UNIVERSIDADE DE CAXIAS DO SUL

GIOVANE DA SILVA BERTOL

TESTES AUTOMATIZADOS COM PL/SQL NA VALIDAÇÃO DE ROTINAS DE BANCO DE DADOS ORACLE

CAXIAS DO SUL
2014

GIOVANE DA SILVA BERTOL

PROTÓTIPO PARA TESTES AUTOMATIZADOS COM PL/SQL NA VALIDAÇÃO DE ROTINAS DE BANCO DE DADOS ORACLE

Trabalho de conclusão de curso, apresentado como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação pela Universidade de Caxias do Sul, Curso de Sistemas de Informação.

Orientador: Prof. Iraci Cristina da Silveira de Carli.

CAXIAS DO SUL
2014

RESUMO

Com o grande aumento na utilização de softwares para gerenciamento e tomadas de decisões nas empresas, a indústria de software passou a ser muito demandada e consequentemente a qualidade e agilidade na entrega de software passaram a ser atributos de obrigatoriedade. A utilização de testes de software tornou-se essencial e indispensável para as empresas de desenvolvimento, obrigando-as a aderir e moldar técnicas de teste de software a fim de atender a demanda em um nível aceitável de qualidade de produto. O tempo para os testes normalmente não são inclusos em um projeto de software com um tempo adequado para aderir qualidade ao processo. Sendo assim, os testes precisam ser flexíveis a cada processo de desenvolvimento, buscando soluções para executá-los e escrevê-los de forma ágil e eficaz. Portanto, sempre que possível, um teste de software deve ser automatizado e reutilizável, visando diminuir a interação humana no processo. Este projeto teve por objetivo o desenvolvimento de um protótipo, a ferramenta *AllInOne Test Automation*, para automação de testes de software. O protótipo possui uma arquitetura de software para automação de testes, incluindo os processos de planejamento, desenvolvimento, execução, análise, medição e gerenciamento de objetos de testes.

ABSTRACT

The increasing use of software for management and decision making in the companies, got the software industry very demanded and consequently the quality and speedy delivery of software became mandatory attributes. The use of software testing has become an essential and indispensable for development companies, forcing them to get and shaping techniques for testing software in order to meet demand at an acceptable level of product quality. The time for testing is usually not included in a software project with a suitable time to adhere to the quality process. Thus, the tests need to be flexible to each development process, seeking solutions to run them and write them quickly and efficiently. Therefore, whenever possible, a test software should be automated and reusable, aiming to reduce the human interaction in the process. This project aims to develop a prototype, the AllInOne Test Automation, tool for test automation software. The prototype had a software architecture for automation of tests, including planning, development, implementation, testing, measurement and management of test objects.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UML	UNIFIED MODELING LANGUAGE (LINGUAGEM DE MODELAGEM UNIFICADA)
SGBD	SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS
IDE	*INTEGRATED* DEVELOPMENT *ENVIRONMENT (*AMBIENTE INTEGRADO DE DESENVOLVIMENTO)
ERP	*ENTERPRISE RESOURCE PLANNING (SISTEMA INTEGRADO DE GESTÃO EMPRESARIAL)*
MVVM	*MODEL VIEW VIEWMODEL*
MRP	MANUFACTURING RESOURCE PLANNING (PLANEJAMENTO DE RECURSOS DE MANUFATURA)

SUMÁRIO

1 INTRODUÇÃO

Conforme Sommerville (2011), teste de software destina-se a mostrar que um programa faz o que se propõe. Portanto, o teste, caso implementado corretamente, poderá aumentar consideravelmente a qualidade de entrega de um software.

"O teste unitário ou de unidade, é o processo de testar os componentes de programa como métodos ou classes" (Somerville, 2011). De acordo com Pressman (2006), o teste de unidade enfoca a lógica interna de um procedimento e de suas estruturas, tendo como limite o componente. O teste de unidade pode ser escrito antes que o código seja iniciado, proposto pelas abordagens ágeis[1], ou depois do código fonte ter sido desenvolvido.

O teste de regressão, conforme Pfleeger (2004), é utilizado para garantir que uma funcionalidade já existente executa corretamente após uma nova versão do programa. Sommerville (2011), também afirma que testes de regressão, podem ser executados para verificar se uma mudança no programa não introduziu novos erros.

Ainda de acordo com Sommerville (2011), sempre que possível, um teste deve ser automatizado utilizando um *framework* para tal propósito. A automatização de um teste, desde que bem estruturada, poderá suprir os testes manuais que são dispendiosos e custosos podendo onerar a equipe.

A automação de testes requer um trabalho dedicado. Portanto para ter sucesso e atingir os resultados esperados, deve-se planejar e identificar quais testes serão utilizados e de qual forma montá-los e, somente após, pensar na automação das partes mais onerosas, (Costa 2004 apud BACK,1996, p. 15). Ainda Costa (2004 apud KANNER,1997, p. 15) relata, que um teste automatizado pode ser entre 3 à 10 vezes mais custoso para ser criado, valido e documentado, do que testes manuais.

O teste automatizado é mais trabalhoso e caro nas primeiras execuções. Após tende a ser mais vantajoso que o teste manual, sendo hoje uma opção atrativa para as empresas que desejam aplicar qualidade em seus ciclos de desenvolvimento.

[1] Padrões de projetos ágeis.

1.1 PROBLEMA E QUESTÃO DE PESQUISA

Com base na ASSESPRO (Associação das Empresas Brasileiras de Tecnologia da Informação), na medida em que o uso dos softwares cresce em boa parte dos processos nas organizações, a pressão para o desenvolvimento de software em curto espaço de tempo aumenta. Esta pressão gera necessidades de novos processos para garantir a qualidade e confiabilidade de um sistema.

O modo convencional de desenvolvimento de uma funcionalidade é estudar o problema, elaborar uma solução e a seguir desenvolver. Após estas etapas os testes são realizados pelo desenvolvedor de modo manual. Assim, caso existirem erros, o desenvolvedor precisará identifica-los e corrigi-los.

Quando o sistema torna-se muito grande e complexo, a manutenção e execução de testes manuais são dificultadas por serem dispendiosas e cansativas, por consequência, normalmente todos os testes não conseguem ser validados. É neste cenário que surgem os erros de software.

Pode-se concluir que existe alto investimento para manutenção dos testes, pois eles precisam ser exercitados um à um e corrigidos caso existirem modificações nas funcionalidades. Com isso, faz-se necessário a utilização de ferramentas mais eficazes e flexíveis para a geração automatizada de testes.

Uma ferramenta que auxilia a fase de testes de forma automatizada, poderá alcançar os seguintes benefícios: compensar a falta de uma equipe de testes, ter maior abrangência sobre a funcionalidade a ser testada, possuir mais velocidade na execução de testes e apresentar elevado volume de exercícios de uma funcionalidade.

Baseando no problema de pesquisa elencado anteriormente foi criada a seguinte questão de pesquisa:

“Como aprimorar o exercício de testes de unidade e regressão, para programas desenvolvidos em linguagem PL/SQL, aumentando a produtividade de testes de software e a qualidade de entrega?”.

1.2 OBJETIVOS

O objetivo deste trabalho é desenvolver um protótipo de uma ferramenta para geração, armazenamento e execução de testes automatizados. Essa atenderá

especificamente os tipos de teste como: testes de regressão e de unidade, ambos escritos em linguagem PL/SQL que utilizam procedimento armazenados no banco de dados Oracle. Este trabalho terá como foco a linguagem PL/SQL, contudo a ferramenta estará estruturada para acomodar outras linguagens no futuro.

Para automatizar testes na linguagem PL/SQL, foi observado que existem apenas aplicações comerciais, sendo assim por conta de custos as empresas, esses testes geralmente são realizados manualmente, com auxílio de uma ferramenta para testes de unidade.

A metodologia para o desenvolvimento deste trabalho dar-se-á utilização de cinco etapas:

a) Fundamentação teórica sobre testes de software e testes automatizados, com o intuito de embasar e referenciar os conceitos que serão utilizados no decorrer deste trabalho.
b) Especificações dos requisitos que o protótipo deverá atender, criando assim os requisitos funcionais.
c) Modelagem do protótipo, gerando os artefatos de: diagrama de caso de uso, diagrama de classes e diagrama de sequência. Essa modelagem utilizará os padrões de projeto para desenvolvimento. Serão utilizados: O MVVM (Model, View, ViewModel), Singleton, Facade e DAO.
d) Desenvolvimento do protótipo, terá como base a criação de testes de unidade e de regressão, automatizados para a linguagem PL/SQL, criando uma estrutura genérica. Assim possibilitando a agregação de futuras novas linguagens em um projeto de expansão. A ferramenta será desenvolvida na linguagem C# com auxílio da IDE (*Integrated Development Environment*)[2] *Visual Studio 2013 Express*[3], ambas da empresa Microsoft, sendo uma aplicação para área de trabalho. O protótipo utilizará o framework de testes de unidade utPLSQL armazenado no banco de dados Oracle.
e) Testar a ferramenta utilizando um projeto de desenvolvimento com números e dados fictícios.

[2] Ambiente de Desenvolvimento Integrado.
[3] A IDE Visual Studio 2013 Express é de uso gratuito inclusive para aplicações comerciais.

1.3 ORGANIZAÇÃO DO TRABALHO

Este trabalho está dividido em capítulos, com o objetivo de linearizar o conhecimento teórico até a proposta de solução.

No capítulo 2 é apresentado o referencial teórico para a modelagem e desenvolvimento do protótipo.

O capítulo 3 apresenta o desenvolvimento da proposta deste trabalho, destacando-se a modelagem e prototipagem de tela.

No capítulo 4 é apresentado a implementação da ferramenta, demostrando estrutura de projetos e a forma de instalação.

O capítulo 5 demonstra uma simulação de testes, com o objetivo de testar a ferramenta, validar seus requisitos e apresentar suas funcionalidades.

2 TESTES DE SOFTWARE

O objetivo de um teste é revelar defeitos e um bom teste tem alta probabilidade de encontrar erros, conforme Pressman (2006). Além disso o teste deve demonstrar que um programa faz o que a ele é proposto, de acordo com Sommerville (2011).

Para Sommerville (2011) os testes de software possuem dois objetivos: demonstrar ao desenvolvedor e ao cliente que o software atende o que foi requisitado e identificar diferentes situações onde o software se comporta de maneira incorreta.

Dentro de um processo de teste, “O teste é parte de um amplo processo de verificação e validação (V&V). Verificação e validação não são a mesma coisa” (SOMMERVILLE, 2011, p. 144). Sommerville (2011 apud BOEHM, 1979, p. 144), expressou a diferença entre os processos de validação e verificação:

- Validação: Estamos construindo o produto certo?
- Verificação: Estamos construindo o produto da maneira certa?

