNCIAS PARA O
DESENVOLVIMENTO DE APLICAÇÕES CORPORATIVAS - BUGSYS

Trabalho de Conclusão de Curso
apresentado às Faculdades Integradas de
Araraquara, como requisito para obtenção
do Título de Bacharel em Sistema de
Informação.

Área de Concentração: Engenharia de
Software

Orientador: Me. Cristina Cibeli Vidotti Ivo de Medeiros

Araraquara - SP
Novembro 2013

RAFAEL STAIN CASSAU

SISTEMA DE GERENCIAMENTO DE OCORRÊNCIAS PARA O DESENVOLVIMENTO DE APLICAÇÕES CORPORATIVAS- BUGSYS

Trabalho de Conclusão de Curso
apresentado às Faculdades Integradas de
Araraquara, como requisito para obtenção
do Título de Bacharel em Sistema de
Informação.

Área de Concentração: Engenharia de
Software

Data de aprovação: 25/11/2013

MEMBROS COMPONENTES DA BANCA EXAMINADORA:

Orientador: Cristina Cibeli Vidotti Ivo de Medeiros
Título: Mestre em Sistemas Avançados de Computação
Universidade: Universidade Federal de São Carlos – UFSCar (São Carlos SP)

Prof. Fábio Papini Fornazari
Título: Doutor em Gestão de Educação Escolar
Universidade: Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP (Campus Araraquara SP)

Prof. André Luiz da Silva
Título: Mestrando em Educação Escolar
Universidade: Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP (Campus Araraquara SP)

Local: Faculdades Integradas de Araraquara

LOGATTI – Araraquara SP

AGRADECIMENTOS

Agradeço primeiramente a Deus, que me deu o privilegio de existir, me deu o folego de vida, saúde e capacidade para poder servi-lo em espirito e em verdade. Agradeço aos meus pais Marco Aurélio Cassau e Rute Stain Cassau que com muita dedicação e carinho me educaram e me ensinaram a ser um homem digno, honesto e justo, agradeço a minha futura esposa Caroline pela paciência que teve comigo por dedicar o nosso tempo de lazer para a realização desse trabalho e agradeço a meu grande amigo Michael pelo apoio técnico que me concedeu durante todo o desenvolvimento do trabalho.

O verdadeiro cidadão dos céus

*“SENHOR, quem habitará no teu tabernáculo?
quem morará no teu santo monte?*

*2 Aquele que anda em sinceridade, e pratica a justiça, e
fala verazmente, segundo o seu coração;*

*3 Aquele que não difama com a sua língua, nem faz mal
ao seu próximo, nem aceita nenhuma afronta contra o seu
próximo;*

*4 Aquele a cujos olhos o réprobo é desprezado; mas
honra os que temem ao Senhor; aquele que, mesmo que
jure com dano seu, não muda.*

*5 Aquele que não empresta o seu dinheiro com usura,
nem recebe peitas contra o inocente; quem faz isso nunca
será abalado”.*

RESUMO

O sistema que foi desenvolvido no decorrer deste projeto tem a finalidade de gerenciar as ocorrências encontradas no dia a dia de uma equipe de desenvolvimento de *software*. Entende-se por ocorrência, todo o fato gerador de algum impedimento para a realização do projeto, em sua grande maioria essas ocorrências são defeitos encontrados durante a fase de testes dos sistemas.

Palavras Chave: ocorrências, processos, projeto, gerenciamento, formalização.

ABSTRACT

The system that was developed during this project aims to manage occurrences found in daily from a team of software development. Understood to be occurring, the entire taxable year of any impediment to the realization of the project, mostly these occurrences are defects found during the testing phase systems.

Keywords: events, processes, project, management, formalization.

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelo em cascata	8
Figura 2 – Processo de desenvolvimento prototipação.....	9
Figura 3 – Processo de desenvolvimento incremental	10
Figura 4 – Processo de desenvolvimento ágil XP	12
Figura 5 – Processo de desenvolvimento ágil Scrum.....	15
Figura 6 – Troca de mensagens entre objetos	18
Figura 7 – Conceitos do paradigma orientado a objetos.....	20
Figura 8 – Diagramas da UML.....	21
Figura 9 - Diagrama de Classe.....	25
Figura 10 - Diagrama de Caso de Uso.....	28
Figura 11 - Diagrama de Sequencia.....	30
Figura 12 - Diagrama de Casos de Uso do Sistema	39
Figura 13 - Diagrama de Classes	47
Figura 14 - Diagrama de Sequência - Cadastro de Ocorrências.....	49
Figura 15 - Modelo Entidade-Relacionamento do Sistema	51
Figura 16 - Modelo Relacional do Sistema.....	53
Figura 17 - Tela de Autenticação do Sistema	56
Figura 18 - Tela de Listagem de Usuários do Sistema	57
Figura 19 - Tela de Cadastro de Usuários no Sistema	57
Figura 20 - Listagem de Clientes do Sistema	58
Figura 21 - Tela de Cadastro de Clientes do Sistema.....	58
Figura 22 - Cadastro de <i>Workflows</i> do Sistema	59
Figura 23 - Cadastro e Alteração de Projetos do Sistema.....	60
Figura 24 - Listagem de Ocorrências do Sistema	61
Figura 25 - Cadastro e Alteração de Ocorrências.....	62

LISTA DE TABELAS

Tabela 1 - Lista de Ocorrências	44
Tabela 2 - Preenchimento dos campos Ocorrência.....	45
Tabela 3 – Edição do Status da Ocorrência.....	45
Tabela 4 – Edição do Status e de Usuário da Ocorrência.....	46

LISTA DE SIGLAS

BUGSYS	Sistema de Gerenciamento de Ocorrências para Aplicações Corporativas
CASE	(<i>Computer-Aided Software Engineering</i>)
CMMI	(<i>Capability Maturity Model Integration</i>)
CSS	(<i>Cascading Style Sheets</i>)
HQL	(<i>Hibernate Query Language</i>)
HTML	(<i>Hyper Text Markup Language</i>)
HTTP	(<i>Hyper Text Transfer Protocol</i>)
JEE	(<i>Java Enterprise Edition</i>)
JSP	(<i>Java Server Pages</i>)
JVM	(<i>Java Virtual Machine</i>)
MPS-BR	(Melhoria de Processo de Software Brasileiro)
OO	(Orientação a Objetos)
ORM	(<i>Object Relational Mapping</i>)
PMI	(<i>Project Management Institute</i>)
SGBD	(Sistema Gerenciador de Banco de Dados)
SQL	(<i>Structured Query Language</i>)
UML	(<i>Unified Modeling Language</i>)
XP	(<i>Extreme Programming</i>)

SUMÁRIO

INTRODUÇÃO.....	3
1. REVISÃO BIBLIOGRÁFICA	5
1.1. Engenharia de <i>Software</i>	5
1.1.1. Definição	5
1.1.2. Modelos de Processo de <i>Software</i>	6
1.1.2.1. Modelo cascata.....	7
1.1.2.2. Modelo prototipação	8
1.1.2.3. Modelo incremental.....	9
1.1.2.4. Modelos Ágeis.....	10
1.1.2.4.1. XP - <i>Extreme Programming</i> (Programação Extrema).....	12
1.1.2.4.2. Scrum.....	14
1.1.2.5. Modelo Orientado a Objetos	16
1.1.3. UML – <i>Unified Modeling Language</i> (Linguagem de Modelagem Unificada)	21
1.1.3.1. Definição	21
1.1.3.2. Diagrama de Classes	24
1.1.3.3. Diagrama de Caso de Uso	25
1.1.3.4. Diagrama de Sequência	29
1.1.4. Qualidade de <i>software</i>	30
1.1.5. Projeto.....	33
2. PROJETO BUGSYS.....	37
2.1. Escopo do sistema	37
2.2. Levantamento de requisitos	37
2.3. Análise e projeto do sistema.....	38
2.4. Diagrama de Casos de Uso	39
2.5. Especificação de Caso de Uso	40
2.6. Diagrama de Classes.....	47
2.7. Diagrama de Sequência	49
2.8. Modelo Entidade-Relacionamento	51
2.9. Modelo Relacional.....	53
2.10. Desenvolvimento	54
2.11. Apresentação do Sistema.....	56
3. CONSIDERAÇÕES FINAIS	64

REFERÊNCIAS BIBLIOGRÁFICAS	66
----------------------------------	----

INTRODUÇÃO

Atualmente com a crescente evolução tecnológica e a necessidade de gerenciar informações, grandes empresas no ramo de desenvolvimento de sistemas surgem oferecendo soluções padrões para a informatização e desenvolvimento de sistemas sob demanda de acordo com a necessidade do cliente.

Porém como a maioria desses sistemas corporativos são grandes e complexos foram criados modelos de processos de *software* capazes de gerir todo o ciclo de vida do projeto, desde o levantamento de requisitos até a execução de testes e implantação.

Dentro desta perspectiva, se faz necessário a existência de um sistema de informação interno capaz de garantir que o modelo de processo de desenvolvimento escolhido seja seguido à risca. Tal sistema deve ser capaz de gerenciar os membros da equipe responsável pelo desenvolvimento do projeto, como também deve ser capaz de gerenciar problemas encontrados durante o desenvolvimento, registrando esses problemas em forma de ocorrências, podendo ser elas, erros, defeitos, falhas, melhorias, sugestões, solicitações de mudança entre outras.

Essas ocorrências em sua grande maioria são criadas internamente pela equipe de desenvolvimento, mas podem também ser criadas externamente pelo cliente no qual o projeto está sendo desenvolvido.

Dado a origem do problema, foi desenvolvido um sistema que é capaz de gerenciar as ocorrências geradas durante o desenvolvimento de sistemas de informação, sejam essas ocorrências cadastradas externamente pelos clientes, ou internamente por testadores, desenvolvedores, analista de sistemas, analista de requisitos, arquitetos, ou qualquer membro relacionado ao desenvolvimento do projeto.

O *Bugsys* (Sistema de Gerenciamento de Ocorrências para Aplicações Corporativas) foi desenvolvido para a *web* e suas principais funcionalidades consistem em gerenciar os membros da equipe (usuários) do sistema, manter os projetos e seus respectivos casos de uso, manter os clientes relacionados aos projetos, gerenciar *workflows* responsáveis pelas fases do modelo de processo de desenvolvimento, gerenciar o ciclo de vida das ocorrências geradas nessas fases que consistem em abrir, fechar, aceitar, rejeitar, corrigir, validar, não validar, homologar ou suspender ocorrências.

O trabalho está organizado sob a forma de itens, onde no item 1 é realizado um estudo e levantamento bibliográfico sobre a engenharia de *software* e seus principais tópicos relacionados ao desenvolvimento de sistemas.

No item 2 foi desenvolvido o projeto em si como parte prática do trabalho de conclusão de curso, onde foram aplicados os conceitos estudados no item 1, bem como os conhecimentos adquiridos durante o curso e toda a experiência adquirida no mercado de trabalho.

No item 3 são apresentadas as considerações finais.

1. REVISÃO BIBLIOGRÁFICA

1.1. Engenharia de *Software*

1.1.1. Definição

Atualmente estamos vivenciando uma nova era, a era da informação, aonde a disseminação da informação é muito rápida e os meios de acesso a ela são variados e de fácil manipulação, sendo assim a informática teve que evoluir de maneira ágil, e para que essa evolução acontecesse com qualidade, houve a necessidade de definir metodologias e paradigmas para que o ciclo de vida do *software*, que é responsável por gerir a informação, se tornasse algo consistente.

Nos dias de hoje o termo *software* é facilmente associado a um simples programa de computador capaz de fornecer uma solução específica para um determinado problema, mas na verdade *software* “não é apenas um programa, mas também toda a documentação associada com um conjunto de informações de configuração necessária para fazer com que esses programas operem corretamente”. (SOMMERVILLE, 2004, pg. 5).

O desenvolvimento de grandes sistemas de informação começou a ser difundido em meados de 1970, nessa época o desenvolvimento de *software* ocorria com pouco ou nenhum planejamento, paradigma ou modelo de processo a ser seguido, não se pensava ou planejava como fazer, apenas era feito, sendo assim os custos para manter esses sistemas operando era muito alto, pois muitas vezes uma empresa desenvolvia o sistema e outra empresa era responsável por manter o sistema funcionando, mas isso era praticamente impossível sem documentos (artefatos) de como o *software* havia sido desenvolvido, os gastos eram muito além dos estimados e os resultados obtidos eram muito inferior aos resultados esperados.

Essa época em meados de 1970 ficou conhecida como a época da crise do *software*, devido ao grande investimento e pouco resultado a idéia de informatizar os negócios estava sendo deixada de lado, até que então começaram a pensar e definir padrões e modelos de processo de *software* para que os custos com desenvolvimento e principalmente com manutenção diminuíssem, nessa época a engenharia de *software* começou a surgir.

A engenharia de *software* surgiu com intuito de definir um conjunto de padrões, paradigmas e principalmente processos capazes de definirem a

construção e o ciclo de vida de sistemas de informação. Pressman define engenharia de *software* como:

Engenharia de *software* é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter *software* de maneira econômica que seja confiável e funcione de forma eficientemente em máquinas reais. (PRESSMAN, 2011, pg.39).

É importante aplicar todos os fundamentos da engenharia de *software* a fim de obter qualidade dos *softwares* desenvolvidos, a grande vantagem de sua utilização é que através da aplicação de seus fundamentos é possível garantir qualidade e longevidade dos *softwares*, é importante frisar que garantir qualidade do *software* não é como garantir qualidade de qualquer outro produto, pois *software* é um produto abstrato e intangível.

A fim de que esses sólidos fundamentos de engenharia sejam aplicados ao desenvolvimento de sistemas, processos de *software* foram criados.

1.1.2. Modelos de Processo de *Software*

Existem vários modelos de processo de *software*, porém cada um deles implementa de maneira específica, um conjunto de etapas (fases) que definem de maneira mais genérica o que é um processo de *software*.

Segundo Pressman “um processo de *software* é definido como uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um *software* de alta qualidade”. (PRESSMAN, 2011, pg. 52)

Pressman (2011, pg. 40) segue dizendo que as principais etapas que definem o processo genérico de *software* são:

- **Comunicação** - Fase responsável por analisar o problema juntamente com as partes interessadas para que o mesmo seja compreendido e para que todos os objetivos traçados sejam alcançados, essa fase é conhecida como a fase de levantamento de requisitos.
- **Planejamento** - Fase responsável por traçar uma rota para que os objetivos traçados na fase anterior sejam concluídos com eficiência, nessa fase é elaborado um plano de projeto contendo os riscos e os

recursos necessários para garantir a construção do *software*, essa fase é conhecida como a fase de projeto.

- **Modelagem** - Fase responsável por definir através de *design* de modelos como o produto de *software* será implementado e como seus componentes internos irão interagir entre si, esta etapa se torna necessária, pois nela começa a ser concretizado o que foi definido nas etapas anteriores, essa fase é conhecida como a fase de análise.
- **Construção:** Nessa etapa é que o desenvolvimento do *design*, modelos e arquiteturas definidas na fase anterior são construídos, sendo com código automatizado ou código manual e nessa etapa também é realizado os testes das funcionalidades implementadas, essa fase é conhecida como a fase de desenvolvimento e testes.
- **Emprego:** Nessa etapa o *software* construído inteiramente ou parcialmente é entregue ao cliente que utiliza e avalia o produto retornando o seu *feedback*, de maneira que correções, alterações ou novas funcionalidades são pedidas dando continuidade ao ciclo de vida do *software*, essa fase é conhecida como a fase de implantação.

Existem vários modelos de processo de *software*, sendo os principais descritos abaixo:

1.1.2.1. Modelo cascata

O Modelo de processo em cascata foi o primeiro modelo de processo de *software* criado, e sua definição prega que apenas uma etapa de cada vez deve ser seguida, sendo assim seria necessário terminar a primeira etapa para dar início a segunda etapa e assim sucessivamente até o término do processo.

O modelo cascata, algumas vezes chamado de ciclo de vida clássico, sugere uma abordagem sequencial e sistemática para o desenvolvimento de *software*, começando com o levantamento de necessidades por parte do cliente, avançando pelas fases de planejamento, modelagem, construção, emprego e culminando no suporte contínuo do *software* concluído. (PRESSMAN 2011, pg. 59)

A figura 1 representa um modelo em cascata:

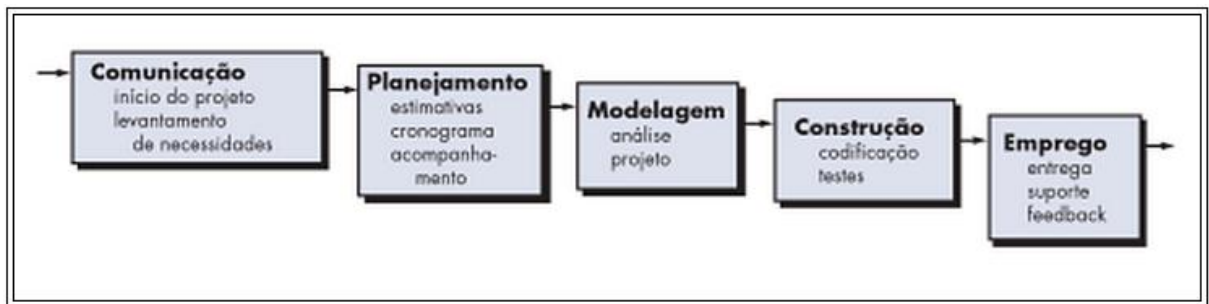


Figura 1 - Modelo em cascata
Fonte: Pressman 2011, pg. 60.

É perceptível que o modelo em cascata não é o modelo de processo mais adequado para os dias atuais, pois hoje em dia é exigida alta produtividade, e para que tal produtividade ocorra seria mais viável usar um modelo de desenvolvimento que permita que as etapas do processo sejam realizadas paralelamente, evitando assim problemas decorrentes de falhas em fases superiores o que não seria viável caso o modelo de desenvolvimento escolhido fosse o modelo em cascata.

1.1.2.2. Modelo prototipação

O modelo de Processo Prototipação diferentemente do modelo em cascata permite a execução de atividades paralelas nas fases do processo, pois ele é baseado em um modelo evolutivo e abrange o processo como um todo e não por etapas.

Sommerville explica que:

Os clientes e usuários finais de *software* acham muito difícil expressar seus reais requisitos. É quase impossível prever como um sistema afetará práticas de trabalho, como interagirá com outros sistemas e que operações dos usuários devem ser automatizadas. A análise cuidadosa e as revisões sistemáticas de requisitos ajudam a reduzir as incertezas sobre o que o sistema deve fazer. Contudo não há um substituto real para experimentar um requisito antes de concordar com ele. Isso será possível se um protótipo de sistema estiver disponível. (SOMMERVILLE, 2004, pg. 145)

A figura 2 representa um processo de desenvolvimento prototipação:

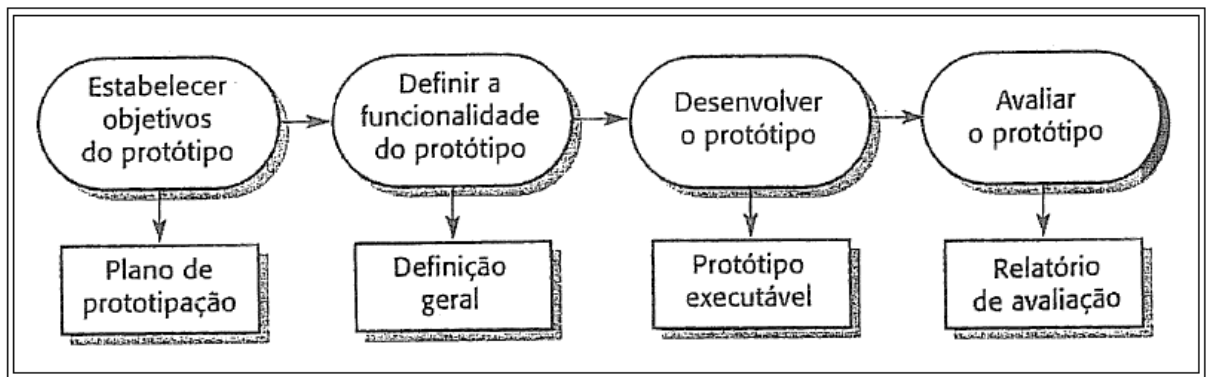


Figura 2 – Processo de desenvolvimento prototipação

Fonte: Sommerville 2004, pg. 146.

O processo é abrangido como um todo, pois para solucionar o problema de desenvolvimento sequencial e permitir a alta produtividade é fornecido para o cliente um protótipo, que deve ser validado, aceito a sua funcionalidade, será implementada. Após o término dessa funcionalidade pequenas alterações são realizadas, são feitos mais protótipos que serão sucessivamente implementados fazendo com que o processo seja contínuo como um todo. A grande vantagem desse modelo de processo é que com ele o cliente consegue realmente saber aquilo que ele precisa sem pedir funcionalidades a mais e nem deixar de possuir recursos importantes para o seu sistema, mas a principal desvantagem desse modelo de processo é que caso o cliente seja muito leigo, vendo apenas um protótipo ele poderá pensar que seu sistema esteja pronto ou que a implementação do mesmo não demande muito esforço e tempo.

1.1.2.3. Modelo incremental

Pressman (2011, pg. 61) fala que o modelo incremental combina elementos dos fluxos que deveriam ser seguidos sequencialmente como no modelo em cascata e fluxos que podem ser executados paralelamente como no modelo prototipação de maneira organizada de acordo com a evolução e o tempo. Cada sequência gera fluxos incrementais podendo ser esses entregáveis, aprovados ou liberados de maneira similar aos incrementais gerados por um fluxo de processos evolucionários como o modelo de processo prototipação.

Pressman (2011, pg. 62) segue dizendo que quando o modelo incremental é utilizado, frequentemente o primeiro incremento a ser implementado

é um produto essencial, tendo somente as principais funcionalidades para o seu funcionamento, sem implementar fluxos alternativos que para aquela entrega não seria necessário. Geralmente esse primeiro produto é avaliado de maneira detalhada ou já é diretamente usado pelo cliente, que após ter uma prévia do produto já demandará o próximo incremento, incluindo melhorias e fluxos alternativos do incremento anterior, seguindo essa ordem até a entrega do produto final.

O principal foco do modelo de processo Incremental é definir um produto funcional a cada entrega, mesmo que este não esteja totalmente pronto.

A figura 3 representa um processo de desenvolvimento incremental:

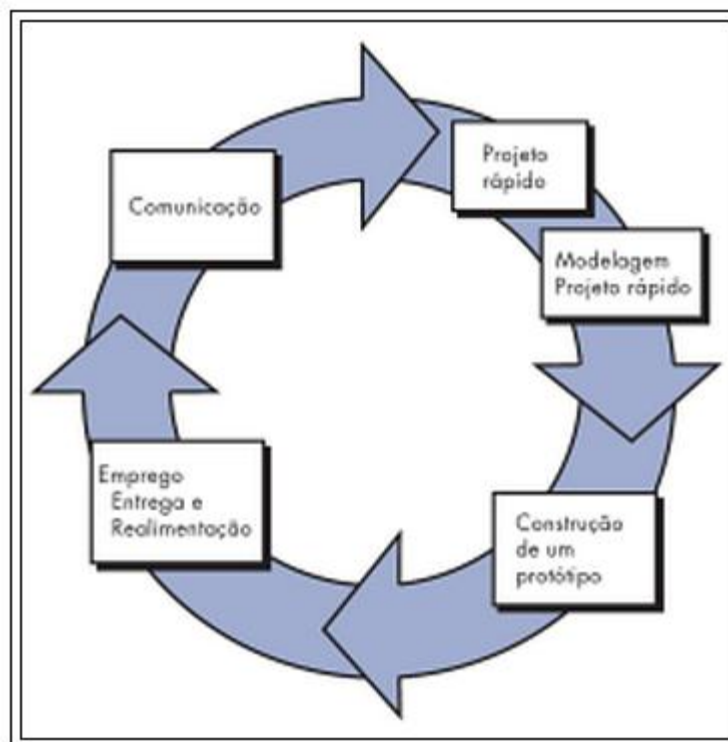


Figura 3 – Processo de desenvolvimento incremental

Fonte: Pressman 2011, pg. 63.

1.1.2.4. Modelos Ágeis

Com o aumento crescente das tecnologias e plataformas de desenvolvimento de sistemas, o ciclo de vida dos *softwares* tende a ser cada vez mais curto, fazendo com que quem deseja se estabelecer no mercado inove com maior frequência os seus produtos e serviços, as condições e necessidades do

mercado atual mudam diariamente fazendo com que as necessidades dos clientes sejam alteradas constantemente.

É evidente que o tempo de construção de um sistema utilizando modelos de processos tradicionais é muito superior ao tempo de construção de um sistema que possui o seu foco principal na entrega contínua de funcionalidades para o cliente e não exija tanta documentação quanto um modelo tradicional exigiria.

Modelos de processos ágeis funcionam dessa forma, possuem o foco principal na entrega contínua de funcionalidades ao cliente, minimizando ao máximo as documentações que segundo seus idealizadores não julgam ser tão necessárias.

PRESSMAN (2011, pg. 82) segue dizendo que “uma das características mais convincentes da abordagem ágil é a sua habilidade de reduzir os custos da mudança ao longo de todo o processo de *software*.”

Os modelos de processo ágeis surgiram com o objetivo de solucionar os problemas não previstos pela engenharia de *software* tradicional.

Em essência, métodos ágeis se desenvolveram em um esforço para sanar fraquezas reais e perceptíveis da engenharia de *software* convencional. O desenvolvimento ágil oferece benefícios importantes, no entanto, não é indicado para todos os projetos, produtos, pessoas e situações. Também não é a antítese da prática de engenharia de *software* consistente e pode ser aplicado como uma filosofia geral para todos os trabalhos de *software*. (PRESSMAN, 2011, pg. 82)

Existem alguns princípios que definem agilidade, dentre eles os mais importantes segundo Pressman (2011, pg. 84) são:

- A maior prioridade sempre é satisfazer o cliente lhe entregando de forma adiantada e continua aquilo que para ele é vital e valioso, o seu produto de *software*.
- Todos os pedidos de mudança devem ser bem acolhidos, mesmo que isso acarrete em um atraso no desenvolvimento, pois os processos ágeis se aproveitam das mudanças como uma vantagem competitiva para estreitar sua relação com o cliente.
- Os projetos devem ser construídos em torno de indivíduos motivados, que tenham ambiente e apoio necessário, e que se sintam capazes de atender as necessidades da empresa.

- A maneira mais eficiente de transmitir informação para a equipe de desenvolvimento e com uma conversa aberta de forma presencial.
- *Software* em funcionamento é a principal medida do progresso.
- Simplicidade é essencial
- Os melhores requisitos, arquitetura e desenvolvimento sempre estão presentes nas equipes que se auto organizam e que tenham um intervalo para se auto avaliar e trabalhar em constante sintonia.

Para que fique um pouco mais claro como funciona os processos ágeis, segue abaixo uma visão geral de dois dos principais processos ágeis utilizados nos dias de hoje.

1.1.2.4.1. XP - *Extreme Programming* (Programação Extrema)

Programação extrema é o que prega esse modelo de processo de desenvolvimento em que seu principal foco é a entrega contínua de funcionalidades, sendo documentado somente o mínimo necessário para o entendimento e futuras manutenções, aplicando todo o seu esforço naquilo que realmente é importante para o cliente, o produto final.

A *Extreme Programming* (programação extrema) emprega uma abordagem orientada a objetos como seu paradigma de desenvolvimento preferido e envolve um conjunto de regras e praticas constantes no contexto de quatro atividades metodológicas: planejamento, projeto, codificação e testes. (PRESSMAN, 2011, pg. 88).

A figura 4 representa um processo de desenvolvimento ágil XP:

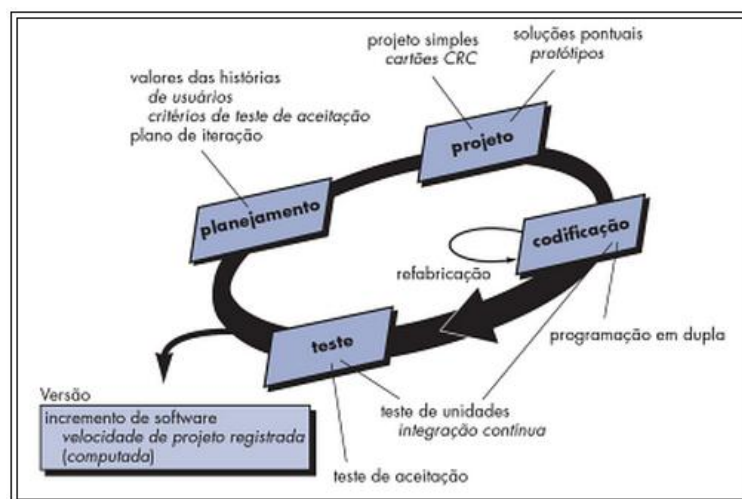


Figura 4 – Processo de desenvolvimento ágil XP

Fonte: Pressman 2011, pg. 88.

A XP baseia-se em uma abordagem de “quatro pilares”, sendo eles planejamento, projeto, codificação e testes.

A etapa de planejamento é responsável por executar uma série de tarefas relacionadas ao entendimento do problema, essa série de tarefas é iniciada com a atividade de ouvir, que faz parte do levantamento de requisitos que capacita a equipe a entender o negócio e suas principais funcionalidades.

O levantamento de requisitos é realizado na atividade de escrita de historias, sendo essa atividade responsável por descrever as funcionalidades do sistema e suas características, cada historia é colocada em uma ficha, e em seguida é atribuído um valor a essa ficha, esse valor descreve a prioridade referente a essa historia, após o termino da escrita da historia a equipe mede o esforço para realizar o desenvolvimento das historias contidas nessa ficha, sendo esse esforço medido por tempo, caso uma ficha passe de três semanas para ser desenvolvida então é sugerido que a mesma seja dividida em mais historias.

Já a etapa de projeto tem como seu principal foco a simplicidade, é sempre preferível uma representação simples e enxuta de um projeto do que uma representação complexa e de difícil interpretação, a atividade de projeto fornece um guia de implementação para uma historia, nada além disso.

Caso seja identificado um problema na etapa de projeto é recomendada a criação imediata de um protótipo operacional dessa parte do projeto, outra característica importante é a refabricação que na verdade é a continua refatoração do código, fazendo com que dessa forma a etapa de projeto ocorra continuamente devido ao alto número de refatorações que são realizadas durante o desenvolvimento.

Na etapa de codificação, diferente de outros modelos de processo, não é iniciado o desenvolvendo das funcionalidades, mas sim o desenvolvimento de um conjunto de testes unitários, para que os objetivos e as regras de negócio fiquem mais claras e simples de serem desenvolvidas.

Após o término da construção de testes unitários é dado o inicio no desenvolvimento das funcionalidades, uma das características mais importantes pregadas pela XP e a codificação em dupla, já que duas cabeças sempre pensam melhor do que uma, e enquanto um desenvolve o outro membro da dupla revisa o que esta sendo desenvolvido.

A etapa de testes é executada de maneira contínua e eficaz, já que na etapa de codificação foi construído vários testes unitários automatizados, dessa maneira fica muito mais prático e rápido a execução de testes regressivos, pois para serem executados não haverá muito esforço, já os testes de aceitação do cliente ocorrem de maneira rápida e eficaz, pois os casos de uso são simples e pequenos para serem testados.

Segundo Teles (2006, pg. 23) o XP se baseia nas seguintes práticas:

- Cliente Presente
- Jogo do Planejamento
- *Stand Up Meeting* – Reuniões em pé
- Programação em Par
- Desenvolvimento Guiado pelos Testes
- *Refactoring* - Refatoração
- Código Coletivo
- Código Padronizado
- *Design* Simples
- Metáfora
- Ritmo Sustentável
- Integração Contínua
- Releases Curtos

Essas praticas fazem com que o XP se torne cada vez mais adotado pelas equipes de desenvolvimento de *software*, pois devido a execução dessas praticas o desenvolvimento de *software* com XP se torna cada vez mais efetivo.

1.1.2.4.2. Scrum

O Scrum assim como a XP mantém o seu foco no produto final minimizando ao máximo as documentações e enfatizando muito o trabalho e o entrosamento entre os membros da equipe.

Os princípios do Scrum são consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades estruturais: requisitos, análise, projeto, evolução e entrega. (PRESSMAN, 2011, pg. 88).

A figura 5 representa um processo de desenvolvimento ágil Scrum:

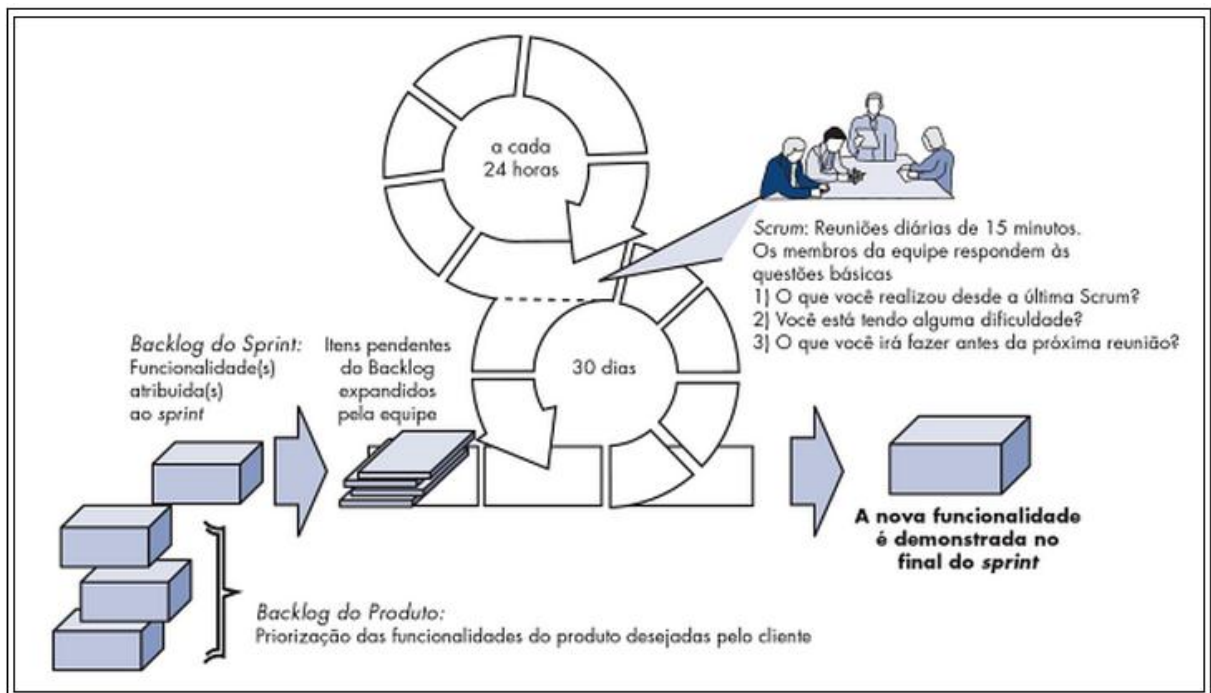


Figura 5 – Processo de desenvolvimento ágil Scrum

Fonte: Pressman 2011, pg. 96.

No Scrum, as atividades metodológicas são divididas em um conjunto de tarefas conhecido como *sprint*, o número de *sprints* necessários para concluir uma atividade varia muito de acordo com o nível de complexidade do produto, e é muitas vezes alterado em tempo real pela equipe Scrum.

Pressman (2011, pg. 95) descreve que o Scrum provou ser capaz de gerenciar projetos com prazos de entrega curtos, requisitos mutáveis e regras de negócio complexas, tudo isso graças a um conjunto de padrões de processo de *software* descrito abaixo:

- **Registro pendente de trabalho (*Backlog*)** – define uma lista de prioridades de funcionalidades que devem ser desenvolvidas, pois oferecem um valor comercial ao cliente.
- **Urgências (*sprints*)** – consiste de um conjunto de tarefas (funcionalidades) necessárias para que um determinado requisito estabelecido no *backlog* seja atingido.

- **Alterações** – As alterações não são introduzidas durante a execução das atividades de urgência, pois o *sprint* permite que os membros da equipe trabalhem em um ambiente estável e de curto prazo.
- **Reuniões** – As reuniões são realizadas todos os dias e duram em média quinze minutos, quem as dirige é um membro da equipe chamado *Scrum Master*. Geralmente existem três perguntas chaves que devem ser respondidas por todos os membros da equipe, sendo elas:
 1. O que foi realizado desde a última reunião com a equipe?
 2. Quais as dificuldades que está encontrando?
 3. O que planeja realizar até que a próxima reunião ocorra?

Após o término de uma tarefa, é entregue ao cliente uma *demo*, que tem como objetivo demonstrar as funcionalidades desenvolvidas, para que as mesmas possam ser avaliadas pelo cliente. É importante frisar que nem todas as funcionalidades planejadas podem estar concluídas nessa etapa, mas sim as funções que podem ser entregues no prazo estipulado, enfatizando assim a entrega mesmo que parcialmente.

Modelos de processo de *software* são capazes de gerenciar e organizar o desenvolvimento de sistemas, mas para que seja utilizado concretamente é necessário a adoção de um paradigma de programação que fornece maneiras de aplicar o processo no desenvolvimento do *software*, atualmente a maioria dos processos de *software* usam como seu paradigma a orientação a objetos.