Esses processos têm por objetivo indicar se o software realmente satisfaz as especificações e oferece as funcionalidades esperadas pelo pagante. Ainda, os processos de verificação e validação continuam em todas as fases do desenvolvimento.

Um bom teste objetiva encontrar o maior número de erros com o mínimo de esforço, relata Pressman (2006). O Engenheiro de Software deve projetar e implementar uma estrutura com “testabilidade”. Estrutura com “Testibilidade” é a estrutura passível de teste, isto é, sua arquitetura é pensada para a possiblidade de testes.

Pressman (2006) ainda releva algumas características que um teste com testabilidade deve ter:

- Operabilidade: O sistema deve ser implementado com qualidade, sendo assim poucos defeitos irão impedir a execução de testes.
- Observabilidade: O código fonte deve ser facilmente acessível, onde estados e variáveis do sistema são visíveis ou consultáveis durante a execução do código.
- Controbilidade: Quanto melhor o controle sobre as rotinas de software, maiores serão as possibilidades de automatização de testes.

- Decompobilidade e Simplicidade: Isolação de camadas do software, criação de rotinas pequenas, com alta possibilidade de isolamento para testes.
- Compreensibilidade: Quanto mais informações ao dispor, maior será a capacidade de criação de testes racionais, diminuindo o tempo e o esforço para criação de testes eficazes.

Um teste não pode demostrar que um software é livre de erros, pois ele ajuda a encontrar erros o que diminui as chances de um cliente o descobrir, denigrindo assim a imagem perante a qualidade do software desenvolvido.

Neste capítulo serão apresentados os fundamentos de testes, os quais serão utilizados para o desenvolvimento da ferramenta proposta neste trabalho.

2.1 ESTRATÉGIAS DE TESTE DE SOFTWARE

Segundo Pressman (2006), uma estratégia de teste de software integra métodos para o planejamento de casos de teste. A estratégia condiz em definir a estrutura básica para projeção de casos de teste, sua execução, a coleta dos resultados e avaliação.

Para agregar a definição de estratégias de software, Rios e Moreira Filho (2013) têm a concepção na qual, estratégias de testes devem formar a base para formulação de um plano de teste. Para isso devem atender as características citadas a seguir:

- Descrever como o software será testado.
- Identificar níveis de testes.
- Métodos de testes utilizados.
- Técnicas e ferramentas.

Para facilitar o planejamento e estruturação de testes, Pressman (2006) aborda a utilização de métodos de testes, que são utilizados para a definição do quanto será aprofundado, em nível de estrutura de um componente. Os métodos de testes são explicados na seção 2.3.

É necessário ainda, para a escolha dos testes à serem elaborados e planejados, a definição de tipos de teste, explanados na seção 2.4.

Além de métodos de testes é necessário criar planos de testes, ou seja, o que os testes irão validar e de qual forma será descrito para fácil entendimento. Nesta situação, far-se-á utilização de caso de teste.

2.2 MÉTODOS DE TESTES

Os métodos de testes são aplicados a cada estágio de teste, por isso devem ser definidos na estratégia de testes como predecessores. Cada estágio de teste significa um nível a ser testado, como por exemplo, o nível de teste de unidade (Rios e Moreira Filho 2013).

Conforme Pressman (2006), qualquer produto que passe por engenharia e pode ser testado, pelo menos uma função que esse produto desempenha conforme sua especificação, poderá demonstrar se está operacional e livre de erros, sendo chamado de teste de caixa-preta.

Também o produto pode ser testado com base no conhecimento interno, isto é, testando todas as combinações possíveis do produto, chamando-o assim de teste de caixa-branca (Rios e Moreira ,2013).

Para complementar o entendimento, Pfleeger (2004), exalta que em um teste de caixa-preta fornecemos todas as entradas possíveis e comparamos com o resultado esperado conforme os requisitos. Quando considerado de caixa-branca, pode-se examinar a lógica interna gerando testes mais abrangentes.

2.2.1 **Teste de Caixa-Branca**

O teste de caixa-branca, de acordo com Pressman (2006), utiliza uma estrutura de controle de componentes para derivar casos de teste nos quais garantam que:

- Os caminhos possíveis de um componente tenham sido testados pelo menos uma vez;
- Exercitem os caminhos lógicos;
- Executem os ciclos nos limites e dentro dos intervalos de classes operacionais;
- Testem as estruturas de dados internas para garantir a validade.

Sommerville(2011) em uma abordagem mais simplória, observa que um teste de caixa-branca é aquele que pode-se olhar o código fonte, com o intuito de encontrar novos testes possíveis. Em um exemplo descrito por Sommerville (2011), utilizando testes de caixa-branca, onde no código exista o tratamento de exceções, pode-se criar uma partição de exceção para exercitar também esta situação.

Um teste de caixa-branca muito rigoroso, segundo Pressman (2006), poderia levar a total qualidade de um software desenvolvido. Mas essa abordagem é problemática, pois a escrita e o raciocínio dos testes seria imensamente dispendioso. O ideal é escolher os caminhos mais críticos para serem testados.

2.2.2 **Teste de Caixa-Preta**

Um teste de caixa-preta, segundo Pressman (2006), busca testar requisitos funcionais do software. Permite a um engenheiro de software, por exemplo, elaborar conjuntos de condições de entradas que possam exercitar todos os requisitos funcionais do software.

Segundo Pfleeger (2004), o teste de caixa-preta utiliza-se de especificações de documentação para a composição dos testes. Sommerville (2011) relata que podem ser utilizadas as especificações de um requisito para identificar partições de equivalência e não é necessário o conhecimento de como funciona o sistema.

Pressman (2006) descreve que o testes de caixa-preta, diferentemente do de caixa-branca, tende a ser aplicado nos últimos estágios do processo. A atenção é focalizada nas informações retornadas pelas funções. Os testes podem ser compostos para responder perguntas, conforme a tabela 1.

Tabela 1 - Perguntas que podem ajudar na utilização do método de caixa preta

Descrição da Pergunta
Como a validade funcional é testada?
Que classes de entradas vão constituir bons casos de testes?
O sistema é particularmente sensível a certos valores de entrada?
Como são isolados os limites de uma classe de dados?
Que efeitos as combinações de dados vão ter na operação do sistema?

Fonte: Pressman (2006), adaptado pelo autor.

2.3 TIPOS DE TESTE

Em tipos de testes estão incluídos os diversos métodos para descoberta de defeitos, de acordo com Pfleeger (2004 apud JONES,1991, p. 290), esses têm o fim de determinar quais têm a maior probabilidade de encontrar erros.

Para este trabalho serão utilizados especificamente os testes de unidade e os testes de regressão.

2.3.1 **Testes de Unidade**

Segundo Sommerville (2011), testes de unidade podem ser chamados também de testes unitários. São testes que tem a finalidade de testar a menor unidade de um programa, ou seja, funções individuais ou métodos da forma mais simples de um componente.

Os testes devem testar procedimentos e funções com diferentes parâmetros de entrada. Com base em Pressman (2006), “os testes de unidade enfocam a lógica interna de processamento e as estruturas de dados dentro dos limites de um componente”, ou seja, devem ser testados com uma estrutura de dados que possa retornar valores corretos.

Para o desenvolvimento de um modelo de testes, Pressman (2006) inclui de forma procedimental condições para o teste de um módulo. São estes:

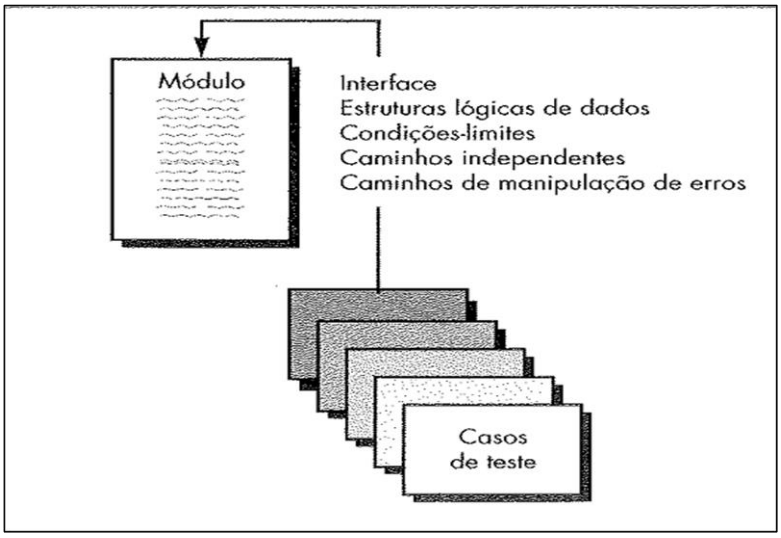
- A interface é testada com o objetivo de garantir que a informação flua de acordo, tanto na entrada como na saída.
- A estrutura de dados, por sua vez, é verificada de modo que garanta que os dados armazenados temporariamente mantenham sua integridade durante todos os passos de execução do algoritmo.
- Todos os caminhos independentes ao longo da estrutura devem ser exercitados.
- As condições-limites são testadas a fim de garantir que o módulo opere corretamente nos limites estabelecidos.
- Todos os caminhos de manipulação de erros devem ser testados.

De acordo com Pressman (2006), “Testes nos limites é uma das mais importantes tarefas do teste de unidade”. Isto se dá pela necessidade de utilizar

valores para testes conforme o máximo ou mínimo especificado nos componentes, pois assim provavelmente descobrirão erros.

A figura 1 ilustra um esquema para testes de unidade.

Figura 1 - Testes de Unidade



Fonte: Pressman (2006, p. 295).

2.3.2 **Testes de Regressão**

O teste de regressão, conforme Pfleeger (2004), é utilizado para garantir que uma funcionalidade já existente, execute corretamente após uma nova versão do programa. Sommerville (2011), também afirma que testes de regressão podem ser executados para verificar se uma mudança no programa não introduziu novos erros.

Sendo assim o teste de regressão é a reexecução de algum subconjunto de teste, garantindo que novas mudanças não afetaram o comportamento correto de programas já desenvolvidos (Pressman, 2006).

Para criação e manutenção de uma suíte de testes de regressão, Pressman (2006), elege três diferentes classes de casos de teste:

- Uma amostra de testes, para exercitar todas as funções do software.
- Testes adicionais que provavelmente serão afetadas pelas modificações.

- Testes que integram as funções que foram modificadas.

À medida que o teste de regressão aumenta de proporção, é necessário avaliar e utilizar somente os testes mais eficazes, aqueles que testam as maiores classes de erros (Pressman, 2006).

2.4 PLANOS DE TESTE

O plano de teste para sistemas maiores, na concepção de Sommerville (2007), compreende a definição de recursos tanto de hardware como de software necessários. Ainda um plano de teste deve estabelecer o cronograma de teste e os procedimentos de teste.

Para Rios e Moreira Filho (2013) um plano de teste é um documento que identifica o desenho lógico do processo de teste e tem algumas funções essenciais como:

- Avaliar os requisitos do sistema para o processo de testes.
- Avaliar a base para criação da massa de dados para os testes.
- Definição o tamanho do projeto de teste.
- Definir ambiente de teste.
- Determinar os recursos humanos envolvidos.
- Declaração de tipos de testes.
- Métodos de testes, definindo a forma na qual serão executados os testes.
- Definição de critérios de aceitação dos resultados dos testes.
- Definir um cronograma de testes.

De acordo com Pfleeger (2004), o plano de teste pode servir também como resumos do processo de teste para o software. Com base na figura 2 é possível ter uma prévia da estrutura de documentos gerados por um plano de teste.

Figura 2 – Documentos produzidos durante o processo de teste.

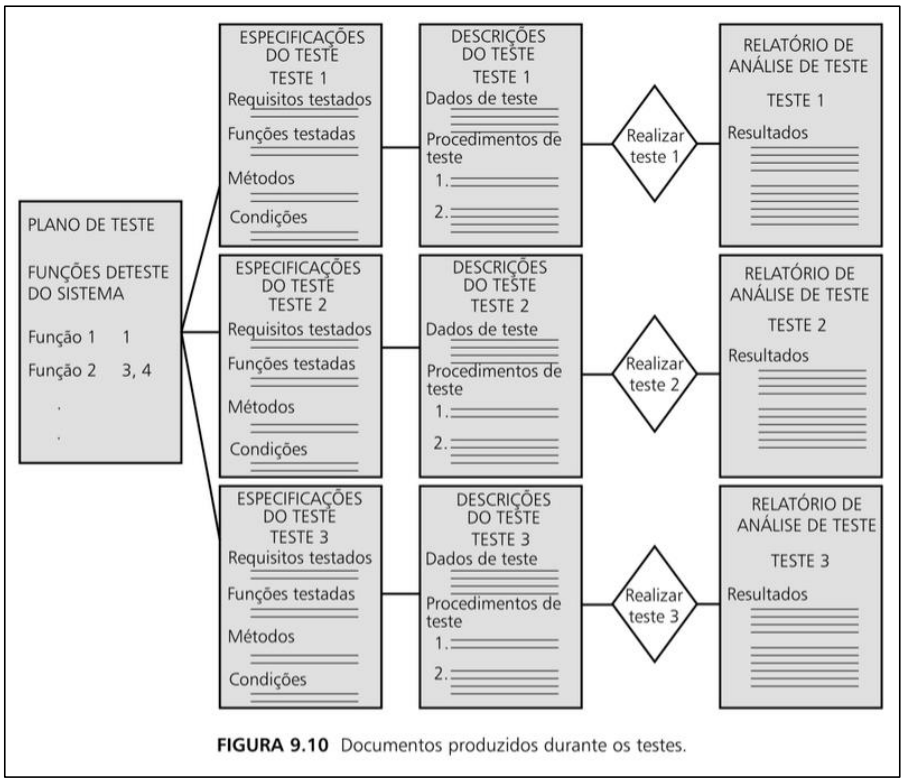


FIGURA 9.10 Documentos produzidos durante os testes.

Fonte: Pfleeger (2004).

O conceito de plano de teste será utilizado para avalição e levantamento dos requisitos de software na modelagem do protótipo.

2.5 CASOS DE TESTE

Um caso de teste deve definir o que vai ser testado, por exemplo: pode ser criado um caso de testes que analise se determinada unidade realiza com sucesso o que dela se espera e, outro caso de teste, que simule entradas inválidas que possam gerar erros na unidade (SOMMERVILLE, 2011).

Para a escolha de um caso de teste, Pfleeger (2004) aborda que ao testar um componente, é realizada a escolha dos dados e as condições de entrada, permitindo assim que um componente os manipule e observe a saída. Sendo assim,

em um caso de teste, deve-se ter um conjunto de dados para entrada e um conjunto de resultados esperados, que são as saídas.

A qualidade de um caso de teste, segundo Pfleeger (2004), é descrita por atributos, sendo estes:

- Procura encontrar defeitos;
- Procura testar mais aspectos, com o intuito de diminuir os casos de teste;
- Menor custo para a engenharia do caso de teste, como tempo despendido;
- Menor custo de manutenção dos casos de teste já elaborados e implantados.

Pfleeger (2004) relata um exemplo para criação de um caso de teste, quando um componente esperar um valor positivo como entrada, pode-se incluir um caso de teste para forçar erros, com os seguintes valores de entrada:

- Um inteiro positivo muito grande, para testar estouro de variável por exemplo;
- Um inteiro positivo;
- Um positivo decimal de ponto-fixo;
- Um número maior que 0, mas menor que 1;
- Zero;
- Um número negativo;
- Um carácter não numérico;

Os exemplos acima forçarão situações não tratadas, podendo encontrar erros, portanto a utilização de dados inesperados é necessária.

2.5.1 **Identificação de um caso de teste**

Sommerville (2011) propõe que sejam utilizados casos de testes que validam o comportamento do componente, ou seja, se realmente ele faz o que se propõe e outro que será baseado em diretrizes, que refletirão a experiência anterior com base nos erros frequentes dos desenvolvedores. Essas metodologias de identificação de testes auxiliam na criação de casos de testes eficazes.

2.5.1.1 Validação do comportamento do componente

Na validação do comportamento de um componente é necessário analisar se os resultados esperados foram atingidos, baseado em retorno da execução de um teste.

Para exemplificar, a tabela 2 representa um requisito levantado após a análise e com base nele será elaborado um caso de teste.

Tabela 2 - Exemplo de Validação de um requisito

1. Requisito 1	
Assunto:	Cálculo da Margem de Contribuição
Detalhamento:	Rotina para retornar valor de percentual de faturamento e matéria prima, conforme fórmulas abaixo. Percentual Faturamento Líquido = 100% (Sempre) Percentual da Matéria Prima = Valor Matéria Prima / Faturamento Líquido

Fonte: O autor.

Na tabela 2 pode ser observado que é necessário realizar a alteração do cálculo da margem de contribuição. A fim de validar o comportamento do componente que realiza esse cálculo e retorna o valor final do percentual da matéria prima, deve-se prever valores numéricos de entrada e saída, para o cálculo da fórmula:

$$\%MP = (MP / FAT) * 100$$ [4]

Estes valores de entrada devem incluir classes numéricas que indicam grupos passíveis de entrada na rotina, como exemplifica a tabela 3.

Tabela 3 - Grupos de entrada de dados

Grupos	Possíveis valores
Números negativos	< 0
Zero	0
Números positivos	>= 1
Valores nulos	Nulo
Tratamento de	Mensagem de Retorno
Carácter	Cai no tratamento de exceção.

Fonte: Autor

[4] Percentual da Matéria Prima = (Valor Bruto Matéria Prima / Valor de Faturamento) * 100

Através destes grupos, deve-se identificar os valores de retorno da rotina e caso estes valores saírem de acordo com o esperado, o comportamento do componente é valido.

Na tabela 4 existe um grupo de tratamento de exceção que também deve ser validado e treinado, como por exemplo: para o cálculo da margem de contribuição. Uma das especificações esperada é que o percentual da matéria prima nunca seja menor que 0, logo se um valor de entrada ser um número negativo, uma exceção deve ser lançada informando a existência de invalidade da operação.

Se o valor de entrada para a variável do valor de faturamento ser 0, o procedimento irá lançar uma exceção indicando a existência de uma divisão por 0, devendo ser tratada e esperada pelo grupo de tratamento de exceções.

A tabela 4 demonstra os valores de entrada passados a rotina de cálculo da margem de contribuição e seus resultados esperados, dando-se pela fórmula:

$$\%MP = (MP / FAT) * 100$$

Tabela 4 – Possíveis entradas e saídas esperadas para cálculo da margem de contribuição

Grupos	1°Entrada	2°Entrada	Saída Esperada
Números negativos	- 5	2	Exceção de Valor Negativo
Números negativos	1000	- 5000	Exceção de Valor Negativo
Carácter	Y	2	Exceção de Erro numérico
Zero	10	0	Exceção de divisor igual a 0
Números positivos	50	20	2,5 %
Números positivos	1000	10000	10 %
Valores nulos	Nulo	1000	0 %[5]
Valores nulos	100	Nulo	10 %[6]

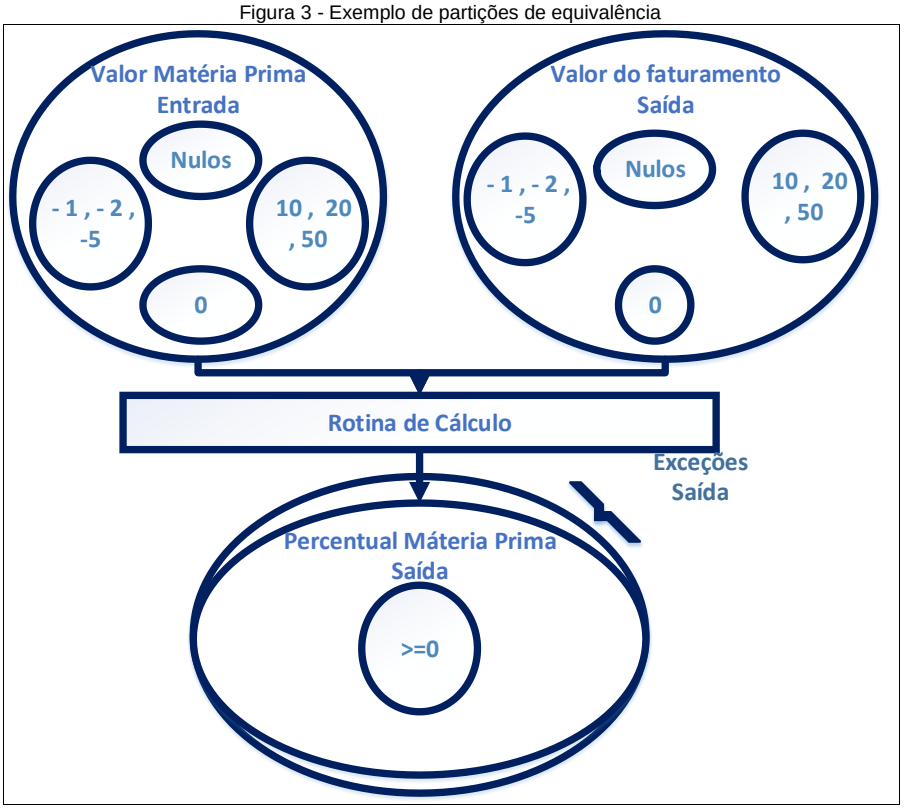
Fonte: Autor.

Conforme Sommerville(2011), estes grupos numéricos podem ser classificados como estratégia de partições de equivalência de entrada e saída, onde é necessário identificar um conjunto de partições em um componente, para que seja possível escolher os casos de testes.