1.1.2.5. Modelo Orientado a Objetos

Esse modelo consiste em aplicar sólidos conhecimentos de engenharia através do paradigma orientado a objetos a fim de obter *software* de qualidade.

Bezerra (2007, pg. 5) define que o paradigma orientado a objetos é uma forma de abordar o problema e pensar em uma solução baseada nesse modelo, abstraindo a complexidade do problema em objetos do mundo real, e fazendo com que esses objetos se comuniquem entre si através de mensagens.

Abaixo é descrito os princípios do paradigma orientado a objetos:

Qualquer coisa é um objeto.
Objetos realizam tarefas por meio da requisição de serviços a outros objetos.
Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
A classe é um repositório para comportamento associado ao objeto.
Classes são organizadas em hierarquias. (BEZERRA, 2007, pg. 5).

Objeto, no mundo real pode ser definido por qualquer coisa que em seu contexto possua características e comportamentos que possam descrevê-lo, por exemplo, um carro.

Um carro possui características sendo algumas delas, cor, ano, modelo, marca, motor, entre outras, carro possui também alguns comportamentos, entre eles, ligar, desligar, acelerar, frear, trocar marcha, etc.

Classe, no contexto de sistemas pode ser definida por um modelo responsável por descrever as características e comportamentos de seus objetos, um objeto é a concretização de um modelo definido em uma classe. Essa classe é capaz de construir vários objetos baseados em seu modelo, porém cada objeto terá o seu próprio estado em memória.

Sendo assim um carro pode ser descrito através de sua classe pelo paradigma orientado a objetos. Essa classe será capaz de criar vários carros com cores, anos, modelos, marcas, motores entre outras características diferentes e maneiras diferentes de ligar, desligar, acelerar, frear, trocar macha entre outros comportamentos específicos.

Na terminologia da orientação a objetos, cada ideia é denominada classe de objetos, ou simplesmente classe. Uma classe é a descrição dos atributos e serviços comuns a um grupo de objetos. Sendo assim, pode-se entender uma classe como sendo um molde a partir do qual objetos são construídos. Ainda sobre terminologia, diz-se que um objeto é uma instância de uma classe. (BEZERRA, 2007, pg. 7).

Um sistema é formado por um conjunto de objetos interagindo entre si através de mensagens, as mensagens são chamadas que um objeto faz ao outro objeto através de seus comportamentos, no mundo real essas chamadas podem ser interpretadas como estímulos, assim como um ser humano recebe estímulos de qualquer outra entidade ao seu redor e os responde, um objeto é capaz de pedir a outro objeto que execute um comportamento específico através de uma troca de mensagem, podendo essa chamada ser respondida ou não de acordo

com a implementação do comportamento que está sendo executado no outro objeto.

A figura 6 representa a troca de mensagens entre objetos:

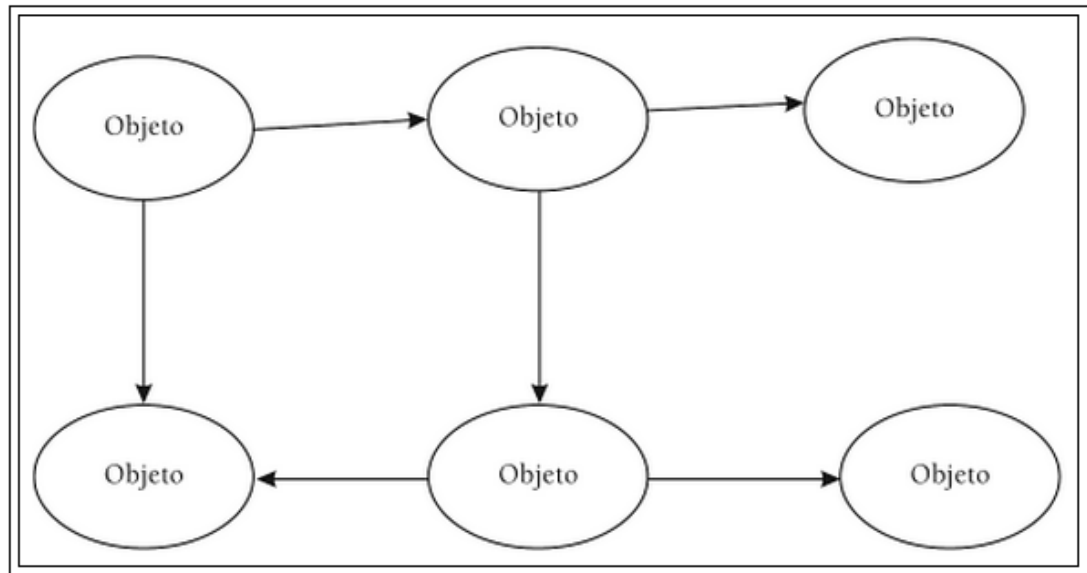


Figura 6 – Troca de mensagens entre objetos

Fonte: Bezerra 2007, pg. 8.

Quando se diz na terminologia de orientação a objetos que objetos de um sistema estão trocando mensagens significa que esses objetos estão enviando mensagens uns aos outros com o objetivo de realizar alguma tarefa dentro do sistema no qual eles estão inseridos.
(BEZERRA, 2007, pg. 8)

O modelo orientado a objetos é amplamente difundido e utilizado, pois diferente do paradigma estrutural que utiliza de uma abordagem mais complexa para analisar os problemas, o paradigma orientado a objetos utiliza objetos do cotidiano para modelar e desenvolver soluções, porém essa não é a única vantagem desse paradigma.

Bezerra (2007, pg. 6) diz que a orientação a objetos possui alguns conceitos principais, sendo eles:

- **Abstração** – “A abstração é um processo mental pelo qual nós seres humanos nos atentamos aos aspectos mais importantes (relevantes) de alguma coisa, ao mesmo tempo que ignoramos os aspectos menos importantes.” (BEZERRA, 2007, pg. 8).

A abstração no paradigma orientado a objetos ocorre quando uma classe declarada como abstrata é implementada de maneira que

somente os comportamentos e características mais relevantes da mesma são descritos de forma genérica para que posteriormente esses comportamentos possam ser sobrescritos por classes que possuem comportamentos e características em comum através do conceito de herança. As classes abstratas não podem ser concretizadas em memória, somente podem ser herdadas.

- **Herança** – A herança no paradigma orientado a objetos ocorre quando uma classe é declarada como abstrata ou não a fim de descrever comportamentos e características em comum para um determinado conjunto de objetos. Permitindo que posteriormente essas características possam ser reutilizadas e esses comportamentos possam ser sobrescritos por classes específicas que compartilham características e comportamentos em comum.

O conceito de abstração ocorre da mesma maneira na herança, pois ambas usam a generalização, porém o que diferencia uma classe genérica não abstrata de uma classe genérica abstrata tecnicamente é que uma classe genérica não abstrata, pode ser concretizada na memória enquanto uma classe genérica abstrata somente pode ser herdada por classes que compartilham as mesmas características e comportamentos.

- **Polimorfismo** – O polimorfismo indica a capacidade de implementar de maneira específica o comportamento genérico definido em uma classe que está sendo herdada, em outras palavras, o polimorfismo indica a capacidade de abstrair várias implementações diferentes em um único comportamento genérico. Isso só é possível graças aos conceitos de abstração e herança.
- **Encapsulamento** – Bezerra (2007, pg. 9) segue dizendo que o encapsulamento consiste em restringir o acesso referente à implementação de um comportamento que está sendo chamado pelo objeto requisitante, ou seja, o encapsulamento é responsável por não permitir que o objeto requisitante conheça detalhes da implementação do comportamento que está definido no objeto requisitado, apenas o

objeto requisitado deve conhecer a implementação de seu comportamento e deve fornecer uma *interface* de uso para que outros objetos requisitantes a use.

Com objetos bem encapsulados é possível criar classes menos complexas com poucos comportamentos e características específicas a fim de compor comportamentos entre elas, e promover constante reutilização de código.

A figura 7 representa os principais conceitos do paradigma orientado a objetos:

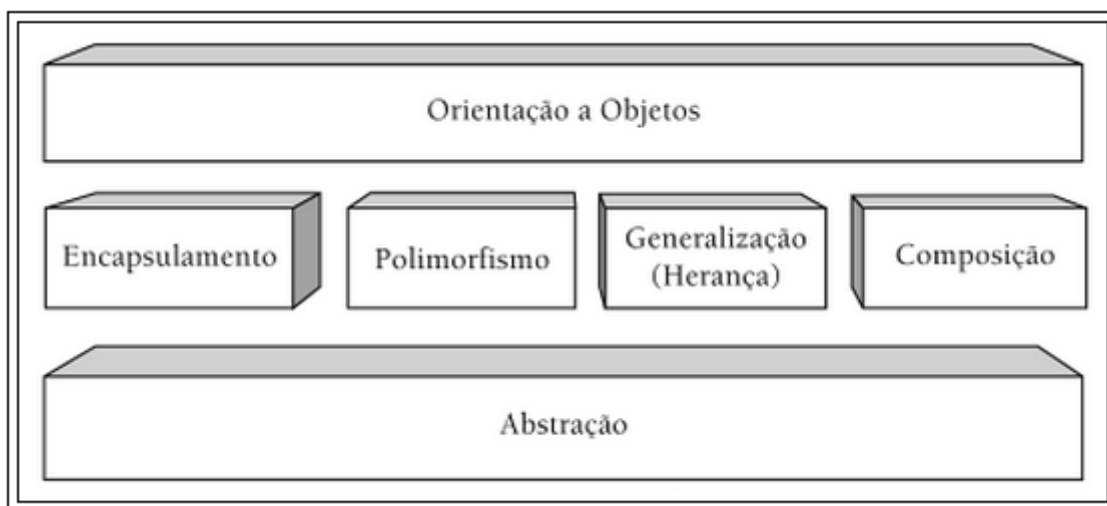


Figura 7 – Conceitos do paradigma orientado a objetos
Fonte: Bezerra 2007, pg. 9.

O modelo orientado a objetos proporciona grandes vantagens em relação ao seu antecessor, o modelo estruturado, porém para que *softwares* possam ser desenvolvidos e documentados utilizando esse paradigma é necessário que existam maneiras de modelar e documentar esses sistemas, devido a essa necessidade uma linguagem de modelagem de dados unificada foi criada.

1.1.3. UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada)

1.1.3.1. Definição

Lima (2011, pg. 33) Diz que a UML é um conjunto de ferramentas gráficas que individualmente são capazes de modelar sistemas de informação de varias maneiras diferentes. Atualmente a UML está na versão 2.3, e possui um conjunto de 15 diagramas, cada um responsável por modelar um aspecto específico do sistema.

UML (*Unified Modeling Language*) é uma família de notações gráficas, apoiada por um metamodelo único, que ajuda na descrição e no projeto de sistemas de *software*, particularmente daqueles construídos utilizando o estilo orientado a objetos (OO). (FOWLER, 2005, pg. 25)

A figura 8 representa a classificação dos tipos de diagrama da UML:

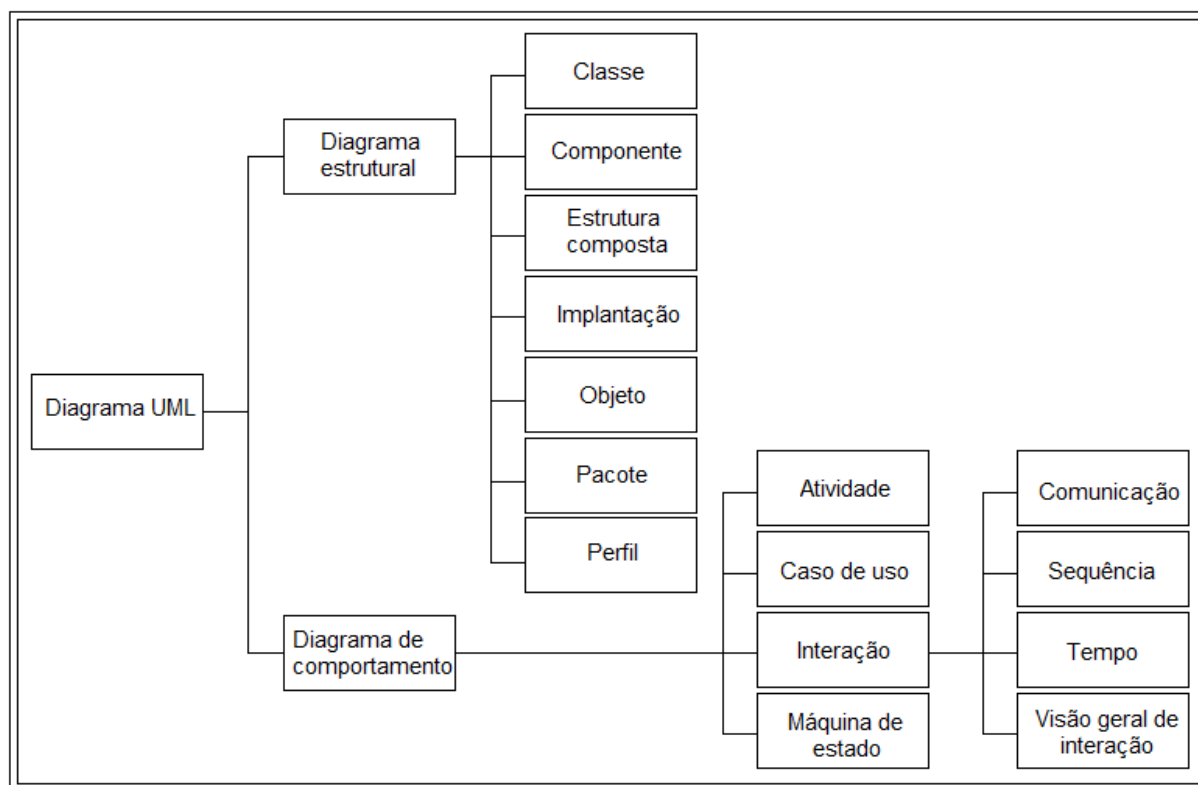


Figura 8 – Diagramas da UML

Fonte: Lima 2011, pg. 34

As linguagens gráficas existem há muito tempo na indústria de *software*, o principal motivo para a sua existência é que as linguagens de programação que são utilizadas no desenvolvimento não estão em um nível de abstração suficiente

para facilitar as discussões sobre projeto. Sendo assim, analisando um código fonte não é possível compreender de uma maneira geral a sua função dentro de um sistema.

Outro motivo que levou a criação da UML é que com a linguagem formal a descrição de um sistema se torna muito complexa, pois a linguagem formal pode ser ambígua, redundante, e não possui ferramentas para a escrita clara e objetiva de funcionalidades de um sistema de informação.

Todo esse conjunto de necessidades levou a criação de modelos de diagramas separados de modelagem que posteriormente se uniram e se tornaram unificados. A utilização da UML dentro do processo de desenvolvimento de *software* ocorre de várias maneiras diferentes, porém segundo Fowler (2005, pg. 25) as mais utilizadas são:

- **Esboço:** Fowler (2005, pg. 26) diz que um esboço geralmente é utilizado pelos desenvolvedores para ajudar a transmitir alguns aspectos do sistema através da UML, esses esboços podem ser utilizados em diferentes etapas do processo de desenvolvimento.

Na etapa de projeto um esboço pode ser utilizado a fim de apresentar um problema para os membros da equipe que posteriormente irão propor soluções a serem projetadas.

Na etapa de desenvolvimento, um esboço geralmente é construído antes de realizar a implementação a fim de transmitir ideias e alternativas sobre o que está prestes a ser feito, não é falado sobre todo o código fonte a ser desenvolvido, mas sim sobre questões importantes que devem ser compreendidas antes de iniciar o desenvolvimento.

Na engenharia reversa um esboço é geralmente utilizado para explicar o funcionamento de uma parte do sistema já em funcionamento, não é construído um esboço para representar cada classe que compõe essa funcionalidade, mas sim os aspectos mais importantes e complexos, antes de se aprofundar no código fonte.

Os esboços também são muito utilizados como documentos, pois por serem informais e dinâmicos não necessitam de perfeição em sua escrita, sendo assim são amplamente utilizados para facilitar a comunicação.

- **Projeto:** Fowler (2005, pg. 27) diz que a UML como projeto entra em contraste com o esboço, pois a UML como projeto prega a completeza, ou seja, a utilização de todas as ferramentas proporcionadas pela UML a fim de modelar um projeto de *software* completo e minuciosamente detalhado no sentido que todas as decisões estejam expostas para que o programador possa segui-las como uma atividade simples e direta sem muitas considerações.

Já quando um projeto deve ser modelado para um sistema em funcionamento o processo se torna muito mais complexo do que a criação de esboços, pois essas classes que compõe o sistema devem ser bem especificadas, para obter êxito nesse processo geralmente ferramentas CASE (Engenharia de *Software* Auxiliada por Computador) são utilizadas.

As ferramentas CASE são capazes de realizar a engenharia reversa de maneira automatizada, ou seja, com um *software* especializado é possível obter toda a documentação de um sistema já em funcionamento através de seu código fonte, existem diversas outras funcionalidades disponíveis nas ferramentas CASE que auxiliam na geração da documentação e modelagem de sistemas.

A UML 2.3 disponibiliza três gêneros diferentes de diagramas, sendo eles:

- **Diagramas de Estrutura** - Esses diagramas descrevem elementos estáticos do sistema que não alteram com o tempo, exemplo, descrevem o relacionamento entre entidades do sistema.
- **Diagramas de Comportamento** - Esses diagramas são responsáveis por descrever características comportamentais do sistema e seus processos de negócio, funcionalidades.
- **Diagramas de Interações** - Esses diagramas são responsáveis por descrever a maneira de como os objetos interagem entre si.

1.1.3.2. Diagrama de Classes

O diagrama de classes é um dos mais conhecidos e usados entre os diagramas que compõe a UML, ele faz parte do conjunto de diagramas estruturado e sua principal função é descrever as características e comportamentos das classes e também mapear os relacionamentos estáticos entre elas.

Um diagrama de classes descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos existentes entre eles. Os diagramas de classe também mostram as propriedades e as operações de uma classe e as restrições que se aplicam à maneira como os objetos estão conectados. (FOWLER 2005, pg. 52)

Atributos ou propriedades correspondem às características estruturais de uma classe, ou valores que objetos construídos a partir dessas classes são capazes de armazenar em seu estado na memória.

Métodos são comportamentos de ação descritos na classe que seus objetos serão capazes de executar quando possuírem um estado em memória.

Fowler (2005, pg. 52) segue dizendo que todos os atributos assim como os métodos possuem visibilidade, ou restrições de acesso para permitir que seus estados não sejam compartilhados com os demais objetos reduzindo assim o risco de mau uso de um atributo ou método indevido a outro objeto.

Existem notações responsáveis por definir a generalização de uma classe, a generalização define graficamente um modelo relacionado à herança em que uma ou muitas classes herdaram de uma classe mais genérica.