Na figura 3 é ilustrado um exemplo de partições de equivalência, especificando conjuntos de dados possíveis de serem utilizados na rotina de cálculo. Quanto mais diversificado os conjuntos de dados, maiores serão as chances de encontrar erro no procedimento.

[5] O valor nulo deve ser tratado no caso de cálculo.
[6] O valor nulo deve ser tratado no caso de cálculo.

Figura 3 - Exemplo de partições de equivalência



Fonte: O Autor.

Portanto, para que um componente seja validado quanto à sua funcionalidade, um teste de unidade deve prever entradas de dados e comparar as saídas esperadas para este teste.

2.5.1.2 Validação com base em diretrizes

Segundo Sommerville (2011), diretrizes encapsulam conhecimento de tipos de casos de testes que são eficazes a descobrir erros. Essas diretrizes refletem experiências anteriores, dos tipos de erros cometidos com frequência pelos programadores no desenvolvimento de componentes.

Por exemplo, um programador que realiza seus testes em um componente, normalmente fornece valores de entrada que sejam válidos, esperando sempre que o resultado esperado seja retornado. Desta forma, toma-se como uma diretriz, a

inclusão de grupos de valores inválidos de entrada. Para um caso de teste, força-se um lançamento de uma exceção, que deve ser tratada pelo componente e retornar mensagem ou um valor esperado para um valor inválido.

Sommerville (2011 apud WHITTAKER ,2002, p. 150) inclui alguns exemplos de diretrizes que podem ser usadas no projeto de caso de teste:

- Escolha entradas que forcem o sistema gerar todas as mensagens de erro;
- Projete entradas que causem *overflow* de *buffers* de entrada;
- Repita a mesma entrada ou uma série, inúmeras vezes;
- Obrigue a geração de saídas inválidas;
- Obrigue os resultados de cálculos a serem muito grandes ou muito pequenos;

Conforme a experiência na elaboração de caso de teste for crescendo, pode-se criar diretrizes próprias que se adequem a organização.

2.5.2 **Definição de caso de teste**

A definição de um caso de teste é executada posteriormente ao levantamento, análise e complemento de requisitos, com o objetivo de deliberar o desenvolvimento dos testes.

Na elaboração de um caso de teste é necessário levantar o que deverá ser testado e como. Para isto, precisa estar claro que tipos de dados serão utilizados em cada componente e quais caminhos poderão ser seguidos pela rotina.

Os testes de unidade poderão exercitar testes conforme demonstrado na tabela 5. Também é possível escolher outros testes que forem julgados como importantes.

Tabela 5 - Testes Exercitados por Unidade

Tipos de Testes	Testes Exercitados
Erros de Cálculo	Precedência Aritmética Inicialização Incorreta Falta de Precisão
Comparação de Dados	Tipos de Dados diferentes Operadores ou operação lógica incorretos Expectativa de igualdade quando um erro de precisão torna a igualdade improvável Terminação de ciclos inadequada ou

	inexistente Falha na saída, quando iteração divergente

Fonte: Autor, baseado em Pressman (2006).

Quanto à precedência aritmética, o teste de unidade pode mostrar facilmente um erro, pois o valor de entrada deverá resultar em uma saída esperada, e no caso de precedência incorreta, este valor será divergente, assim resultando em falha do componente.

O analista ou desenvolvedor também poderá analisar a inicialização de variáveis, por exemplo, na execução de um teste de um componente. O desenvolvedor poderá esquecer de declarar uma variável ou atribuir a ela um valor incorreto, portanto testando todos os caminhos possíveis, os erros de inicialização de variáveis poderão ser identificados.

Em outra situação em que um caso de teste pode ser utilizado, é o teste de precisão de variáveis. Este teste pode avaliar se passando valores, com casas decimais, o componente utiliza formas de arredondamento equivalentes em todo o processo, retornando o valor esperado.

Um caso de testes também deverá prever dados de entrada que forcem a iteração dos laços existentes no componente. Estes dados precisam validar os limites dos índices que o laço atua, para que assim seja testado as possíveis situações que possam gerar erros.

Com este planejamento é possível determinar um caso de teste para uma especificação, que deve identificar todos os testes necessários para a qualidade da entrega do software.

2.5.3 **Estrutura básica de um caso de teste**

Os casos de teste devem ser elaborados com base no objeto deliberado pelo plano de teste. Para Rios e Moreira Filho (2013, p. 78), “O caso de teste desce ao nível de detalhe de campos, formulários, arquivos, telas, páginas e outros. Cada item de teste deve ter um critério de avaliação e teste”.

Um caso de teste pode possuir a estrutura, que é a proposta mencionada por Rios e Moreira Filho (2013), conforme a tabela 6.

Tabela 6 - Estrutura de um caso de teste

Item da estrutura	Descrição
Itens ou campos	Relaciona itens a serem testados e como serão executados. Deve ter uma descrição detalhada para cada item de modo individual.
Entradas	Como os dados de entrada serão dispostos para o caso de teste: Manual, através de arquivos ou tabelas de dados.
Resultados Esperados	Resultados esperados para campos ou itens específicos.
Interdependências	Se o caso de teste, depender de outro para sua execução.
Necessidade de ambiente	Necessidades adicionais de equipamentos, ferramentas e pessoal, (Além das já disponíveis) para execução dos testes. O assunto deve ser tratado em detalhes na etapa de preparação do teste.
Datas	Início e término da elaboração e execução dos casos de teste.

Fonte: O Autor.

2.6 TESTES AUTOMATIZADOS

Com o aumento da produtividade do desenvolvimento de softwares, proporcionado pelas técnicas e ferramentas de desenvolvimento, também surgiu a necessidade de maior produtividade na construção e execução de testes. Conforme Costa (2004), a automação de testes é uma forma de atingir essa alta produtividade.

> Existe uma diferença clara entre testes e automação de testes. No primeiro você realiza a tarefa de testar, e no segundo você usa um software que imita a interação com a aplicação no que se refere ao teste tal qual um ser humano faria (com algumas limitações) (Molinari, 2010 apud GRAHAM e FEWSTER, 1999, p.37).

Portanto, um software irá testar após o desenvolvimento do teste sem a interação humana.

Como complemento, Sommerville (2011) também esboça que sempre que possível, um teste deve ser automatizado, utilizando um *framework* para tal propósito. A automatização de um teste, desde que bem estruturada, poderá suprir alguns testes manuais, reduzindo o envolvimento humano em atividades repetitivas.

A automação de testes requer um trabalho dedicado e para ter sucesso e atingir os resultados esperados, deve-se planejar e identificar quais testes serão

utilizados e de qual forma montá-los, e somente após, pensar na automação das partes mais onerosas, Costa (2004 apud BACK,1996, p. 15).

Costa (2004 apud KANNER,1997, p. 15) relata que um teste automatizado pode ser entre 3 à 10 vezes mais custoso para ser criado, validado e documentado, do que testes manuais. Por isso, deve-se observar os testes que poderão ter diversas execuções.

O teste automatizado é mais trabalhoso e caro nas primeiras execuções, após tende a ser mais vantajoso, que o teste manual.

Para os testes automatizados serem eficientes e de fácil manutenção, primeiramente é necessário montar uma estrutura de automação, definir uma estratégia de teste (que identifica o que será utilizado como ambiente de testes) e definir quais serão as etapas para criação dos testes.

2.7 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os conceitos básicos que serão utilizados para o desenvolvimento da ferramenta. Os testes de software são essenciais para a garantia de qualidade do desenvolvimento de software, porém devem ser estudados e analisados os métodos que melhor se encaixam com cada processo em uma organização.

Os testes automatizados necessitam de um planejamento maior quanto a organização e implementação. As empresas precisam ter já definido métodos de testes, para conseguir planejar os resultados esperados da automatização.

No capítulo 3 será descrito a modelagem da ferramenta proposta, baseada nas informações sobre testes já apresentadas.

3 FERRAMENTA ALLINONE TEST AUTOMATION

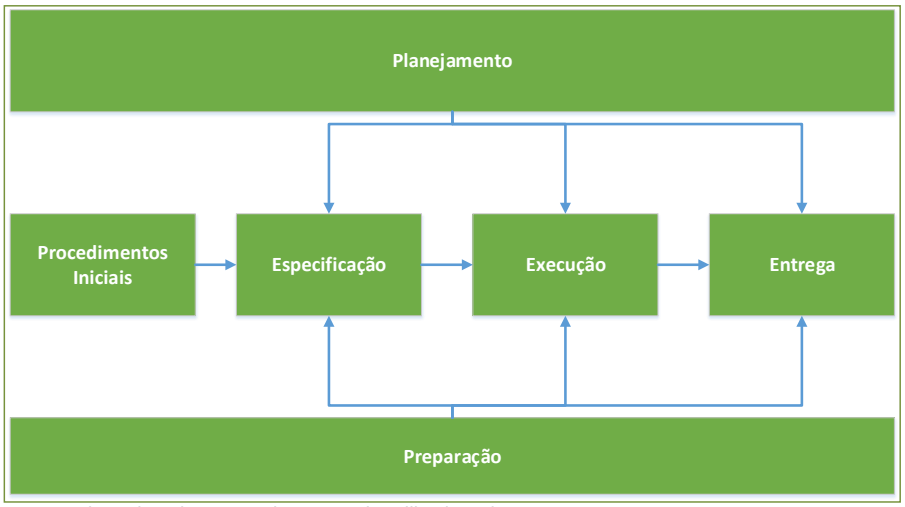
Neste capítulo serão discutidos aspectos referentes à ferramenta: arquitetura, modelagem e tecnologias usadas.

3.1 FLUXOS DE AUTOMAÇÃO DE TESTE

De acordo com Rios e Moreira Filho (2013), um processo de teste de software deve aderir-se ao processo de desenvolvimento. O processo de teste precisa ter uma metodologia tão eficiente quanto a utilizada para o processo de desenvolvimento de software.

Na concepção de Rios e Moreira Filho (2013), um processo de teste automatizado possui 6 fases, conforme a figura 4.

Figura 4 - Processo de Teste



Fonte: Adaptado pelo Autor, Rios e Moreira Filho (2013).

Os principais requisitos de cada fase, baseado em Rios e Moreira Filho (2013) são:

a) Procedimento Iniciais:

- Objetivo do projeto de teste.
- Equipe envolvida e suas responsabilidades.
- Plano do trabalho.

- Avaliação de riscos e estimativas.
- Itens considerados relevantes.

b) Planejamento:
- Elaboração e revisão, da estratégia de testes e do plano de teste.

c) Preparação:
- Formulação do ambiente de teste, como: Equipamentos físicos e equipe envolvida.