É possível definir também o seu nível de agregação que pode ser do tipo simples ou composta, a agregação simples relaciona um objeto ao outro, porém garantindo que o objeto dependente continue existindo caso o objeto que ele esteja sendo relacionado deixe de existir, um bom exemplo seria imaginar um objeto produto que esteja relacionada com uma venda, sendo assim uma venda tem pelo menos um ou muitos produtos, mas se a venda não existir o produto continuará existindo. Já no caso de agregação composta não faz sentido um objeto dependente existir se o objeto que ele esteja sendo relacionado deixe de existir, exemplo, um objeto pedido que esta relacionado com uma venda, caso a venda seja cancelada não faz sentido existir o pedido.

Também é definido no diagrama de classes a multiplicidade entre os objetos, ou seja, qual é o tipo do relacionamento entre os objetos, que pode variar, de zero a um, zero a muitos, um a um, um a muitos, muitos para um e muitos para muitos ocasionando em uma nova classe associativa no ultimo tipo de relacionamento descrito.

A figura 9 representa um exemplo de um diagrama de classe:

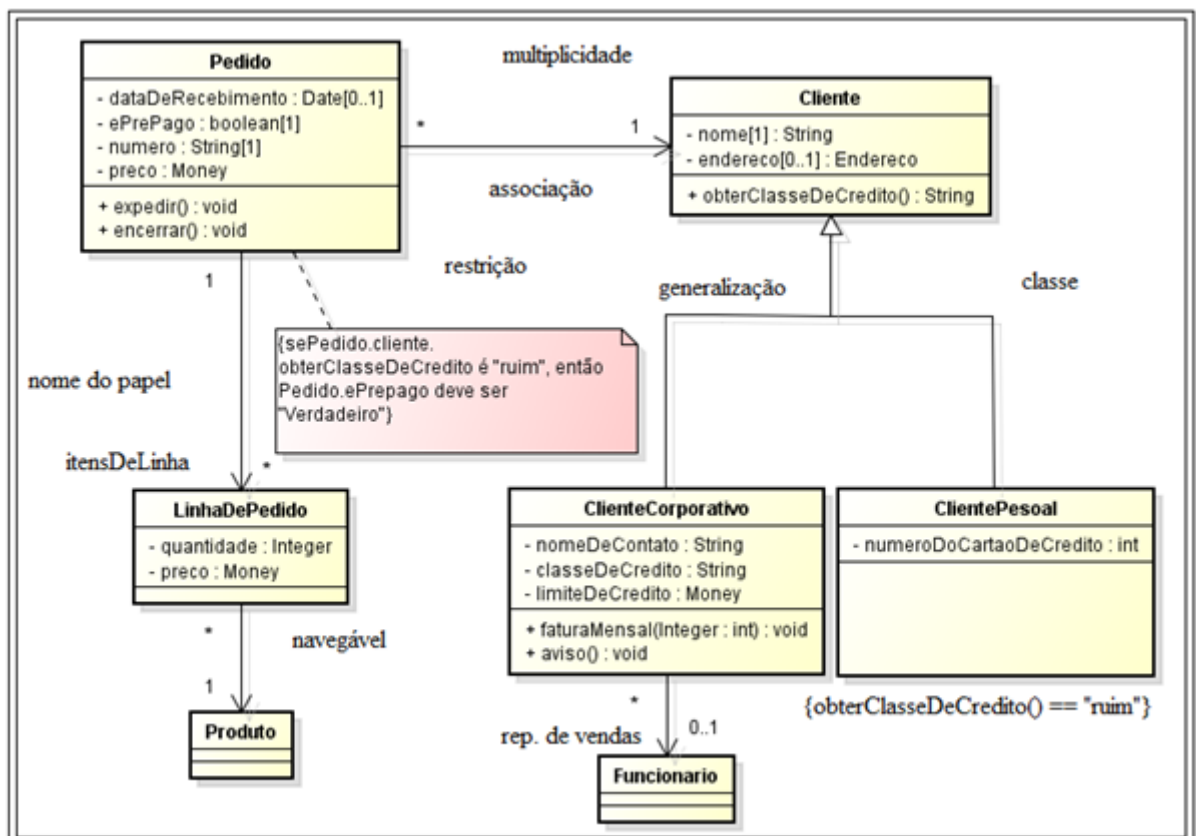


Figura 9 - Diagrama de Classe

Fonte: Fowler 2005, pg. 53.

1.1.3.3. Diagrama de Caso de Uso

O diagrama de caso de uso faz parte da família dos diagramas de comportamento, sendo este responsável por descrever a interação entre os agentes externos e o sistema. Entende-se por agentes externos usuários ou até mesmo outros sistemas capazes de interagir com o sistema que está sendo representado pelo diagrama.

Sua representação gráfica é bem simples e objetiva não contendo muitas notações, o mais importante de um caso de uso não é o seu diagrama e sim sua descrição.

Os casos de uso são técnicas para captar os requisitos funcionais de um sistema. Eles servem para descrever as interações típicas entre os usuários de um sistema e o próprio sistema, fornecendo uma narrativa sobre como o sistema é utilizado. (FOWLER, 2005, pg. 104)

Um sistema é composto por muitos módulos, sendo cada um desses módulos composto por um conjunto de funcionalidades, as funcionalidades descrevem os comportamentos desses módulos, os documentos de casos de uso entram nesse contexto com o objetivo de descrever de maneira técnica as funcionalidades que compõe um modulo.

Não existe nenhuma maneira padronizada para escrever o conteúdo de um caso de uso, diferentes formatos funcionam bem em diferentes casos.

Fowler (2005, pg. 106) segue dizendo que mesmo existindo mais de uma maneira de escrever um caso de uso, a sua descrição interna é dividida em algumas partes, sendo elas:

- **Breve descrição:** Geralmente a breve descrição descreve de maneira genérica os objetivos do modulo em geral.
- **Atores:** Os atores são os agentes externos responsáveis por acessar o modulo, geralmente são usuários que tem permissão de acesso a funcionalidade, porem podem ser outros sistemas.
- **Pré-condições:** As pré-condições descrevem as regras que devem ser atendidas para que o modulo seja acessado pelo ator, caso essas regras não sejam atendidas nenhuma das funcionalidades descritas serão executadas.
- **Fluxo de eventos:** O fluxo de eventos descreve a sequencia de passos de cada funcionalidade do modulo, sendo essas funcionalidades divididas em fluxo básico, fluxos alternativos, fluxos de exceção e regras de negócio.

- **Fluxo básico:** O fluxo básico descreve a sequência de passos da funcionalidade principal, que geralmente é a mais acessada quando o módulo é executado.
- **Fluxos alternativos:** Os fluxos alternativos descrevem as outras funcionalidades compostas no módulo, sendo que essas funcionalidades podem ser acessadas a partir da funcionalidade principal.
- **Fluxos de exceção:** Os fluxos de exceções descrevem as ações que devem ser executadas pelo sistema caso alguma ação imprevista seja executada pelo ator.
- **Regras de negócio:** Geralmente são representadas por uma tabela que contem os componentes que fazem parte da funcionalidade, esses componentes podem ser campos de inclusão de informações, campos de seleção de dados, entre outros, as regras de negócio contém também uma breve descrição de cada componente bem como seus possíveis comportamentos específicos.

Essa técnica é utilizada para evitar a redundância e ambiguidade da linguagem formal, facilitando também o reajuste nos documentos, pois caso uma nova funcionalidade seja adicionada ao módulo ou retirada, com o auxílio desse modelo o reajuste do documento se torna mais simples, o que não aconteceria se essa técnica não fosse utilizada.

Os casos de uso são reconhecidos como uma parte importante da UML. Entretanto, a surpresa é que, de muitas maneiras, a definição de casos de uso na UML é muito rala. Nada na UML descreve como você vai capturar o conteúdo um caso de uso. O que a UML descreve é um diagrama de casos de uso, que mostra como utilizar casos relacionados entre si. Mas praticamente todo o valor dos casos de uso reside no conteúdo e o diagrama é de valor bastante limitado. (FOWLER, 2005, pg. 105)

O diagrama de caso de uso não é obrigatório, pois a maior ênfase é dada na descrição de seu documento e não em sua representação gráfica, pois geralmente a sua representação gráfica mostra de maneira muito genérica como os atores interagem com o sistema.

Basicamente um diagrama de caso de uso é composto por um ou muitos atores, que são representados por um *stickman*, que representa uma figura

humana através de simples traços. Esses atores estão relacionados com seus casos de uso que são representados por uma elipse com o nome do caso de uso em seu interior, um ou muitos atores podem estar relacionados a um caso de uso, esse relacionamento é representado por um traço entre o ator e o caso de uso.

O caso de uso pode estar relacionado com outros casos de uso, esse relacionamento pode ocorrer de duas maneiras diferentes, a primeira através da notação *include* que é representada por uma seta que aponta para o caso de uso a ser incluído, esse relacionamento representa a obrigatoriedade, ou seja, esse caso de uso deve ser executado toda vez que o caso de uso que o relacionou for executado, já o outro relacionamento possível é o *extends*. O *extends* também é representado por uma seta, porém essa seta parte do caso de uso que está sendo relacionado para o caso de uso que o relacionou e representa a execução eventual, ou seja, o caso de uso relacionado pode eventualmente ser executado quando o caso de uso que o relacionou for executado.

A figura 10 representa um exemplo de um diagrama de casos de uso:

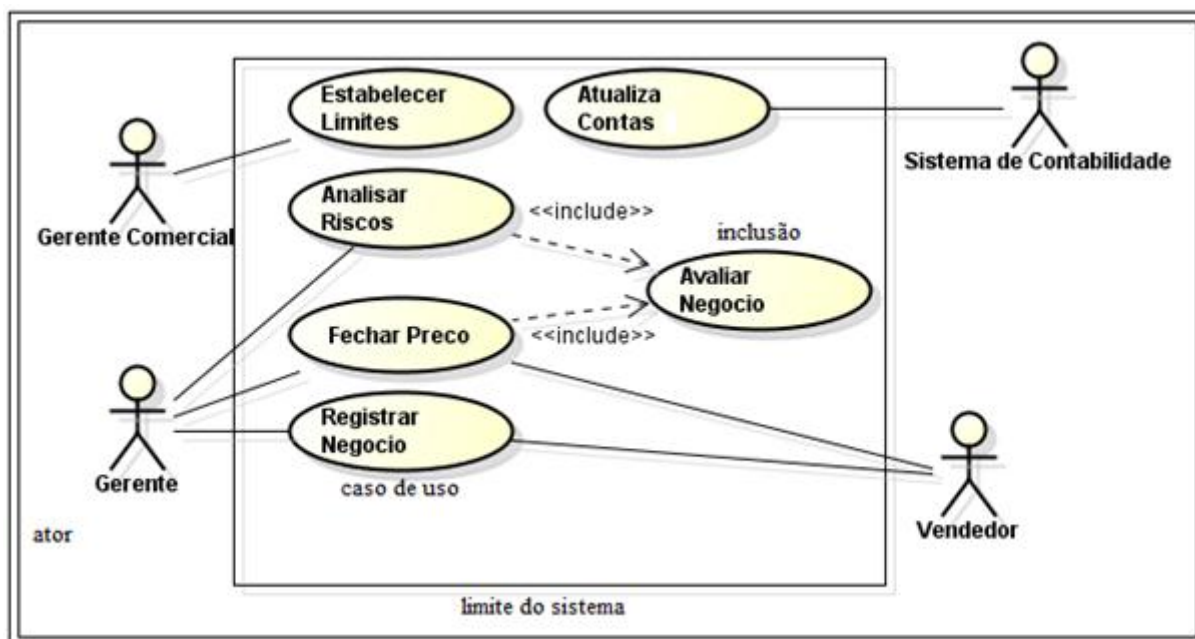


Figura 10 - Diagrama de Caso de Uso
Fonte: Fowler 2005, pg. 107

Com o auxílio da UML o desenvolvimento de sistemas se torna cada vez mais claro, objetivo e simples aumentando cada vez mais o nível de qualidade do *software*.

1.1.3.4. Diagrama de Sequência

O diagrama de sequência faz parte da família dos diagramas de interações, a sua principal função é descrever um cenário de maneira detalhada enfatizando a troca de mensagens entre os objetos que estão inseridos no contexto desse cenário.

Normalmente, um diagrama de sequência captura o comportamento de um único cenário. O diagrama mostra vários exemplos de objetos e mensagens que são passadas entre esses objetos dentro de um caso de uso. (FOWLER, 2005, pg. 67)

Fowler (2005, pg. 68) segue dizendo que os diagramas de sequência mostram as interações realizadas entre os objetos através da troca de mensagens. Em sua representação gráfica os objetos do cenário são representados por um retângulo com o nome do objeto em seu interior. Todos os objetos contidos em um diagrama de sequência são posicionados no início da página da esquerda para direita, e cada objeto possui a sua linha de vida que corre verticalmente na página, essas linhas são responsáveis por representar o ciclo de vida do objeto bem como os comportamentos executados por ele no contexto do cenário.

O cenário é iniciado através de uma seta com um círculo em seu início representando a entrada no cenário, essa seta aponta para o primeiro participante do cenário que dará início a sequência de interações a serem executadas por ele. Quando o cenário é iniciado a barra de ativação do primeiro participante é iniciada, a mesma só é encerrada quando existe o término de todas as interações de outros participantes que são utilizados pelo participante que iniciou a interação. Geralmente essa barra de ativação é a primeira a ser iniciada e a última a ser encerrada já que ela representa o início e o término do cenário, estes componentes podem ser vistos na Figura 11.

As interações entre os participantes são representadas por uma seta direcionada ao participante responsável por executar a interação, em cima da seta é colocado o nome do método a ser executado por esse participante, esse método pode ter um retorno, caso tenha uma seta pode ser adicionada representando o retorno, o participante chamado pode chamar outros participantes e assim sucessivamente até obter o resultado esperado. Em cada chamada a outro participante uma seta um pouco mais abaixo do que a seta que

iniciou a barra de ativação é adicionada fazendo com que a linha de vida cresça verticalmente.

É importante deixar claro que os métodos podem ou não receber parâmetros a serem passados como argumentos para a execução, bem como podem ou não ter retorno.

A figura 11 representa um exemplo de um diagrama de sequência:

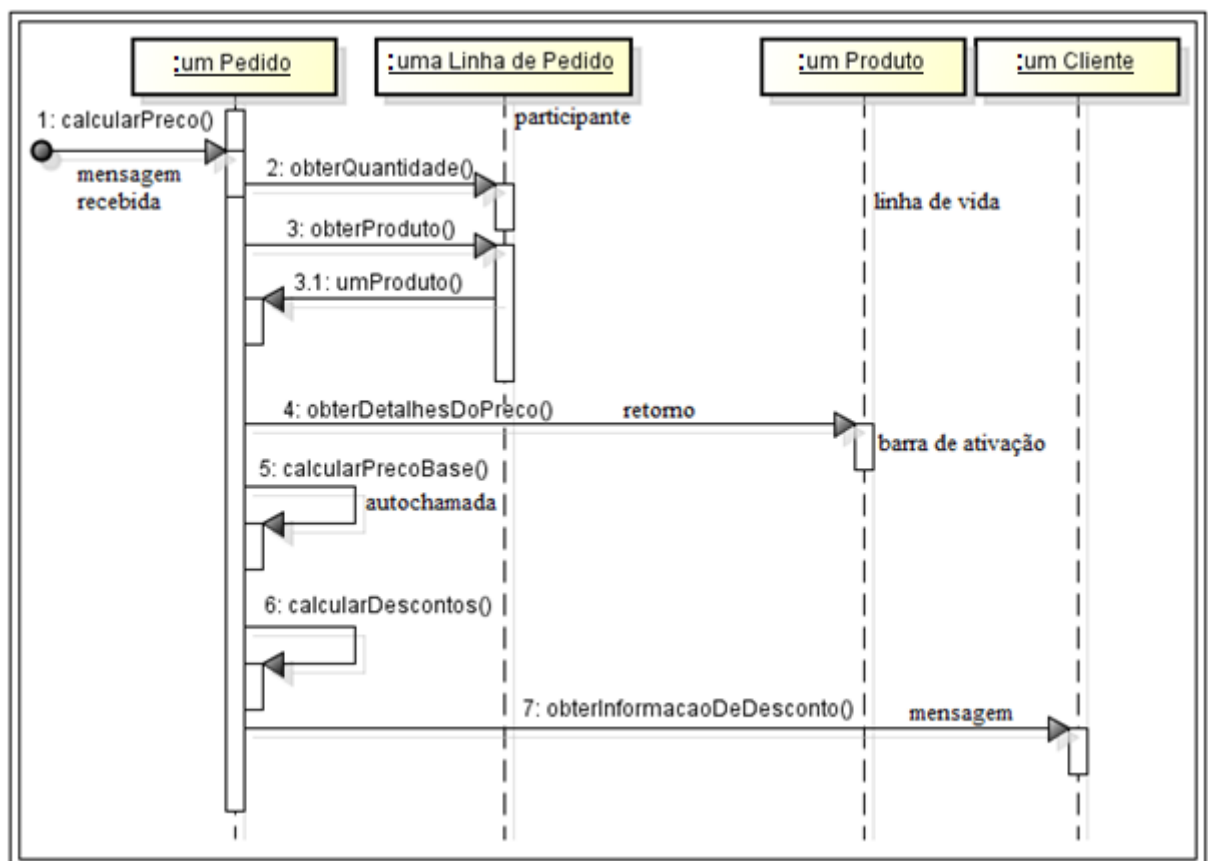


Figura 11 - Diagrama de Sequencia

Fonte: Fowler 2005, pg. 68.

A UML segue um processo de evolução contínuo se tornando uma ferramenta cada vez mais madura e eficiente, sendo assim os sistemas que a utilizam em seu processo de desenvolvimento possuem uma melhor qualidade.

1.1.4. Qualidade de *software*

Hoje em dia existem várias ferramentas que auxiliam o desenvolvimento de *software* afim de que os mesmos sejam construídos com qualidade, mas o

conceito de qualidade de *software* pode variar muito dependendo do ponto de vista de quem a analisa. Abaixo Pressman define qualidade de maneira genérica.

“Qualidade é um conceito complexo e multifacetado” que pode ser descrito segundo cinco pontos de vista diferentes. A visão transcendental sustenta que qualidade é algo que se reconhece imediatamente, mas não se consegue definir explicitamente. A visão do usuário vê a qualidade em termos das metas específicas de um usuário final. Se um produto atende a essas metas, ele apresenta qualidade. A visão do fabricante define qualidade em termos de especificação original do produto. Se o produto atende as especificações, ele apresenta qualidade. A visão do produto sugere que qualidade pode ser ligada a características inerentes (por exemplo, funções e recursos) de um produto. Finalmente, a visão baseada em valor mede a qualidade tomando como base o quanto um cliente estaria disposto a pagar por um produto. Na realidade, qualidade engloba todas essas visões e outras mais. (PRESSMAN, 2011, pg. 359).

Já quando se trata de desenvolvimento de *software*, que é um produto intangível, a qualidade engloba o grau de atendimento as funcionalidades e características descritas no modelo de requisitos, e a conformidade em que a construção segue o projeto resultando no sistema proposto.

Pressman (2011, pg. 359) diz que o *software* não é um produto físico, verificar seu nível de qualidade é algo bem mais complexo. Para que um *software* possua qualidade é necessário avaliar um conjunto de fatores, tais como, requisitos funcionais e não funcionais, desempenho, durabilidade, confiabilidade, manutenibilidade, qualidade de código desenvolvido, para que existam menos manutenções corretivas, entre muitos outros fatores.

Existem modelos de maturidade nacionais como o MPS-BR (Melhoria de Processos de *Software* Brasileiro) e internacionais como o CMMI (*Capability Maturity Model Integration*) ou Integração do Modelo de Maturidade de Recursos que medem a qualidade de *software* utilizando como padrão um conjunto de regras que devem ser seguidas durante o desenvolvimento do mesmo a fim de obter qualidade.

Porém em muitos casos verificar quão aderente o *software* é a esse conjunto de regras não é uma maneira eficaz de garantir qualidade, já que a qualidade do *software* varia muito dependendo do ponto de vista que é analisada.

Para um cliente a qualidade pode ser definida através dos requisitos funcionais do sistema, já para um arquiteto a qualidade pode ser definida pelas tecnologias e arquiteturas tanto de *hardware* quanto de *software* utilizadas para a construção do mesmo, para o analista ou engenheiro de *software* a qualidade

pode ser definida através da qualidade no desenvolvimento, criando componentes com baixo acoplamento e alta coesão.

Nesses casos, uma série de questões são levantadas, sendo algumas delas, pertinentes apenas à fase de construção:

- A equipe de desenvolvimento possui um nível técnico esperado de acordo com a complexidade do projeto?
- A equipe possui experiência?
- A equipe possui conhecimento na tecnologia empregada?
- A equipe tem um prazo bem estipulado para entrega de cada *release*?
- A arquitetura foi bem definida?
- A equipe possui testadores que conhecem o negócio do sistema?

Existem ainda muitos outros fatores a serem questionados, mas o que deve ficar claro é que independente do modelo de processo de *software* utilizado, das metodologias e dos paradigmas adotados sempre será uma tarefa difícil definir qualidade, já que existe um conjunto enorme de fatores a serem avaliados.

Pressman (2011, pg. 361) usa uma maneira mais genérica para definir a qualidade de *software* respondendo as seguintes perguntas:

- **Qualidade do desempenho** - O *software* fornece todo o conteúdo descrito no modelo de requisitos, de maneira que possa gerar valor ao usuário?
- **Qualidade dos recursos** - O *software* fornece recursos que surpreendem quem o use pela primeira vez?
- **Confiabilidade** - O *software* fornece todos os recursos sem falhas? Está disponível quando necessário?
- **Conformidade** - O *software* está de acordo com os padrões internos e externos relacionados com a aplicação? Segue as convenções de projeto e codificação de fato?
- **Durabilidade** - O *software* pode ser modificado, ou corrigido sem a geração involuntária de efeitos colaterais em outros módulos? As

alterações farão com que a taxa de erros aumentem e a confiabilidade diminua com o passar do tempo?

- **Facilidade de manutenção** - O *software* pode ser corrigido ou alterado em um período de tempo aceitável e curto? Os responsáveis pelo suporte têm todas as informações necessárias para realizar as alterações ou corrigir defeitos?
- **Estética** - O *software* tem uma *interface* intuitiva e elegante? Um usuário iniciante no sistema consegue ter a percepção de como é o funcionamento do mesmo?
- **Percepção** - O fornecedor do *software* tem procedência? Já produziu produtos que tiveram qualidade e aceitação no mercado?

Avaliar o *software* utilizando como parâmetro essas perguntas pode ser uma alternativa para medir seu nível de qualidade, pois desenvolver *software* sem falhas ou futuras melhorias é uma tarefa difícil.

A qualidade do *software* é o reflexo de um planejamento bem elaborado durante toda a definição do projeto, que é responsável por definir custos, cronogramas, recursos e todas as tarefas pertinentes ao planejamento do mesmo.

1.1.5. Projeto

O projeto de desenvolvimento de *software* não consiste somente no levantamento de requisitos, análise e projeto, desenvolvimento, testes e implantação, assim como qualquer projeto, um projeto de desenvolvimento de *software* requer muito planejamento para que o mesmo possa obter sucesso.

Para que esse planejamento seja bem realizado é necessário um bom gerenciamento do projeto.

O *Project Management Institute* ou Instituto de Gerenciamento de Projetos (PMI, 2009, pg. 3) diz que gerenciamento de projetos é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender a seus requisitos, e que o gerenciamento do projeto é realizado através da aplicação de processos agrupados abrangendo cinco grupos principais, sendo eles:

- Iniciação;
- Planejamento;
- Execução;
- Monitoramento e controle e
- Encerramento

O PMI (2009, pg. 35) segue dizendo que o grupo de processos de iniciação consiste nos processos realizados para definir um novo projeto ou uma nova fase de um projeto existente. Nos processos de iniciação o escopo inicial é definido de maneira genérica e os recursos financeiros iniciais são comprometidos. Para realizar tal tarefa é necessário que o gerente de projetos possua uma boa experiência, pois qualquer erro ocorrido nessa etapa pode influenciar negativamente o grupo de processos a serem executados na fase de planejamento.

Segundo o PMI (2009, pg. 37) o grupo de processos de planejamento consiste nos processos realizados para estabelecer o escopo total do esforço para a execução do projeto, como também definir e refinar os objetivos e desenvolver o plano de ação necessário para que esses objetivos sejam alcançados. Nos processos de planejamento são desenvolvidos os planos de gerenciamento e os documentos necessários para a execução do projeto, sendo esses documentos, escopo detalhado, cronograma, definição da equipe de desenvolvimento, entre outros, todo esse planejamento será colocado em prática no grupo de processos de execução.

O PMI (2009, pg. 45) descreve que o grupo de processos de execução consiste nos processos realizados para concluir o trabalho definido no plano de gerenciamento do projeto, de forma a cumprir as especificações impostas para a execução do mesmo. Esse grupo de processos é responsável por gerenciar e coordenar as pessoas envolvidas no projeto, bem como gerenciar os recursos disponíveis e executar as atividades do projeto em conformidade com o plano de gerenciamento do mesmo.

Durante a execução do projeto os resultados poderão requerer atualizações no planejamento, podendo ser essas alterações mudanças nos prazos previstos, na produtividade, na equipe de desenvolvimento, bem como nos

recursos alocados para a execução do projeto e riscos imprevistos na fase de planejamento. Geralmente nesse grupo de processos uma grande parte do orçamento previsto é consumido, sendo assim, qualquer ação imprevista nos grupos de iniciação e planejamento pode causar sérios impactos na execução do projeto, todas essas mudanças são encontradas durante a execução dos processos definidos no grupo de monitoramento e controle.

O PMI (2009, pg. 47) segue dizendo que o grupo de processos de monitoramento e controle consiste nos processos necessários para acompanhar, revisar e regular o progresso e o desempenho do projeto, bem como identificar as áreas nas quais serão necessárias mudanças no planejamento do projeto, e iniciar essas mudanças.

A grande vantagem da utilização desse grupo de processos é que o desempenho do projeto é observado e mensurado de forma periódica para identificar divergências em relação ao plano de gerenciamento do mesmo. O grupo de processos de monitoramento e controle também é responsável por controlar as mudanças e recomendar ações preventivas em antecipação a possíveis problemas, monitorar as atividades do projeto em relação ao plano de gerenciamento do mesmo, ou seja, conferir se a execução das atividades está de acordo com os prazos definidos no cronograma e influenciar os fatores que poderiam impedir o controle integrado de mudanças, para que somente as mudanças aprovadas sejam implementadas, após toda a verificação de monitoramento e controle do projeto o grupo de processos referente ao encerramento do mesmo deve ser executado.

O PMI (2009, pg. 53) diz que o grupo de processos de encerramento consiste nos processos executados para finalizar todas as atividades, de todos os grupos de processos de gerenciamento do projeto, visando completar formalmente o projeto, fase, ou obrigações contratuais. Durante a execução desses processos, podem ocorrer, aceitação do cliente, revisões do projeto, documentação de lições aprendidas, arquivamento dos documentos pertinentes ao projeto entre outros.

Os processos definidos nos cinco grupos principais de fases a serem executadas durante o gerenciamento do projeto garantem em sua grande maioria o sucesso de qualquer projeto de *software* a ser desenvolvido, porém para que

esse sucesso possa ser obtido é necessário o planejamento e execução de todas as etapas definidas durante o projeto.

Dentre os grupos de processos citados, o que será abordado é o grupo de processos referente ao monitoramento e controle, pois o *software* a ser desenvolvido será responsável por monitorar as ocorrências geradas durante toda a fase de desenvolvimento, a fim obter informações que auxiliam na tomada de decisões gerenciais.

2. PROJETO BUGSYS

Nesta etapa será apresentada a documentação do sistema BUGSYS, iniciando pelo escopo.

2.1. Escopo do sistema

O *Bugsys* será utilizado para controlar as ocorrências geradas no dia a dia de uma equipe de desenvolvimento de *software*. Essas ocorrências podem ser erros, defeitos, falhas, melhorias, sugestões, solicitações de mudança entre outras. É importante enfatizar a diferença entre erro, defeito e falha é que o erro é considerado uma ação humana, que resulta em *software* com defeitos, que se descobertos levam o sistema a falhar, o defeito por sua vez ocorre devido à omissão de informações, definições de dados, comandos ou instruções incorretas dentre outros fatores, já a falha ocorre quando um sistema não se comporta conforme o esperado ou apresenta resultados diferentes do planejado.

Os principais usuários do sistema são gerentes de projeto, analistas de requisitos, analista de sistemas, testadores de *software* e desenvolvedores, mas também podem ser clientes externos de um determinado projeto no qual o mesmo é proprietário. O sistema possui *workflows* responsáveis por gerenciar o fluxo de uma determinada ocorrência de acordo com seu tipo.

O principal objetivo do sistema é gerar informações relevantes de cada projeto gerenciado por ele, para que essas informações sirvam de apoio para auxiliar os gestores a tomar decisões para otimizar cada vez mais o processo de desenvolvimento de *software*, além de fornecer uma *interface* de comunicação interna e externa entre os membros de uma equipe de desenvolvimento e o cliente, e formalizar a geração de ocorrências.

2.2. Levantamento de requisitos

O levantamento de requisitos foi realizado de acordo com as melhores práticas definidas pela engenharia de *software*, as funcionalidades existentes no sistema foram definidas através de uma análise realizada em um sistema que possui funcionalidades em comum.

As principais funcionalidades do sistema são:

- **Manter usuários** – Gerenciamento de usuários do sistema, juntamente com a sua função.
- **Manter clientes** – Gerenciamento de clientes que possuem projetos em desenvolvimento ou manutenção, sendo gerenciados pelo sistema.
- **Manter Workflows** – Gerenciamento dos *workflows* responsáveis por gerenciar o ciclo de vida das fases do projeto.
- **Manter casos de uso** – Gerenciamento dos casos de uso de um respectivo projeto.
- **Manter projetos** – Gerenciamento de projetos gerenciados pelo sistema.
- **Manter ocorrências** – Cadastro e gerenciamento das ocorrências de cada projeto cadastrado no sistema, sendo esse gerenciamento responsável por abrir, fechar, aceitar, rejeitar, corrigir, validar, não validar, homologar ou suspender ocorrências.

2.3. Análise e projeto do sistema

A análise do sistema foi realizada utilizando a ferramenta *case Astah Community* (ASTAH, 2013), que fornece um conjunto de diagramas definidos pela UML. Com base nas informações levantadas na fase anterior, foram criados o diagrama de caso de uso, as especificações dos casos de uso e o diagrama de classes, bem como toda a modelagem do banco de dados do sistema, e estes serão demonstrados nos próximos itens.

2.4. Diagrama de Casos de Uso

A figura 12 exibe o diagrama de casos de uso que engloba todas as funcionalidades disponíveis no sistema de forma geral:

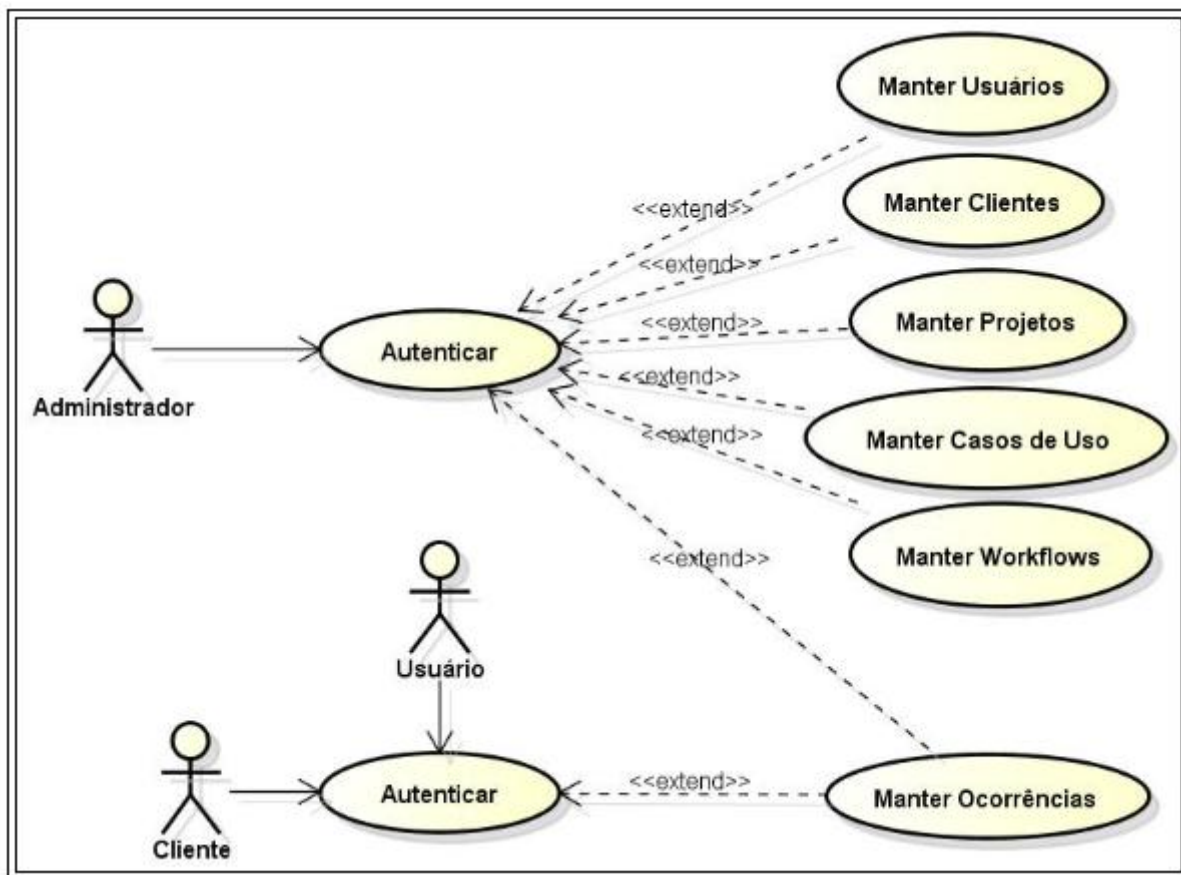


Figura 12 - Diagrama de Casos de Uso do Sistema

O sistema possui basicamente dois perfis de usuário, sendo eles:

Usuários comuns: são os membros da equipe de desenvolvimento de *software*, ou seja, desenvolvedores, testadores, analistas, gerentes de projeto, entre outros, são responsáveis por gerenciar as ocorrências de cada projeto.

Clientes: são usuários que tem acesso aos projetos vinculados a ele, podem gerenciar ocorrências externas referente aos seus projetos.

2.5. Especificação de Caso de Uso

Abaixo segue o caso de uso Manter Ocorrências. A letra “P” foi utilizada para simbolizar o passo a ser seguido, enquanto a letra “A” representa os fluxos alternativos e a letra “E” representa fluxos de exceção.

Manter Ocorrências

1 – Breve descrição

O objetivo deste caso de uso é permitir o cadastro, alteração e pesquisa de ocorrências no sistema.

2 – Atores

- Usuário
- Clientes

3 – Pré-Condições

O usuário deve estar autenticado no sistema e possuir perfil de Usuário ou Cliente.

4 – Fluxo de Eventos

4.1 – Fluxo Básico

- (P1) O caso de uso se inicia quando o ator navega até o menu “Ocorrências”;
- (P2) O sistema exibe a lista de ocorrências abertas em relação ao usuário autenticado [RN1];
- (P3) O caso de uso é encerrado.

4.2 – Fluxos Alternativos

(A1) Abrir Ocorrência

- (P1) O fluxo alternativo se inicia no passo (P2) do fluxo básico quando o ator deseja incluir um novo registro;
- (P2) O sistema exibe os campos para preenchimento [RN2];
- (P3) O ator preenche as informações solicitadas e aciona a opção “salvar” (E1);
- (P4) O sistema grava as informações e exibe a mensagem “Registro incluído com sucesso.”;
- (P5) O sistema retorna ao passo em que foi chamado.

(A2) Aceitar Ocorrência

(P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;

(P2) O sistema exibe os campos da ocorrência já preenchidos [RN3];

(P3) O ator altera a opção “Status” de “Aberto” para “Aceito” e adiciona um comentário sobre a ação;

(P4) O ator aciona a opção “salvar”;

(P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;

(P6) O sistema retorna ao passo em que foi chamado.

(A3) Corrigir Ocorrência

(P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;

(P2) O sistema exibe os campos da ocorrência já preenchidos [RN4];

(P3) O ator altera a opção “Status” de “Aceito” para “Corrigido”, altera o usuário responsável e adiciona um comentário sobre a ação;

(P4) O ator aciona a opção “salvar”;

(P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;

(P6) O sistema retorna ao passo em que foi chamado.

(A4) Rejeitar Ocorrência

(P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;

(P2) O sistema exibe os campos da ocorrência já preenchidos [RN4];

(P3) O ator altera a opção “Status” de “Aberto” para “Rejeitado”, altera o usuário responsável e adiciona um comentário sobre a ação;

(P4) O ator aciona a opção “salvar”;

(P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;

(P6) O sistema retorna ao passo em que foi chamado.

(A5) Validar Ocorrência

(P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;

(P2) O sistema exibe os campos da ocorrência já preenchidos [RN3];

(P3) O ator altera a opção “Status” de “Corrigido” para “Validado” e adiciona um comentário sobre a ação;

(P4) O ator aciona a opção “salvar”;

(P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;

(P6) O sistema retorna ao passo em que foi chamado.