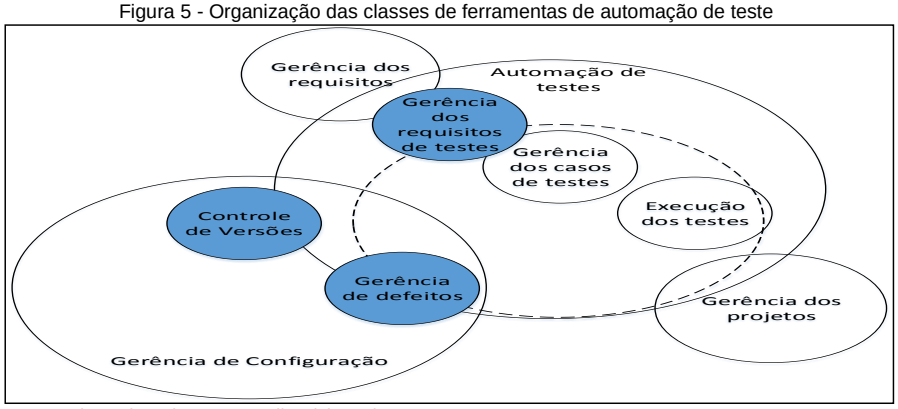
d) Especificação:
- Desenvolvimento dos casos de testes, *scripts* e roteiros.

e) Execução:
- Execução dos testes elaborados, registro dos resultados obtidos. Avaliação dos testes.

f) Entrega:
- Finalização do processo, implantação do software.

Molinari (2014) esboça a organização de ferramentas para automação de testes baseada nas visões das entidades, QAI (Quality Assurance Institute) e a ISTQB (International Software Testing Qualifications Board), consideradas umas das principais em certificações em testes no mundo. Ele considera que a automação de teste incorpora várias disciplinas, essas semelhantes com a concepção de Rios e Moreira Filho (20013), que são divididas em classes. A figura 5 esboça o ideal das classes para Molinari (2014).

Figura 5 - Organização das classes de ferramentas de automação de teste



Fonte: Adaptado pelo autor, Molinari (2014).

Pode-se observar na organização das classes da figura 5 que a automação de testes é vinculada a todo o processo. Sendo assim, a gerência de requisitos de testes, gerências dos casos de testes e execução dos testes são de âmbito da automação de testes. A estrutura de automação tem ligação com as demais gerências integrando-as aos testes, porém não faz parte diretamente dessas gerências, conforme Molinari (2014).

As gerências que não fazem parte integralmente da automação, respeitam o ciclo geral de desenvolvimento de software, mostrando assim a forte integração com os testes.

A arquitetura de automação, segundo Costa (2004), deve contemplar 6 funcionalidades, com o propósito de ser ágil e correto. Essas funções estão esboçadas na tabela 8, onde também é definido quais os artefatos produzidos pelas funções.

Tabela 7 - Funcionalidades de uma arquitetura de automação

Funcionalidade	Descrição	Artefatos
Planejamento do teste	Determinar quais são os testes mais importantes.	Dados estimados do tempo demandado para processo de teste. Importação dos testes e entendimento do cenário.
Desenvolvimento do teste	Especificação e implementação da configuração do teste.	Caso de teste, geração de dados para teste, suítes de testes.
Execução do teste	Execução do código e gravação de artefatos.	Saídas dos testes, registro da execução e status do teste.
Análise de falhas	Documentação e verificação do comportamento de execução dos testes. Comparar o resultado encontrado pelo esperado.	Registro se falhou ou não.
Medições dos testes	Medição de cobertura dos testes.	Comparação do estimado com realizado.
Gerenciamento das configurações dos artefatos de teste	Preservação dos dados relativos a configuração dos testes e execuções anteriores em um repositório.	Possibilidade de rastreamento de testes anteriores.

Fonte: Costa (2004), adaptado pelo autor.

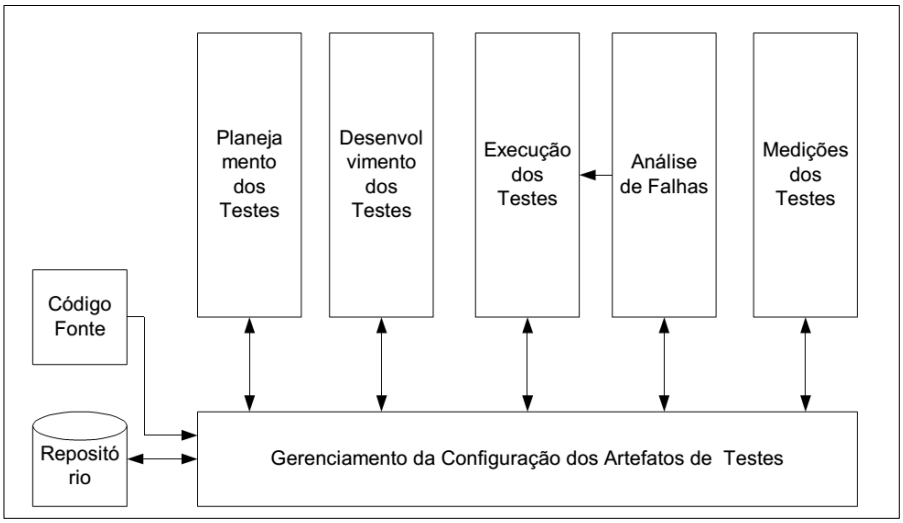
Embasado nas três arquiteturas explanadas anteriormente, pode-se definir uma estrutura que se adeque a metodologia de desenvolvimento do protótipo para este trabalho. Na seção seguinte, será apresentada a estrutura de automação de testes para o protótipo, possibilitando assim elencar os requisitos e dar subsídio para a modelagem.

3.2 ESTRUTURA PARA AUTOMAÇÃO DE TESTES NO PROTÓTIPO

A automação de testes precisa contemplar o requisito de reusabilidade dos artefatos de testes, por exemplo a facilidade de reuso de *scripts*, a rastreabilidade entre os objetos como requisitos, casos de testes, execução e falhas. Para tal é necessário que se construa uma arquitetura eficiente, afirma Costa (2004).

A figura 6 ilustra a arquitetura conforme as funcionalidades descritas na tabela 7 – Funcionalidades de uma arquitetura de automação.

Figura 6 - Arquitetura de automação de testes de software



Fonte: Costa (2004 apud VOGEL, 1993, p. 35).

Pode-se observar na figura 6 que as funcionalidades são independentes ambas consultam ao repositório, ou seja, cada funcionalidade pode funcionar sem depender de outras. Apenas a execução dos testes e análise de falhas podem trabalhar em conjunto, pois serão gravados os resultados com base na execução e análise dos testes.

3.2.1 **Planejamento dos testes**

De modo geral, a atividade de planejamento dos testes é tratada como sendo sem relacionamento com as demais atividades da automação de testes. Costa (2004) afirma que, embora não tenha relacionamento, é necessário que o planejamento dos testes receba informações do projeto de desenvolvimento, onde os testes estão envolvidos.

Os dados provenientes do projeto de desenvolvimento terão a finalidade de estimar o tempo a ser despendido para a criação de testes. Sendo assim, será possível identificar os projetos, nos quais extrapolaram o tempo estimado, levantando possíveis causas e soluções para que as estimativas futuras sejam mais precisas.

Para sintetizar, um projeto de teste sempre deverá ter quantidade de horas para sua execução. Se caso o projeto de teste for planejado para utilizar 10 horas e contabilizando sua implementação esse valor for superior, então no processo existiu algum problema que pode ser corrigido em novos projetos, melhorando sempre a precisão de estimativas.

Além de estimar o tempo, com base em Rios e Moreira Filho (2013), no planejamento será necessário definir as prioridades dos testes e para indicadores do processo, a data de início da elaboração do plano e fim, como também a quantidade de horas utilizadas.

3.2.2 **Desenvolvimento dos testes**

Para Costa (2004), o desenvolvimento de artefatos de teste pode se adequare a três requisitos: aos relacionados em requisitos de software e teste, os básicos e os que serão utilizados como oráculos.

Caso os testes forem desenvolvidos baseados em requisitos de software, Costa (2004) aborda que um dos objetivos é o relacionamento entre o requisito de software e o teste.

Costa (2004) elenca outros objetivos para o desenvolvimento de teste relacionados a requisitos, conforme é demostrado abaixo:

- O gerenciamento de requisitos de software e o de teste, sejam feitos por uma mesma ferramenta ou que possua uma forte integração.

- Cada requisito deve ser definido com um grau de risco, para assim tratá-lo com maior ênfase.
- A ferramenta de automação de testes, deve rastrear os requisitos de software e os requisitos de teste além de seus casos de teste.

Acerca dos casos de teste, a ferramenta também deve prover mecanismos, para:

- A criação de casos de teste, também chamados artefatos básicos.
- Suítes de teste ou cenários de testes.
- Roteiros de testes.
- Critérios de teste.
- E a geração de dados de teste.

Os casos de teste devem ser agrupados por roteiro, ou seja, aquele roteiro será capaz de executar vários casos de teste e um suíte de teste agrupará roteiros.

Os dados de entrada podem ser gerados manualmente ou automaticamente, assim sendo a ferramenta de automação proverá mecanismos para tal fim.

Ainda no desenvolvimento deve-se gerar os oráculos de testes. Oráculos de teste, segundo Costa (2004), são responsáveis por determinar se um teste falhou ou foi executado com sucesso. Para isso esses oráculos podem ser desenvolvidos como um Oráculo de amostra resolvida, que segundo Costa (2004, apud Binder, 2000, p. 38), os resultados são incorporados manualmente ao caso de teste.

Para este trabalho não será utilizado o conceito de oráculos. As comparações serão analisadas pelo módulo de análise de testes, que será explicado a seguir.

3.2.3 **Execução dos testes**

De acordo com Costa (2004), a execução do teste deve ser feita por um Driver[7] de testes, que é responsável por arranjar a execução do teste. Cabe ao Driver:

- Chamar o módulo para preparação dos dados para os casos de testes.
- Recuperar os casos de teste de um suíte.

[7] Driver: Inicialização de teste automatizados.

- Após a execução, chamar o módulo de análise de falhas.
- Chamar módulo de limpeza.

Os scripts também poderão ser gerados a partir de um passo a passo. Esses baseados em um caso de teste, ou seja, na definição de um caso de teste, serão cadastrados os dados de entrada e os registros de análise dos testes, as saídas, que serão introduzidos como parâmetros em um script de teste automático.

É desejável que o protótipo tenha a possibilidade de escolher entre casos de testes, roteiros e suítes de teste. Sendo também passível de seleção de somente os testes que falharam, para a possibilidade de sua reexecução.

A ferramenta, conforme Costa (2004), deverá prover mecanismos para facilitar a depuração, como guardar registros de execução dos testes. Assim será possível identificar o progresso de um teste.

A fase de execução de testes está ligada a análise de falhas, para tanto a execução de testes deve registrar o status de um teste, se passou, falhou ou não executou. Portanto a execução de testes deve chamar o módulo de análises de falhas.

3.2.4 **Análise de falhas**

Conforme Costa (2004), os oráculos desenvolvidos na fase de desenvolvimento de testes, irão atuar como comparadores de resultados, com o intuito de determinar se o teste falhou ou passou.

Se o resultado de retorno, não for o dado esperado, existirá erro no teste. Esse deverá ser registrado, determinando o que era esperado, e o que fora encontrado.

O módulo de análise de falhas deverá ser integrado como o módulo de medições de falhas. Baseado em Rios e Moreira Filho (2013), ao final da análise de falhas, o protótipo deverá ser capaz de:

- Gerar relatórios de defeitos encontrados.
- Possibilitar a análise de casos de testes, com o objetivo de identificar seu andamento perante os testes já realizados.

3.2.5 Medições do teste

Conforme Rios e Moreira Filho (2014), primeiramente os indicadores devem ser definidos, para um projeto ou um processo como um todo. Após definidos, é necessário estabelecer uma regra para cálculo do mesmo, relevando os dados estimados com os encontrados.

Costa (2004) cita que na automação do teste, a ferramenta deve viabilizar a coleta de dados processados pelo módulo de análise de falhas. Esse processo tem o objetivo de alimentar indicadores, para que possibilitem melhorar a qualidade do software, do processo de desenvolvimento e do processo de teste.

Dos indicadores referenciados por Costa (2004), serão utilizados os abaixo:

- Tempo para conserto da falha após a execução.
- Falhas por programas e responsáveis.
- Tempo utilizado para o plano de teste.
- Relação final do tempo estimado x tempo efetivo.

Os indicadores de planejamento de teste também serão medidos, por exemplo, o quanto foi esperado para execução do processo de teste na medida de tempo e o quanto foi realmente utilizado.

3.2.6 Gerenciamento de configuração dos artefatos de teste

Dentre os principais requisitos para o gerenciamento de configuração de artefatos, Costa (2004) releva a reusabilidade.

Conforme Costa (2004) a reusabilidade é a capacidade de armazenar, recuperar e reutilizar os artefatos de teste, sendo fundamental para a automação de teste. Entre os artefatos de teste, o que tem maior índice de reuso, são os scripts de teste.

O gerenciamento de configuração tem a responsabilidade de identificar quais os suítes de teste que possuem determinado artefato vinculado, a fim de possibilitar a reutilização de testes já desenvolvidos para um artefato.

3.3 RECURSOS UTILIZADOS NO DESENVOLVIMENTO DO PROTÓTIPO

Esta seção tem o objetivo de referenciar as tecnologias utilizadas para o desenvolvimento do protótipo do software, *AllInOne Test Automation.*

Para o desenvolvimento da aplicação proposta será utilizada a linguagem de programação C# (Sharp)[8].

Como ambiente integrado de desenvolvimento (IDE) será utilizado o programa proprietário da Microsoft, Visual Studio 2013 em sua versão Express[9].

No desenvolvimento deste projeto será utilizado um sistema de gerenciamento de banco de dados (SGBD). Este SGBD será o POSTGRES devido a sua simplicidade de uso e de arquitetura relacional[10].

Na modelagem UML deste projeto será utilizado o programa Astah Community. Com este é possível gerar os diagramas de classe e caso de uso, que serão utilizados como base para este projeto[11].

Devido ao protótipo ter como base a geração de testes automatizados para a linguagem PL/SQL, foi escolhido o framework utPLSQL para gerenciar os testes diretamente nos SGBD Oracle.

O utPLSQL oferece um conjunto de pacotes que podem ser utilizados para testes de unidades de forma eficiente e fácil, assim definindo um processo padrão para a instalação, configuração e utilização do framework.

Ainda o utPlSQL é independente de plataforma sendo instalado por scripts SQL, tendo a necessidade de configuração manual.

No momento da elaboração deste trabalho, a versão disponível do utPLSQL é a 2.2[12].

3.4 MODELAGEM DO PROTÓTIPO

Embasado na arquitetura de Costa (2004) para a automação de testes, os requisitos foram modelados. Foi utilizado o diagrama de classes para a compreensão dos objetos do protótipo, de diagramas de caso de uso para

[8] http://msdn.microsoft.com/en-us/vstudio/aa718325.aspx
[9] https://app.vssps.visualstudio.com
[10] http://www.postgresql.org/
[11] http://astah.net/editions/community.
[12] http://sourceforge.net/projects/utplsql/files/

especificar a interação de cada requisito, de diagramas de projeto com o objetivo de demostrar os objetos com seus atributos e métodos principais, prototipagem de tela e diagrama de sequência para os casos de uso médios e complexos.

Para a demonstração da arquitetura de software foram utilizados diagramas de componentes e diagrama de classe de projeto em camadas.

Na moldagem da arquitetura foi utilizado o modelo MVVM (Model-View-View-Model)[13].

3.4.1 **Diagrama de Classe do sistema**

A figura 7 representa o diagrama de classe de sistema no qual como classe principal tem-se a de Projeto.

A classe de projeto referencia os dados de um projeto de desenvolvimento, tendo por função indicar o status do projeto de testes e quantas horas estimadas para testes, valendo-se de um percentual sobre o projeto de desenvolvimento do software.

A classe de requisitos identifica quais os requisitos do projeto serão testados possuindo a descrição da funcionalidade requisitada.

Já a classe de casos de teste, especifica os testes que serão desenvolvidos para cada requisito, necessitando das etapas de testes para identificar os parâmetros de entrada e resultados esperados. Além disso, o caso de teste tem a funcionalidade de Modo de teste, o qual identifica se o teste será executado em um plano de execução automaticamente, caso esteja cadastrado a um plano de execução.

Cada etapa de teste pode possuir vários parâmetros de entrada e de saída. Para cada etapa existe um *script* de teste que irá validar a unidade, utilizando todos os parâmetros de entrada e saída para a etapa.

O *script* de teste gera o código para ser implementado no *framework* utPLSQL, criando um procedimento armazenado no SGBD Oracle.

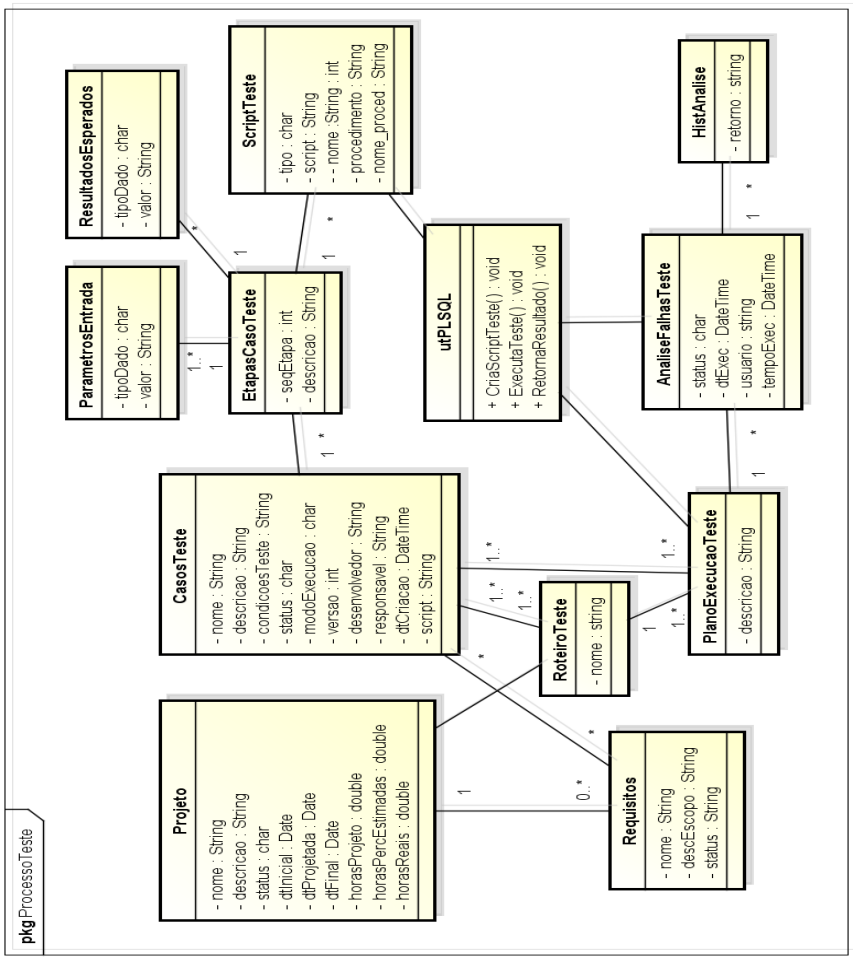
A classe roteiro de teste agrupa casos de testes e pode ser vinculada a vários planos de execução para cada projeto.

[13] http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx.

O plano de execução tem o objetivo de exercitar todos os scripts dos casos de testes, esses relacionados com um roteiro ou não. Ainda, executa o *framework* utPLSQ para execução dos testes em PL/SQL.

Após o término da execução é utilizado a classe de análise de falhas, a qual grava os dados da execução como também realiza integração com a classe utlPLSQL, retornado os resultados armazenados no SGBD Oracle.

Figura 7 - Diagrama de Classes de Sistema



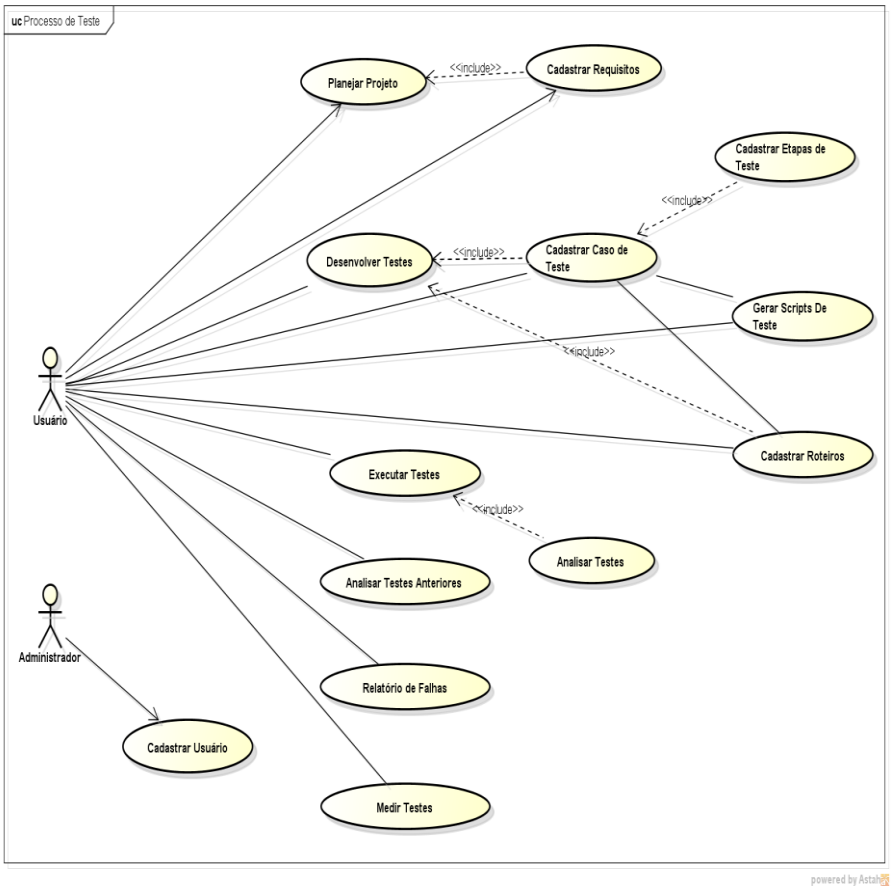
Fonte: O autor.