(A6) Não Validar Ocorrência

(P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;

(P2) O sistema exibe os campos da ocorrência já preenchidos [RN4];

(P3) O ator altera a opção “Status” de “Corrigido” para “Não Validado”, altera o usuário responsável e adiciona um comentário sobre a ação;

(P4) O ator aciona a opção “salvar”;

(P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;

(P6) O sistema retorna ao passo em que foi chamado.

(A7) Homologar Ocorrência

- (P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;
- (P2) O sistema exibe os campos da ocorrência já preenchidos [RN3];
- (P3) O ator altera a opção “Status” de “Validado” para “Homologado” e adiciona um comentário sobre a ação;
- (P4) O ator aciona a opção “salvar”;
- (P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;
- (P6) O sistema retorna ao passo em que foi chamado.

(A8) Fechar Ocorrência

- (P1) O fluxo alternativo se inicia no (P2) do fluxo básico quando o ator aciona a opção editar deseja aceitar uma ocorrência atribuída a ele;
- (P2) O sistema exibe os campos da ocorrência já preenchidos [RN3];
- (P3) O ator altera a opção “Homologado” de “Fechado” para “Homologado” e adiciona um comentário sobre a ação;
- (P4) O ator aciona a opção “salvar”;
- (P5) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;
- (P6) O sistema retorna ao passo em que foi chamado.

(A9) Suspender Ocorrência

- (P1) O fluxo alternativo se inicia no (P3) do fluxo básico quando o ator deseja suspender uma ocorrência;
- (P2) O sistema exibe os campos da ocorrência já preenchidos [RN3];
- (P3) O ator altera o campo “Status” para “Suspendido”;
- (P5) O ator aciona a opção “salvar”;

- (P6) O sistema altera as informações e exibe a mensagem “Registro alterado com sucesso.”;
- (P7) O sistema retorna ao passo em que foi chamado.

4.3 – Fluxos de Exceção

(E1) Campo(s) obrigatório(s) não preenchido(s)

- (E1.1) O fluxo se inicia no passo (P3) do fluxo (A1) quando o ator não preenche algum dos campos obrigatórios;
- (E1.2) O sistema altera a cor da(s) borda(s) do(s) campo(s) não preenchido(s) para vermelho;
- (E1.3) O sistema retorna ao passo em que foi chamado.

5 – Regras de Negócio

[RN1] Lista de Ocorrências

Nome do Campo	Tipo (Tamanho)	Descrição	Regras do Campo	Obrigatório	Somente Leitura
Tipo de Ocorrência	Alfanumérico (30)				Sim
Status da Ocorrência	Alfanumérico (30)				Sim
Projeto	Alfanumérico (30)				Sim
Usuário Responsável	Alfanumérico (30)				Sim
Fase do Projeto	Combobox (30)				Sim

Tabela 1 - Lista de Ocorrências

[RN2] Preenchimento dos campos Ocorrência

Nome do Campo	Tipo (Tamanho)	Descrição	Regras do Campo	Obrigatório	Somente Leitura
Projeto	Combobox (30)			Sim	Não
Tipo de Ocorrência	Combobox (30)			Sim	Não
Usuário Responsável	Combobox (30)			Sim	Não
Workflow	Alfanumérico			Sim	Sim
Fase do Projeto	Combobox (30)			Sim	Não
Caso de Uso	Combobox (30)			Sim	Não
Status	Combobox (30)			Sim	Não
Data de Criação	Data		Campo preenchido automaticamente pelo sistema	Sim	Sim
Descrição da Ocorrência	Alfanumérico (250)			Sim	Não

Tabela 2 - Preenchimento dos campos Ocorrência**[RN3] Edição do Status da Ocorrência**

Nome do Campo	Tipo (Tamanho)	Descrição	Regras do Campo	Obrigatório	Somente Leitura
Projeto	Combobox (30)				Sim
Tipo de Ocorrência	Combobox (30)				Sim
Usuário Responsável	Combobox (30)				Sim
Workflow	Alfanumérico				Sim
Fase do Projeto	Combobox (30)				Sim
Caso de Uso	Combobox (30)				Sim
Status	Combobox (30)			Sim	Não
Data de Criação	Data				Sim
Descrição da Ocorrência	Alfanumérico			Sim	Não

Tabela 3 – Edição do Status da Ocorrência

[RN4] Edição do Status e de Usuário da Ocorrência

Nome do Campo	Tipo (Tamanho)	Descrição	Regras do Campo	Obrigatório	Somente Leitura
Projeto	Combobox (30)				Sim
Tipo de Ocorrência	Combobox (30)				Sim
Usuário Responsável	Combobox (30)			Sim	Não
Workflow	Alfanumérico				Sim
Fase do Projeto	Combobox (30)				Sim
Caso de Uso	Combobox (30)				Sim
Status	Combobox (30)			Sim	Não
Data de Criação	Data				Sim
Descrição da Ocorrência	Alfanumérico			Sim	Não

Tabela 4 – Edição do Status e de Usuário da Ocorrência

2.6. Diagrama de Classes

A figura 13 exibe o diagrama de classes do sistema:

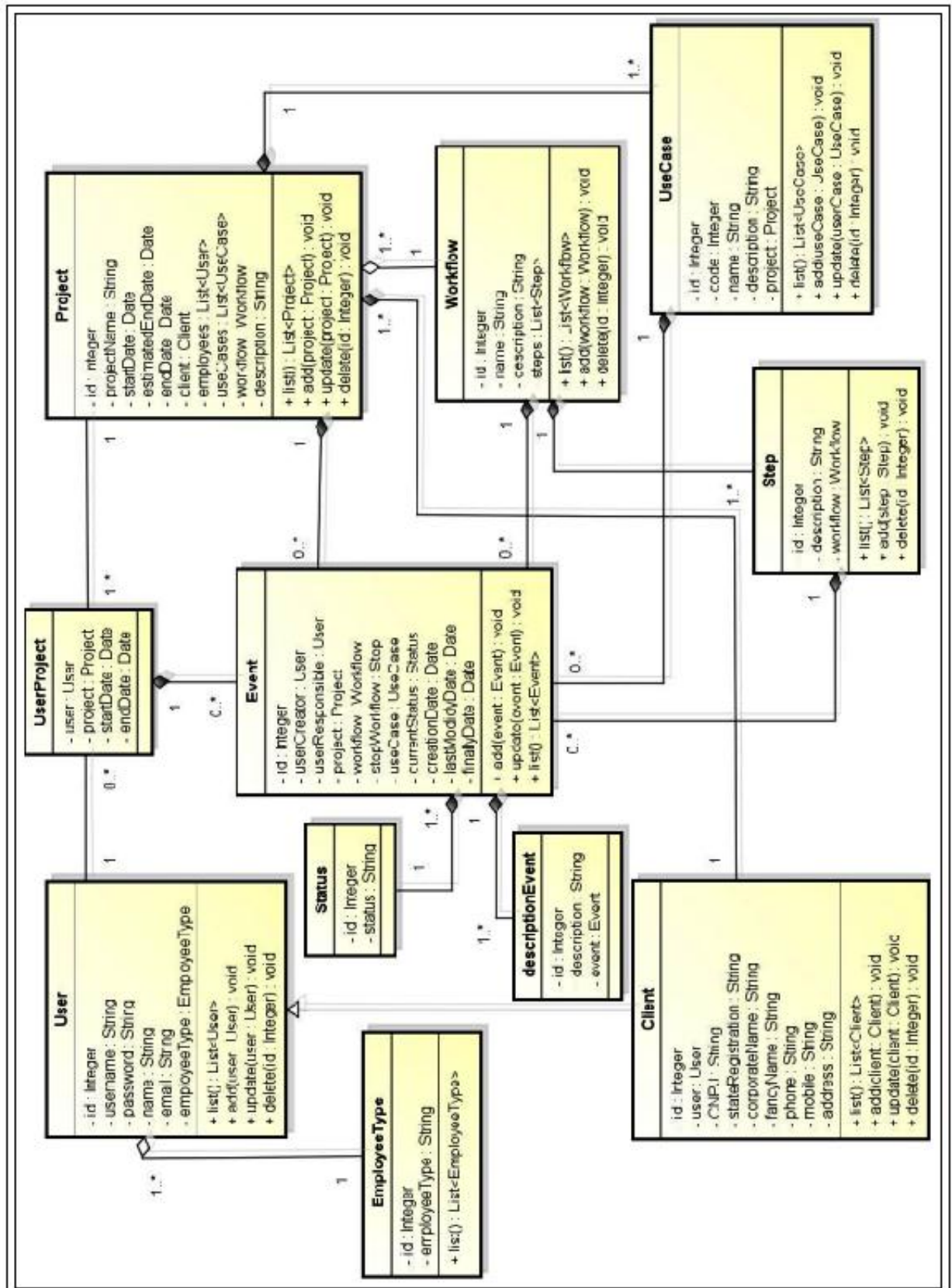


Figura 13 - Diagrama de Classes

O diagrama de classes é responsável por exibir como as entidades do sistema se relacionam entre si e qual é o grau de relacionamento entre elas.

No sistema todos os usuários internos (*User*) possuem um cargo (*EmployeeType*) e podem estar relacionados a vários projetos (*Project*), cada projeto pode estar relacionado com várias ocorrências (*Event*) envolvendo um ou muitos usuários do projeto (*UserProject*), já o usuário externo (*Client*) deve estar relacionado no mínimo a um projeto, porém podem existir vários projetos relacionados a um cliente e o cliente pode estar relacionado com uma ou muitas ocorrências. Um projeto sempre possuirá um fluxo de trabalho (*Workflow*) a ser seguido, esse por sua vez está relacionado com uma lista de fases (*Step*) que o compõe, o projeto ainda está relacionado com um ou muitos casos de uso (*UseCase*) que compõe o projeto, todas as ocorrências possuirão um status e estarão relacionadas a uma fase do *workflow* de um determinado projeto, podendo ser estas ocorrências relacionadas a usuários internos ou externos.

2.7. Diagrama de Sequência

A figura 14 exibe o diagrama de sequência que descreve os passos necessários para cadastrar uma ocorrência:

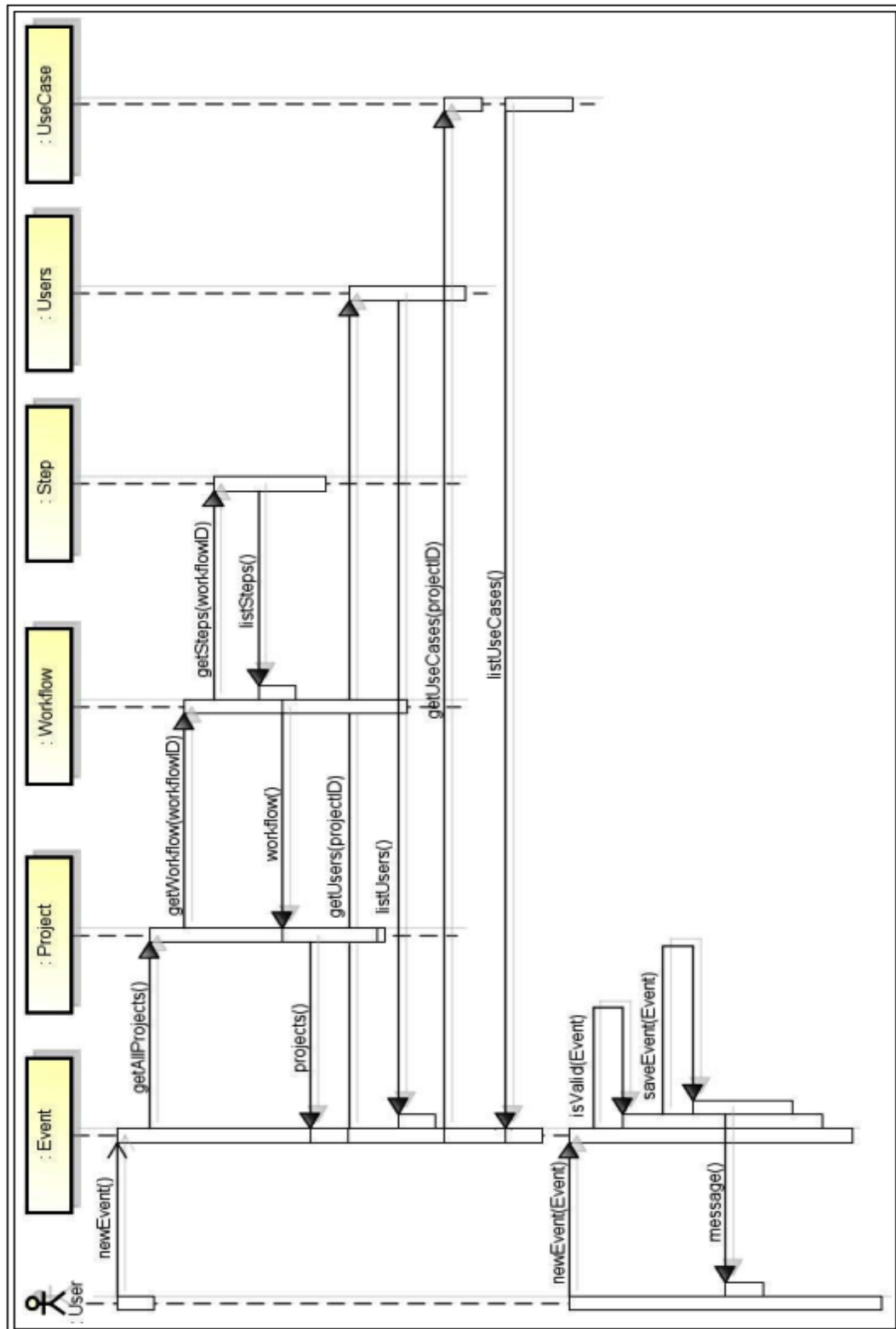


Figura 14 - Diagrama de Sequência - Cadastro de Ocorrências

O diagrama de sequência descreve a interação entre o ator e os objetos do sistema a fim de descrever uma funcionalidade complexa através de uma sequência de passos.

O ator (*User*) acessa o fluxo para inserir uma nova ocorrência (*Event*), como esse cadastro é parametrizado, ou seja, envolve vários objetos já cadastrados, é necessário realizar uma busca por todos os projetos (*Project*) em que o ator esteja vinculado, porém um projeto possui um fluxo de trabalho (*Workflow*) que por sua vez possui uma lista de fases (*Step*) que o compõe, então respectivamente o objeto (*Project*) recupera o (*Workflow*) relacionado a ele, e o objeto (*Workflow*) recupera a lista de (*Step*) que o compõe, após o retorno dos objetos, é necessário recuperar todos os usuários (*Users*) envolvidos com o projeto escolhido, como também todos os casos de uso (*UseCases*) relacionados ao projeto.

Após todos os objetos necessários para o cadastro serem recuperados, o ator preenche os dados da ocorrência e realiza o cadastro, o objeto (*Event*) valida os dados através do método *isValid(Event)*, caso o método retorne sucesso na sua validação então o método *saveEvent(Event)* é executado persistindo os dados no banco de dados e uma mensagem de sucesso é retornada ao ator, caso contrario o método *isValid(Event)* retornará uma mensagem informado a origem do erro para que o ator o corrija e execute o fluxo novamente.

2.8. Modelo Entidade-Relacionamento

A figura 15 exibe o diagrama de entidade relacionamento do banco de dados:

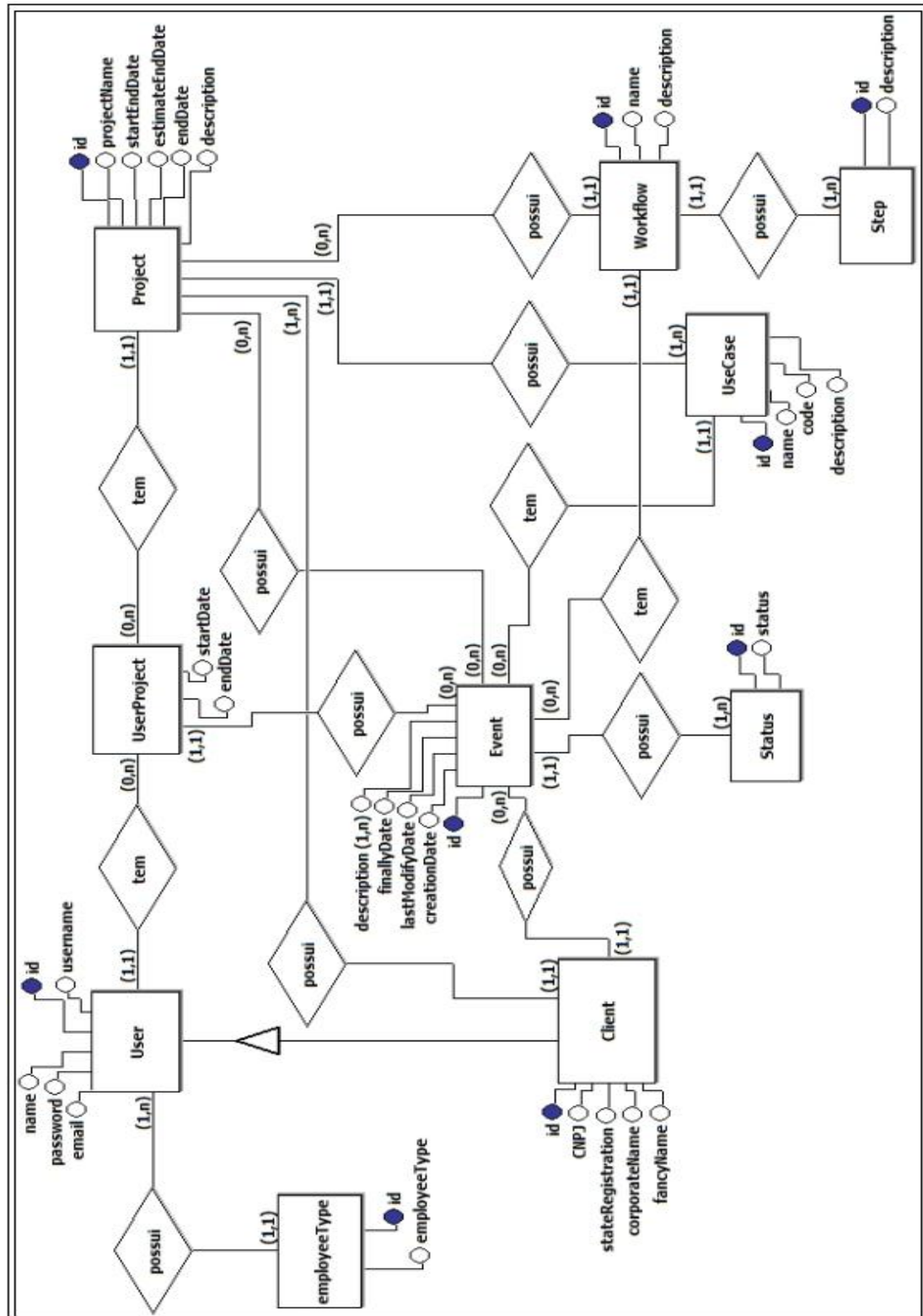


Figura 15 - Modelo Entidade-Relacionamento do Sistema

Esse diagrama é responsável por exibir a estrutura das entidades, bem como os seus relacionamentos através da cardinalidade que é a relação entre duas ou mais entidades indicando o tipo de relacionamento que pode variar de um para um, um para muitos, muitos para um e muitos para muitos.