3.4.2 **Diagrama de Casos de Uso**

Para a modelagem na figura 8 é apresentado um diagrama de caso de uso de modo geral. Esse tem a função de demonstrar as funções que o protótipo deverá atender. Após o diagrama de casos de uso é representado o diagrama de classe de sistema, o qual dá a perspectiva da modelagem baseado nos casos de uso.

Cada caso de uso será especificado e detalhado por: Descrição de caso de uso, protótipo de tela e diagrama de sequência nos processos que envolvem a interação do sistema com o usuário como: Cadastrar Caso de Teste, Cadastrar Etapas de Teste, Gerar Script de Teste, Executar Script de Teste e Analisar Falhas.

Figura 8 –Diagrama de caso de uso



Fonte: O autor.

3.4.3 Arquitetura

A figura 9 apresenta a arquitetura de software do protótipo. Onde para todo o desenvolvimento que envolver diferentes interfaces, essa estrutura estará presente, sendo definida em quatro camadas visando facilitar a manutenção.

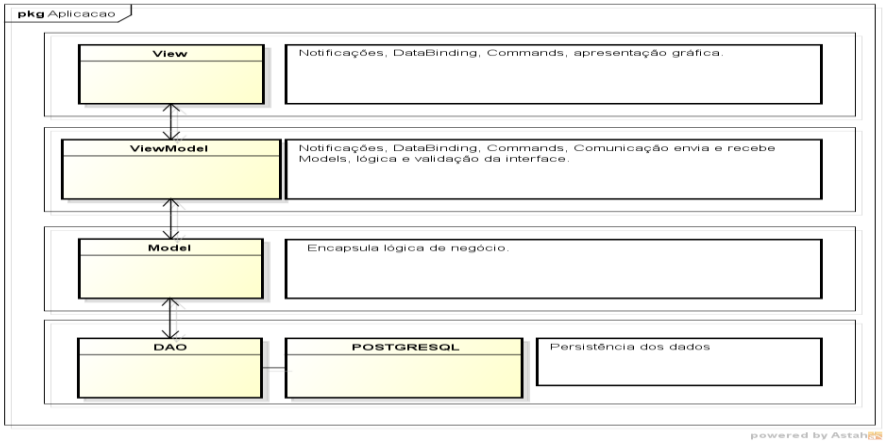
A View representa a interface apresentada ao usuário, é onde ocorrerá a interação. Ainda, interage diretamente com a camada ViewModel que realiza o processamento da lógica dos dados provenientes dos comandos da interface. A interação se dá por meio da interface de DataBindings que disponibiliza os comandos disponíveis da camada ViewModel à camada View.

A camada ViewModel tem uma função vital na arquitetura da aplicação. Ela realiza o processamento dos dados da interação do usuário validando e notificando a interface da mudança de dados. A ViewModel comunica-se diretamente com a camada de negócio permitindo a inicialização de objetos e a notificação dos dados.

A próxima camada, Model, trata a lógica de negócio que gerencia todo o processamento, como: cálculos, chamada à persistência de dados, manipulação de erros e notificação para a camada ViewModel dos dados.

Na quarta camada é tratada a persistência de dados, onde é realizada a comunicação com o banco de dados. Essa camada é implementada através de interface pela camada Model. Nessa camada é gerenciada todas as transações com o SGBD PostgreSQL.

Figura 9 - Diagrama de Arquitetura em camadas



Fonte: O autor.

Comentado [GB1]: Arquitetura Lógica da aplicação.

3.4.4 **Detalhamento dos Requisitos**

Será apresentado o detalhamento dos requisitos modelados em casos de uso, protótipos de tela (para exemplificar os requisitos) e diagrama de sequência para os casos de uso de média e grande complexidade.

a) Planejar Projeto

A tabela 8 descreve o caso de uso Planejar Projeto, indicando o que ele deverá realizar.

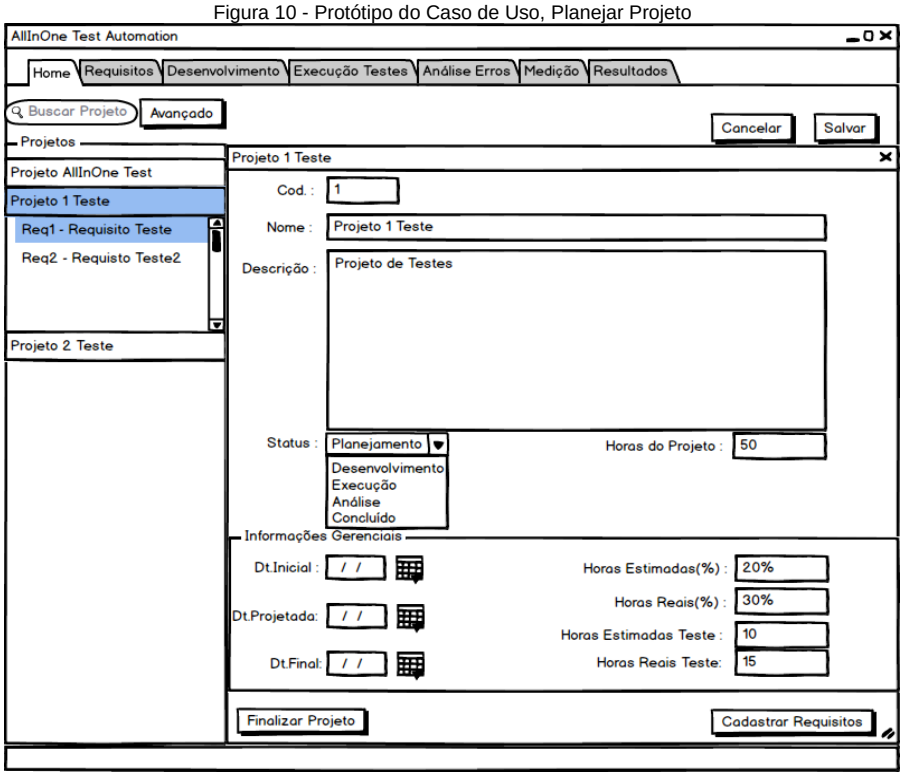
Tabela 8 - Descrição Caso de Uso, Planejar Projeto

Caso de Uso: Planejar Projeto	
Objetivo:	Permitir cadastramento de projetos e manutenção dos mesmos.
Descrição básica:	* O usuário cadastra um projeto, sendo este considerado como agrupador no conceito de suítes de testes. (Identificador, Código, Nome, Descrição Tempo Estimado, Tempo Real,Data Inicial, Data Projetada, Data Final, Status do projeto). Permitir o cadastro e manutenção de projetos. * Além disso, o sistema deverá permitir o cadastro de informações referentes ao tempo total de desenvolvimento do projeto e o percentual estimado para o processo de teste. * Os projetos poderão ser filtrados para o fim de rastreabilidade, que é um requisito do processo de gerenciamento de artefatos. * Após o Cadastro de projetos é possível cadastrar um ou vários requisitos.
Atores:	Usuário.
Pré-Condição	
Pós-Condição	Cadastrar Requisitos.
Fluxo Principal	1. O usuário cadastra os dados do projeto. 2. O usuário clica em cadastrar requisitos. 3. O sistema abre uma nova tela, para cadastro de requisitos.
Fluxo Alternativo A	1.O usuário finaliza projeto. 2. O sistema preenche os dados para finalização do projeto.

Fonte: O autor.

Na figura 10 pode-se observar a tela que deverá ser gerada pelo caso de uso planejar projeto. Nesta tela serão informados os dados básicos para um projeto de teste, definido nos requisitos.

Figura 10 - Protótipo do Caso de Uso, Planejar Projeto



Fonte: O autor.

A tabela 9 demostra a descrição do caso de uso, Cadastrar Requisitos.

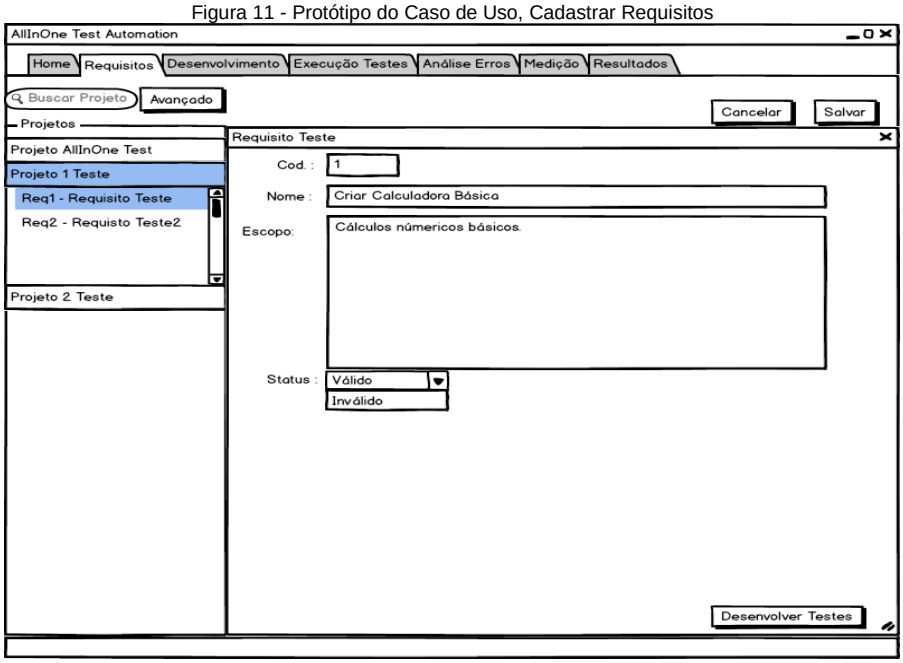
Tabela 9 - Descrição do Caso de Uso, Cadastrar Requisitos

Caso de Uso: Cadastrar Requisitos	
Objetivo:	Permitir cadastramento de requisitos por projetos
Descrição básica:	O usuário poderá cadastrar os requisitos do projeto a serem testados, especificando. Código, nome, escopo e status do requisito, se válido ou inválido. Caso for inválido, não deve ser testado.
Atores:	Usuário.
Pré-Condição	1.Efetuar login no sistema. 2.Ter cadastrado um projeto.
Pós-Condição	Possibilidade de dar continuidade no processo de desenvolvimento.
Fluxo Principal	1. O usuário informa os dados. 2. Clica em desenvolver testes. 3. O sistema aciona o caso de uso, Desenvolver Teste.