Esse tipo de diagrama precede o diagrama Relacional e ambos modelam a maneira de como os dados serão gerenciados pelo banco de dados.

2.9. Modelo Relacional

A figura 16 exibe o diagrama de relacionamento do banco de dados:

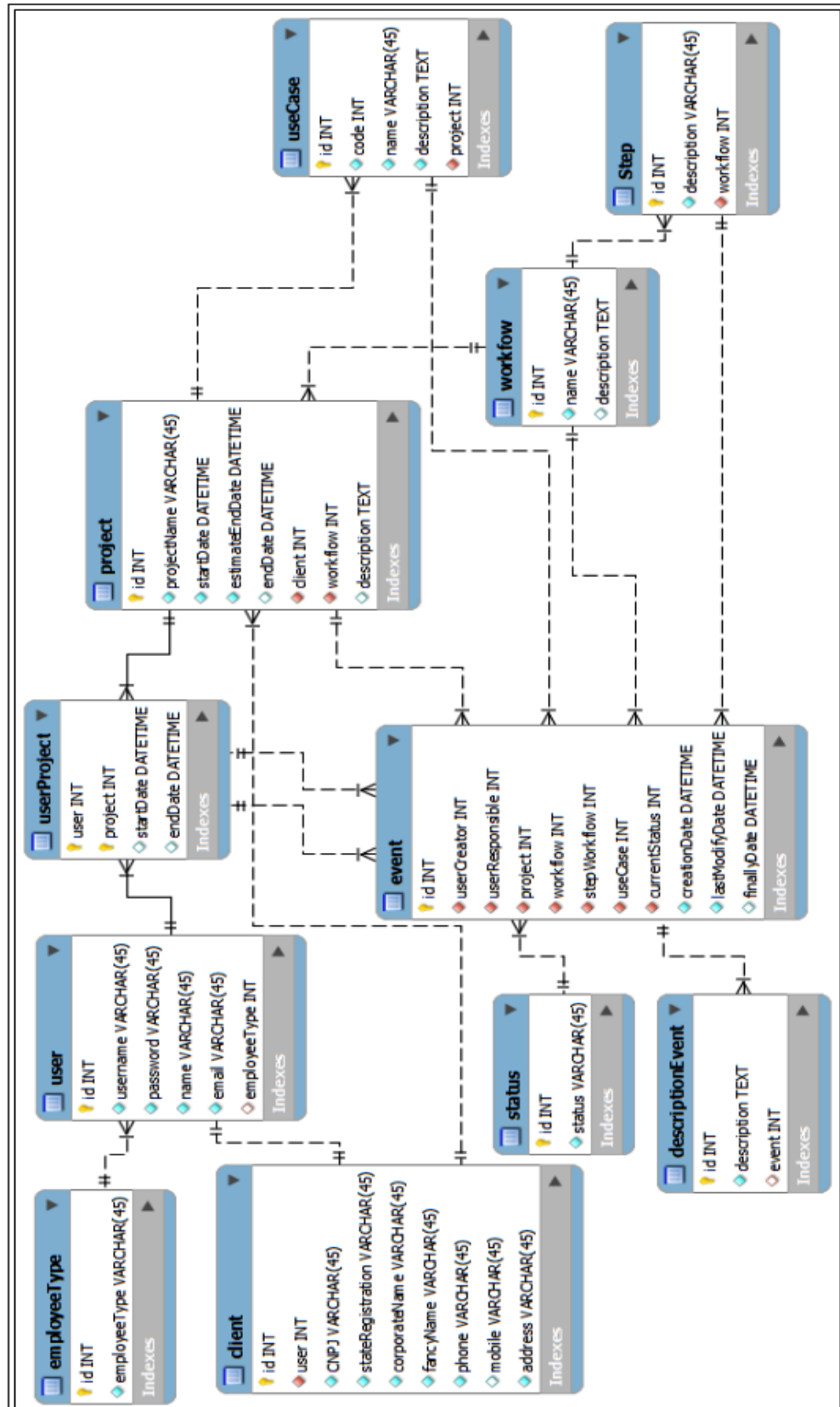


Figura 16 - Modelo Relacional do Sistema

O diagrama relacional já exibe de forma clara o relacionamento entre as tabelas do banco de dados do sistema. Os relacionamentos são representados por um traço que em uma de suas extremidades tem três traços. Esses traços representam o relacionamento “muitos” enquanto a outra extremidade tem dois traços, esses traços representam o relacionamento “um”.

2.10. Desenvolvimento

O sistema foi desenvolvido utilizando a plataforma Java, pois a mesma oferece uma gama de recursos e tecnologias necessárias para o desenvolvimento desse projeto, dentre as tecnologias adotadas para a implementação estão o JEE (*Java Enterprise Edition*) que fornece uma rica especificação para o desenvolvimento de aplicações corporativas em ambiente *web*, utilizando como tecnologia o JSP (*Java Server Pages*).

O sistema também inclui em sua arquitetura *frameworks* que auxiliam no desenvolvimento ágil de sistemas *web*. Para gerenciar todo o controle referente à camada de negócio, será utilizado um *framework* chamado VRaptor. Esse *framework* abstrai toda a complexidade do protocolo HTTP (*Hyper Text Transfer Protocol*) e é responsável por prover dados para a camada de visualização do projeto, bem como também realizar toda a lógica de negócio e fornecer os dados da visão para a camada de persistência de dados.

A camada de persistência de dados utiliza um *framework* ORM (*Object Relational Mapping*) chamado *Hibernate* para realizar o gerenciamento dos dados do sistema, como o próprio nome diz um *framework* ORM é responsável por mapear para o modelo de objetos um banco de dados que utiliza o modelo relacional para interligar as tabelas que compõem um sistema. A sua grande vantagem é o desacoplamento com qualquer SGBD (Sistema Gerenciador de Banco de Dados), pois ele fornece uma linguagem própria para a escrita de *queries* chamada HQL (*Hibernate Query Language*). Com ela é possível escrever um único código SQL (*Structured Query Language*) e o utilizar em vários SGBDs diferentes.

Para a camada de visualização, que é responsável pela *interface* entre usuário e máquina, foi utilizado um *framework* chamado *Bootstrap*, que é responsável por gerenciar os estilos das páginas CSS (*Cascading Style Sheets*). Esse *framework* também fornece *templates* HTML (*Hyper Text Markup Language*)

para o *design* de aplicações *web*, bem como gerência as diversas incompatibilidades entre versões de diferentes *browsers*. Outra grande vantagem que esse *framework* proporciona, é a formatação das páginas HTML para qualquer dispositivo, ou seja, o sistema pode ser acessado de qualquer dispositivo móvel, como *smartphones*, que as páginas HTML serão redesenhadas para o dispositivo específico sem perder o estilo da página. Isso se torna possível com a utilização de uma técnica conhecida como *layout responsive* que já é implementada por esse *framework*.

Para a criação de componentes HTML e execução de validações pertinentes ao cliente, ou seja, o *browser*, foi utilizado um *framework javascript* chamado *jQuery*, esse *framework* fornece uma linguagem intuitiva e simples, mas a sua utilização se torna necessária para tratar a incompatibilidade entre as diversas implementações distintas do *javascript* entre os *browsers*.

Para armazenar os dados do sistema foi utilizado como SGBD o MySQL, pois é um dos SGBDs mais utilizados atualmente em sistemas *web*, devido a sua *performance* nos servidores de aplicação, pois se trata de um SGBD compacto e robusto que proporciona facilidade no seu uso e boa escalabilidade.

Para gerenciar a aplicação na *web*, foi utilizado o *Apache Tomcat* como servidor de aplicação, um servidor de aplicação é responsável por prover a aplicação na *web*, gerenciando todo o seu processamento interno dentro da JVM (*Java Virtual Machine*).

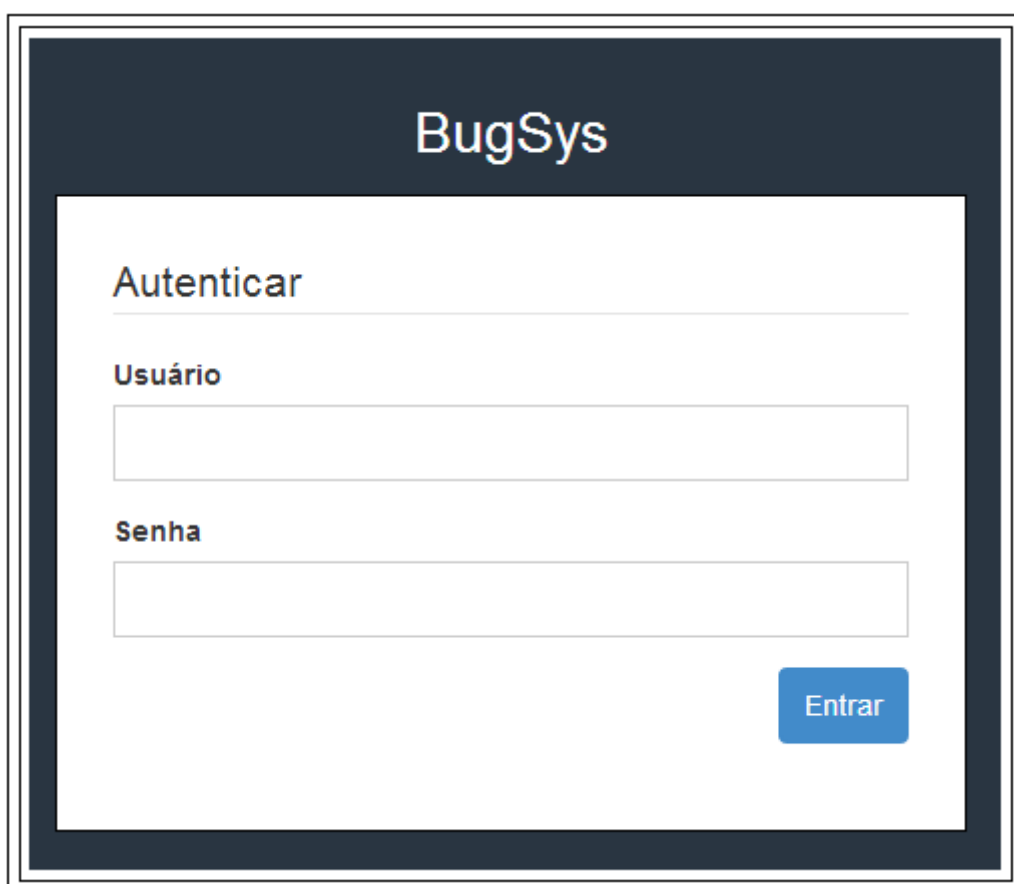
2.11. Apresentação do Sistema

Neste item, serão apresentadas as telas do sistema. Estas telas foram desenvolvidas a partir das especificações dos casos de uso apresentadas neste projeto. Em cada subitem, estará uma imagem com as telas referentes ao caso de uso e uma breve descrição sobre como estas funcionam.

Tela de Autenticação

Este fluxo é responsável pela autenticação do usuário no sistema, os campos “Usuário” e “Senha” são obrigatórios, caso o usuário não os preencha, ou os preencha com valores inválidos ocorrerá uma validação e o mesmo não terá acesso ao sistema.

A figura 17 exibe a tela de autenticação do sistema:



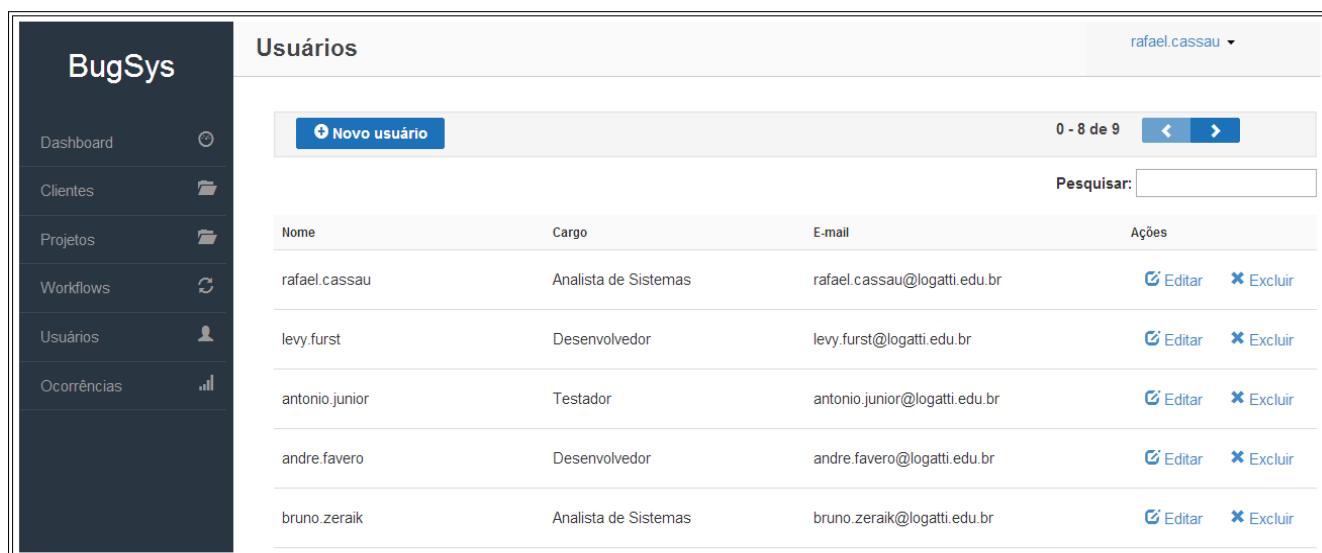
A imagem mostra a interface de autenticação do sistema BugSys. O título "BugSys" está no topo em uma barra escura. Abaixo, o título "Autenticar" precede dois campos de entrada: "Usuário" e "Senha". Um botão azul "Entrar" está na parte inferior direita do formulário.

Figura 17 - Tela de Autenticação do Sistema

Tela de Listagem de Usuários

O caso de uso de usuário é responsável pelo cadastrado e gerenciamento de usuários do sistema.

A figura 18 exibe a lista de usuários cadastrados no sistema com as opções de paginação, filtro, edição e exclusão de usuários:

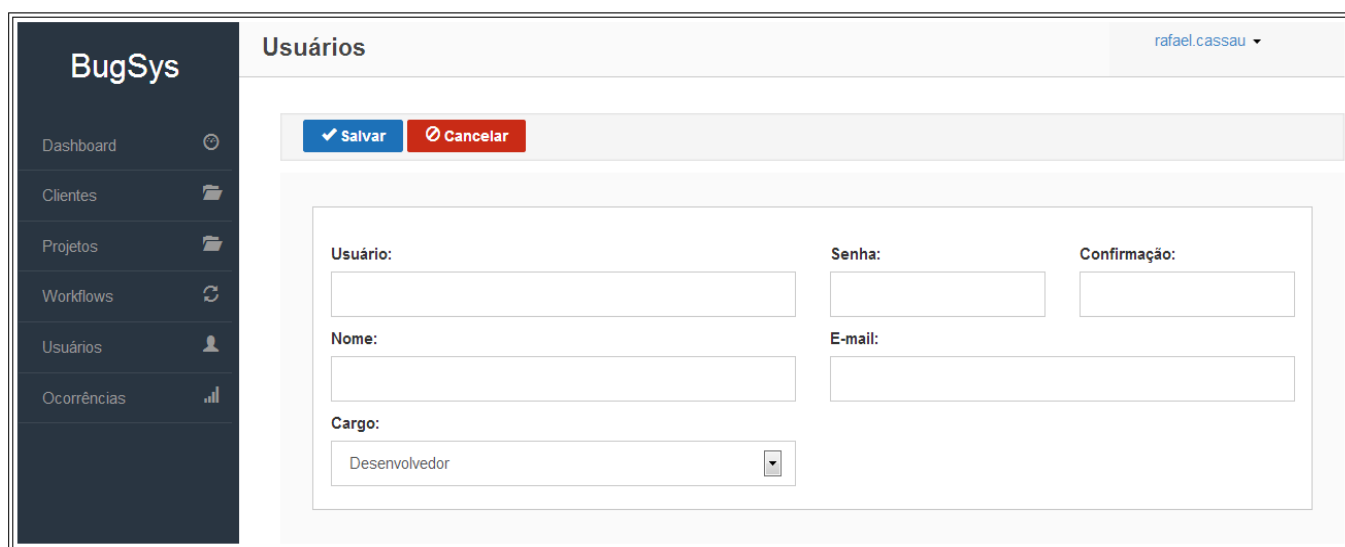


BugSys			
Usuários			rafael.cassau ▾
+ Novo usuário		0 - 8 de 9 < >	
Pesquisar: <input type="text"/>			
Nome	Cargo	E-mail	Ações
rafael.cassau	Analista de Sistemas	rafael.cassau@logatti.edu.br	✎ Editar ✖ Excluir
levy.furst	Desenvolvedor	levy.furst@logatti.edu.br	✎ Editar ✖ Excluir
antonio.junior	Testador	antonio.junior@logatti.edu.br	✎ Editar ✖ Excluir
andre.favero	Desenvolvedor	andre.favero@logatti.edu.br	✎ Editar ✖ Excluir
bruno.zeraik	Analista de Sistemas	bruno.zeraik@logatti.edu.br	✎ Editar ✖ Excluir

Figura 18 - Tela de Listagem de Usuários do Sistema

Tela de Cadastro e Alteração de Usuários

A figura 19 exibe o fluxo de cadastro e alteração de usuários no sistema:



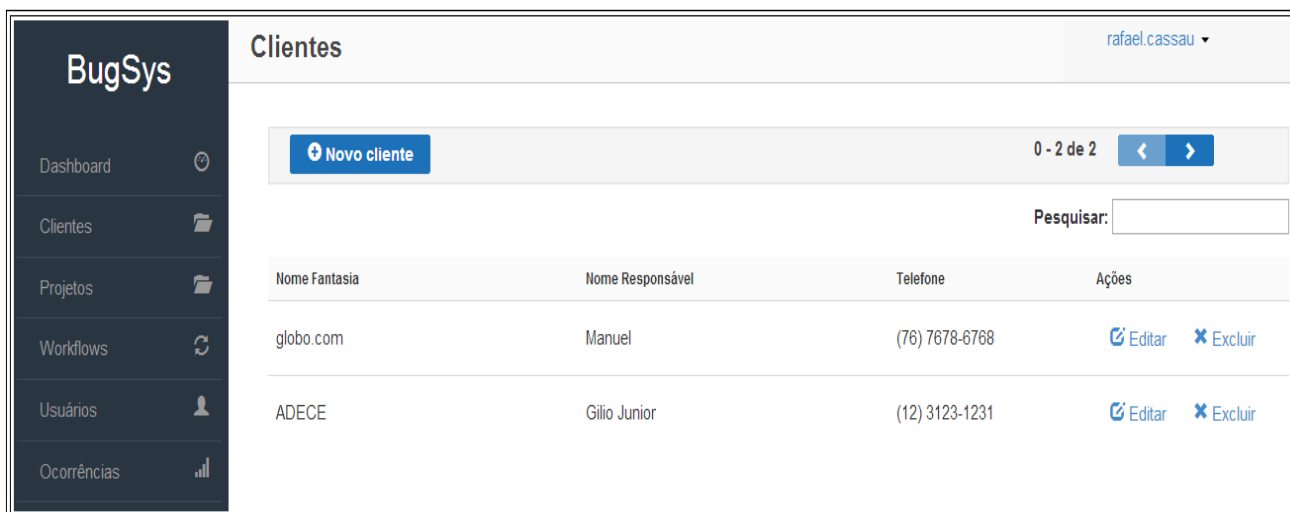
BugSys		
Usuários		rafael.cassau ▾
✓ Salvar ✗ Cancelar		
Usuário: <input type="text"/>	Senha: <input type="text"/>	Confirmação: <input type="text"/>
Nome: <input type="text"/>	E-mail: <input type="text"/>	
Cargo: <div>Desenvolvedor ▾</div>		

Figura 19 - Tela de Cadastro de Usuários no Sistema

Tela de Listagem de Clientes

O caso de uso de cliente é responsável pelo cadastrado e gerenciamento de clientes do sistema.

A figura 20 exibe a lista de clientes cadastrados no sistema com as opções de paginação, filtro, edição e exclusão de clientes:



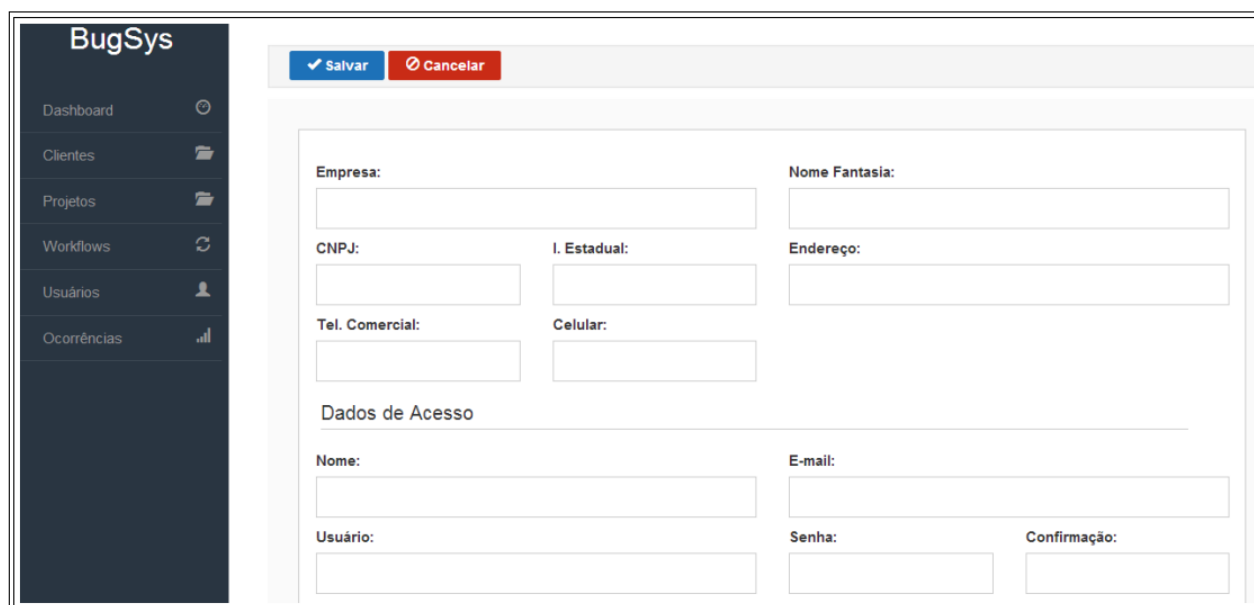
The screenshot shows the 'Clientes' page in the BugSys application. On the left is a dark sidebar with navigation links: Dashboard, Clientes, Projetos, Workflows, Usuários, and Ocorrências. The main content area has a header with the title 'Clientes' and a user profile 'rafael.cassau'. Below the header is a toolbar with a '+ Novo cliente' button and pagination controls showing '0 - 2 de 2'. A search bar labeled 'Pesquisar:' is also present. The main area contains a table with two rows of client data.

Nome Fantasia	Nome Responsável	Telefone	Ações
globo.com	Manuel	(76) 7678-6768	Editar Excluir
ADECE	Gilio Junior	(12) 3123-1231	Editar Excluir

Figura 20 - Listagem de Clientes do Sistema

Tela de Cadastro e Alteração de Clientes

A figura 21 exibe o fluxo de cadastro e alteração de clientes no sistema:



The screenshot shows the 'Cadastro e Alteração de Clientes' form in the BugSys application. The sidebar is the same as in Figure 20. The main content area has a header with 'Salvar' and 'Cancelar' buttons. The form is divided into several sections for data entry.

Form Fields:

- Empresa:** Text input
- Nome Fantasia:** Text input
- CNPJ:** Text input
- I. Estadual:** Text input
- Endereço:** Text input
- Tel. Comercial:** Text input
- Celular:** Text input
- Dados de Acesso:** Section header
- Nome:** Text input
- E-mail:** Text input
- Usuário:** Text input
- Senha:** Text input
- Confirmação:** Text input

Figura 21 - Tela de Cadastro de Clientes do Sistema

Todo cliente do sistema é um usuário externo, sendo assim quando um cliente é cadastrado o mesmo já informa os dados de usuário, para que posteriormente ele possa acessar o sistema e gerenciar as ocorrências relacionadas ao seu projeto.

Tela de Cadastro e Visualização de *Workflows*

A figura 22 exibe o fluxo de cadastro e visualizações de *workflows* no sistema.

A interface do sistema BugSys apresenta uma barra lateral escura com o menu de navegação contendo: Dashboard, Clientes, Projetos, Workflows (destacado), Usuários e Ocorrências. No topo da área principal, há botões para 'Salvar' (verde) e 'Cancelar' (vermelho). O formulário de cadastro contém os seguintes campos:

- Título:** Campo de texto com o valor 'Modelo Cascata'.
- Descrição:** Campo de texto com o valor 'Modelo composto por um conjunto de fases que devem ser seguidas a risca uma após a outra.'
- Fases:** Seção com uma barra de adição (+) e uma lista de fases existentes:

Fases	Ação
Levantamento de Requisitos	X
Projeto	X
Análise	X
Desenvolvimento	X

Figura 22 - Cadastro de *Workflows* do Sistema

Um *workflow* é composto por um conjunto de fases que definem o processo de desenvolvimento de *software* usado em vários projetos gerenciados pelo sistema. Depois de cadastrado um *workflow* não pode ser alterado, pois a sua alteração tem impacto direto com os processos definidos para cada projeto.

Tela de Cadastro e Alteração de Projetos e seus Casos de Uso

A figura 23 exibe o fluxo de cadastro e alteração de projetos e seus respectivos casos de uso, juntamente com todos os membros da equipe vinculados a esse projeto.

The screenshot displays the 'BugSys' application interface for project management. On the left is a dark sidebar with the 'BugSys' logo and navigation links: Dashboard, Clientes, Projetos, Workflows, Usuários, and Ocorrências. The main content area has a top bar with 'Salvar' and 'Cancelar' buttons. Below this, the 'Nome do Projeto:' field contains 'SIG - Sistema Integrado de Gestão do Governo do Ceará'. The 'Data de Inicio:' is '04/01/2011' and the 'Data de Termino Estimada:' is '08/05/2014'. The 'Cliente:' dropdown is set to 'ADECE' and the 'Workflow:' dropdown is set to 'Cascata'. The 'Descrição:' field contains 'ERP web que gerencia todas as câmaras setoriais da ADECE.'.

Below the description is a section titled 'Casos de Uso' with a '+' icon. It contains a table with the following data:

Código	Nome	Descrição	Ações
001	Autenticação	Realiza a autenticação dos usuários existentes na ADECE	Editar Excluir

Below the table is a section titled 'Membros da Equipe'. It shows a list of team members, with 'Rafael Stain Cassau - Analista de Sistemas' currently visible.

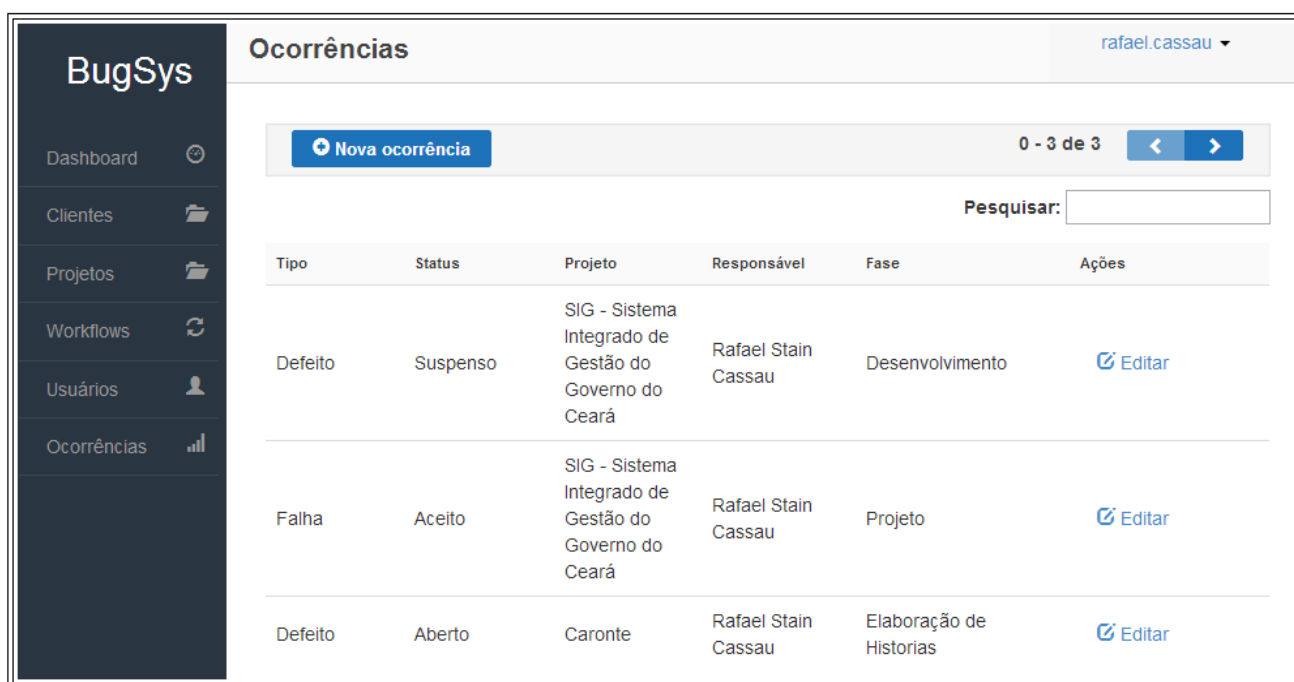
Figura 23 - Cadastro e Alteração de Projetos do Sistema

Um projeto é composto por no mínimo 1 (um) ou muitos casos de uso vinculados a ele, cada caso de uso possui um código que o identifica de forma única para o projeto. O projeto é composto também de 1 (um) ou muitos membros da equipe vinculados a ele, esses membros podem ser Analistas de Requisitos, Analistas de Sistemas, Desenvolvedores, Testadores, Gerentes de Projeto entre outros, porém é obrigatório que exista no mínimo um Gerente de Projeto vinculado a cada projeto, pois o mesmo é responsável pela interação entre as ocorrências

externas abertas pelo cliente e a equipe vinculado ao projeto, que será responsável pela resolução dessas ocorrências.

Tela de Listagem de Ocorrências

A figura 24 exibe o fluxo de listagem de ocorrências de todos os projetos vinculados ao usuário autenticado no sistema.



BugSys						Ocorrências	rafael.cassau ▾
Dashboard		+ Nova ocorrência		0 - 3 de 3	< >	Pesquisar: <input type="text"/>	
Clientes		Tipo	Status	Projeto	Responsável	Fase	Ações
Projetos		Defeito	Suspenso	SIG - Sistema Integrado de Gestão do Governo do Ceará	Rafael Stain Cassau	Desenvolvimento	✎ Editar
Workflows		Falha	Aceito	SIG - Sistema Integrado de Gestão do Governo do Ceará	Rafael Stain Cassau	Projeto	✎ Editar
Usuários		Defeito	Aberto	Caronte	Rafael Stain Cassau	Elaboração de Historias	✎ Editar
Ocorrências							

Figura 24 - Listagem de Ocorrências do Sistema

O fluxo de listagem de ocorrências do sistema exibe todas as ocorrências em que o usuário autenticado seja responsável ou criador da ocorrência.

Tela de Cadastro e Alteração de Ocorrências

A figura 25 exibe o fluxo de cadastro e alteração de uma ocorrência vinculada ao usuário atualmente autenticado no sistema que seja criador ou responsável pela mesma.

The screenshot displays the 'Ocorrências' (Incidents) page in the BugSys application. The sidebar on the left contains the following menu items: Dashboard, Clientes, Projetos, Workflows, Usuários, and Ocorrências. The main content area has a header with the title 'Ocorrências' and the user 'antonio.junior'. Below the header are two buttons: 'Salvar' (Save) and 'Cancelar' (Cancel). The form itself consists of several dropdown menus for the following fields:

- Projeto:** SIG - Sistema Integrado de Gestão do Governo do Ceará
- Tipo de ocorrência:** Defeito
- Usuário responsável:** Rafael Stain Cassau
- Workflow:** Cascata
- Fases do Projeto:** Desenvolvimento
- Casos de Uso:** Manter Membros
- Status:** Suspensão

Below the form, there is a list of recent incidents:

- Antonio Junior**
Esta listando os membros duplicados na consulta.
10/11/2013
- Rafael Stain Cassau**
suspensão.
10/11/2013

At the bottom of the form is a text area labeled 'Descrição da Ocorrência:'.

Figura 25 - Cadastro e Alteração de Ocorrências

Uma ocorrência é composta por um tipo, podendo ser erro, defeito, falha, melhoria, sugestão ou solicitação de mudança, possui também um usuário criador e um usuário responsável pela resolução da mesma, podendo esse mudar de acordo com cada alteração do *status* da ocorrência. Uma ocorrência também está vinculada sempre a 1 (um) caso de uso, que faz parte de um projeto, e sempre estará relacionada a uma fase do *workflow* vinculado ao projeto e, a cada alteração na

ocorrência, seja ela de qualquer tipo, sempre um comentário deve ser adicionado a fim de manter um histórico da mesma.

3. CONSIDERAÇÕES FINAIS

Com a crescente demanda por *softwares* cada vez mais elaborados e complexos, as equipes de desenvolvimento de *softwares* vem se tornando cada vez mais especializadas e com grande número de profissionais, porém a comunicação informal entre esses membros no dia a dia do desenvolvimento é prejudicial pois não existe maneira de medir a produtividade e qualidade do que esta sendo desenvolvido, também se torna bem difícil a tomada de decisões gerencias a fim de melhorar o processo e diminuir o gargalo em alguns setores da produção.

Pensando nesse e em outros problemas foi proposto o desenvolvimento de um sistema capaz de formalizar a geração de ocorrências existentes entre os membros da equipe em diferentes fases do processo, tais como, as fases de análise, desenvolvimento, testes entre outras.

Para tal propósito foi realizado um levantamento bibliográfico a fim de aprofundar os conhecimentos em engenharia de *software*, especificamente em suas metodologias de desenvolvimento e seus diferentes processos. Esse levantamento foi de fundamental importância, pois possibilitou uma visão sistêmica e detalhada do projeto.

Após o levantamento bibliográfico foi feita uma análise em alguns *softwares* semelhantes a fim de descobrir o que de melhor cada um poderia oferecer e realizar o levantamento de requisitos.

Com o término do levantamento de requisitos foi feita uma análise rigorosa a fim de modelar o sistema, após a análise e separação de funcionalidades em casos de uso foi realizado o desenvolvimento utilizando as melhores tecnologias *open source* disponíveis atualmente no mercado. Uma preocupação constante foi desenvolver o sistema de maneira que sua *interface* gráfica fosse simples e intuitiva para fornecer uma melhor experiência para o usuário.

Com o término do desenvolvimento a aplicação foi testada e a cada erro encontrado e corrigido a funcionalidade era testada novamente com o propósito de fornecer um sistema confiável e com baixa taxa de manutenção corretiva.

Após o termino dos testes o sistema foi apresentado e está apto a ser utilizado em um ambiente de produção. Futuramente novas funcionalidades serão desenvolvidas, tais como um rigoroso painel de gerenciamento de acessos a funcionalidades por usuário, um modulo responsável por exibir o *dashboard* do

usuário e gerar relatórios a fim de fornecer uma melhor interação entre os membros da equipe, clientes e processos de desenvolvimento de *software*.

REFERÊNCIAS BIBLIOGRÁFICAS

ASTAH, **Change-Vision, Astah Community 6**, disponível em https://members.change-vision.com/files/astah_community/6_7_0/astah-community-6_7_0-43495-jre-setup.exe, Acessado em 10 jun. 2013.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**, 2ª Ed., Rio de Janeiro: Elsevier, 2007.

FOWLER, M. **UML Essencial – Um breve guia para a linguagem-padrão de modelagem de objetos**, 3ª Ed., Porto Alegre: Bookman, 2005.

LIMA, Adilson da Silva. **UML 2.3: do requisito à solução**, 1ª Ed., São Paulo: Érica, 2011.

PMI. **Conjunto de Conhecimentos em Gerenciamento de Projetos (PMBOK)**, 4ª Ed., *Pennsylvania: Project Management Institute (PMI)*, 2009.

PRESSMAN, Roger S., **Engenharia de Software – Uma abordagem profissional**, 7ª Ed., São Paulo: Artmed, 2011.

SOMMERVILLE, I. **Engenharia de Software**, 6ª Ed., São Paulo: Pearson, 2004.

TELES, Vinícius M. **Extreme Programming – Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**, 1ª Ed., Rio de Janeiro: Novatec, 2006.