Fonte: O Autor.

A tela de cadastro de requisitos é demostrada na figura 11, ela é embasada nos requisitos demonstrados no caso de testes Cadastrar Requisitos.

Figura 11 - Protótipo do Caso de Uso, Cadastrar Requisitos



Fonte: O Autor.

b) Desenvolver Testes

Neste tópico serão apresentadas as definições para atender ao requisito de desenvolver testes. Este requisito gerou os casos de uso: Cadastrar Casos de Teste, Cadastrar Etapas de Teste e Gerar Script de Teste.

A tabela 10 descreve o caso de uso desenvolver teste. Delimitando seus requisitos e funções.

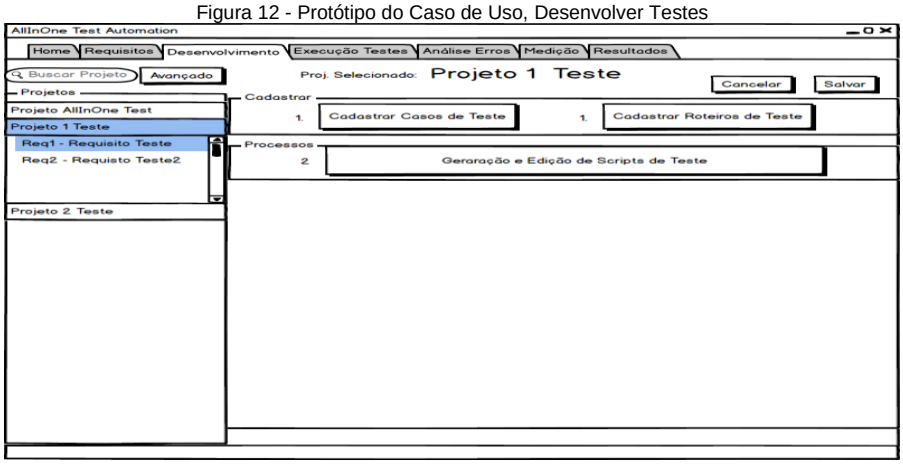
Tabela 10 - Descrição do Caso de Uso, Desenvolver Testes

Caso de Uso: Desenvolver Testes	
Objetivo:	Permitir cadastrar testes vinculados ao requisito selecionado.
Descrição básica:	O usuário poderá navegar através de botões para as telas de cadastro de roteiros, cadastros de casos de teste e geração, edição de scripts de teste.
Atores:	Usuário.

Caso de Uso: Desenvolver Testes	
Pré-Condição	1.Cadastrar Projetos
Pós-Condição	Possibilidade de navegar entre: 1.Cadastro de Casos de teste. 2.Cadastro de Roteiros de Teste. 3.Geração de Scripts de Teste.
Fluxo Principal	1.O usuário clica em Cadastrar Casos de teste. 2.O sistema acionar o caso de uso Cadastrar Casos de Teste.
Fluxo Alternativo A	1.O usuário clica em Cadastrar Roteiro de Teste 2.O sistema acionar o caso de uso Cadastrar Roteiro de Teste.
Fluxo Alternativo B	1.O usuário clica em Geração e Edição de Script de Teste. 2.O sistema acionar o caso de uso Gerar Script de Teste.

Fonte: O autor.

Figura 12 - Protótipo do Caso de Uso, Desenvolver Testes



Fonte: O Autor

Na figura 12 é demonstrado a tela do caso de uso Desenvolver Testes, desenhada a partir das funcionalidades especificadas para o caso de uso.

A tabela 11 apresenta o a descrição do caso de uso, cadastrar casos de teste. A figura 13 demonstra a tela, com base na descrição do caso de uso Cadastrar Caso de Teste.

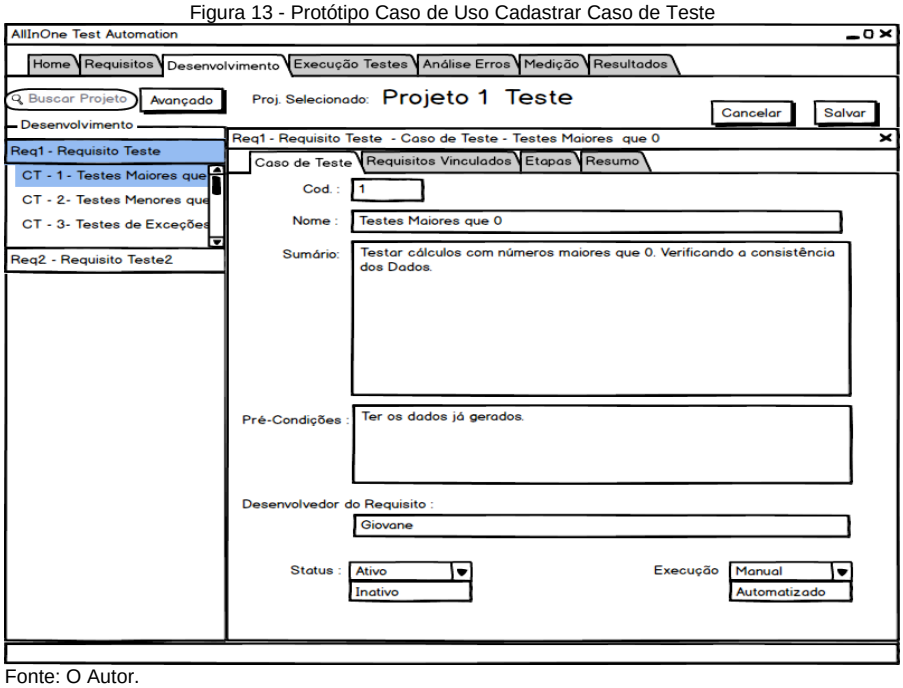
Tabela 11 - Descrição Caso de Uso, Cadastrar Casos de Teste

Caso de Uso: Cadastrar Casos de Teste	
Objetivo:	Permitir cadastrar casos de teste.
Descrição básica:	*O usuário poderá cadastrar casos de testes contendo os passos para execução dos testes, uma grid com informações a serem testadas, definindo parâmetros de entrada e resultados esperados.

Caso de Uso: Cadastrar Casos de Teste	
	*Ainda deverá ser possível vincular um caso de teste a mais de um requisito pelo mesmo cadastro. O usuário poderá informar os atributos: Nome, Descrição Sumarizada, Pré-Condições, se ativo ou inativo, modo de execução, Requisitos Vinculados.
Atores:	Usuário.
Pré-Condição	Atender ao caso de uso, Planejar Testes.
Pós-Condição	1.Possibilidade de cadastro de roteiro. 2.Geração de Scripts de teste.
Fluxo Principal	1.O usuário informa os dados. 2. O usuário clica na aba Requisitos Vinculados. 3. O sistema abre a tela de requisitos vinculados.
Fluxo Alternativo A	1. O usuário clica na aba Etapas. 2. O caso de uso Cadastrar Etapas é acionado.
Fluxo Alternativo B	1. O usuário clica na aba Resumos 2. O sistema abre a tela de requisitos vinculados.

Fonte: O Autor.

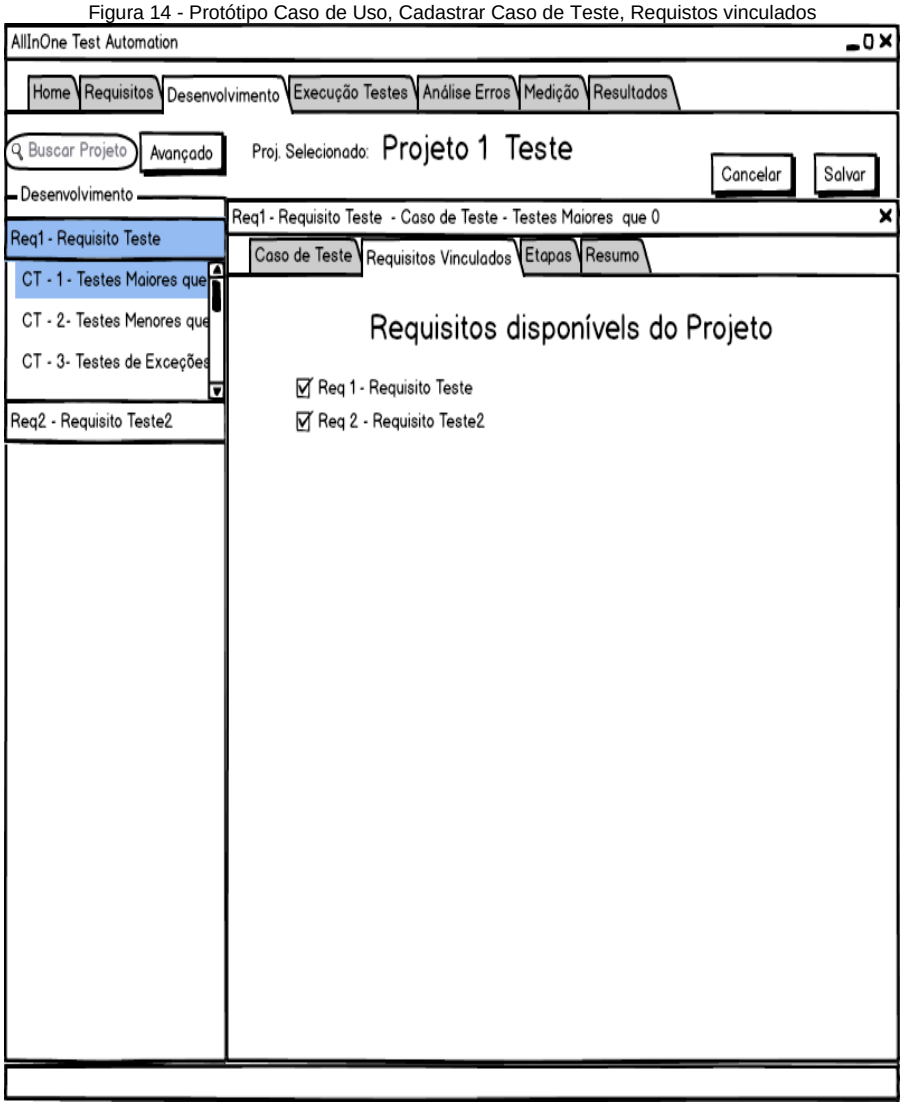
Figura 13 - Protótipo Caso de Uso Cadastrar Caso de Teste



Fonte: O Autor.

Na figura 14 é exposto a tela para vínculos de requisitos, os requisitos foram evidenciados na tabela 11.

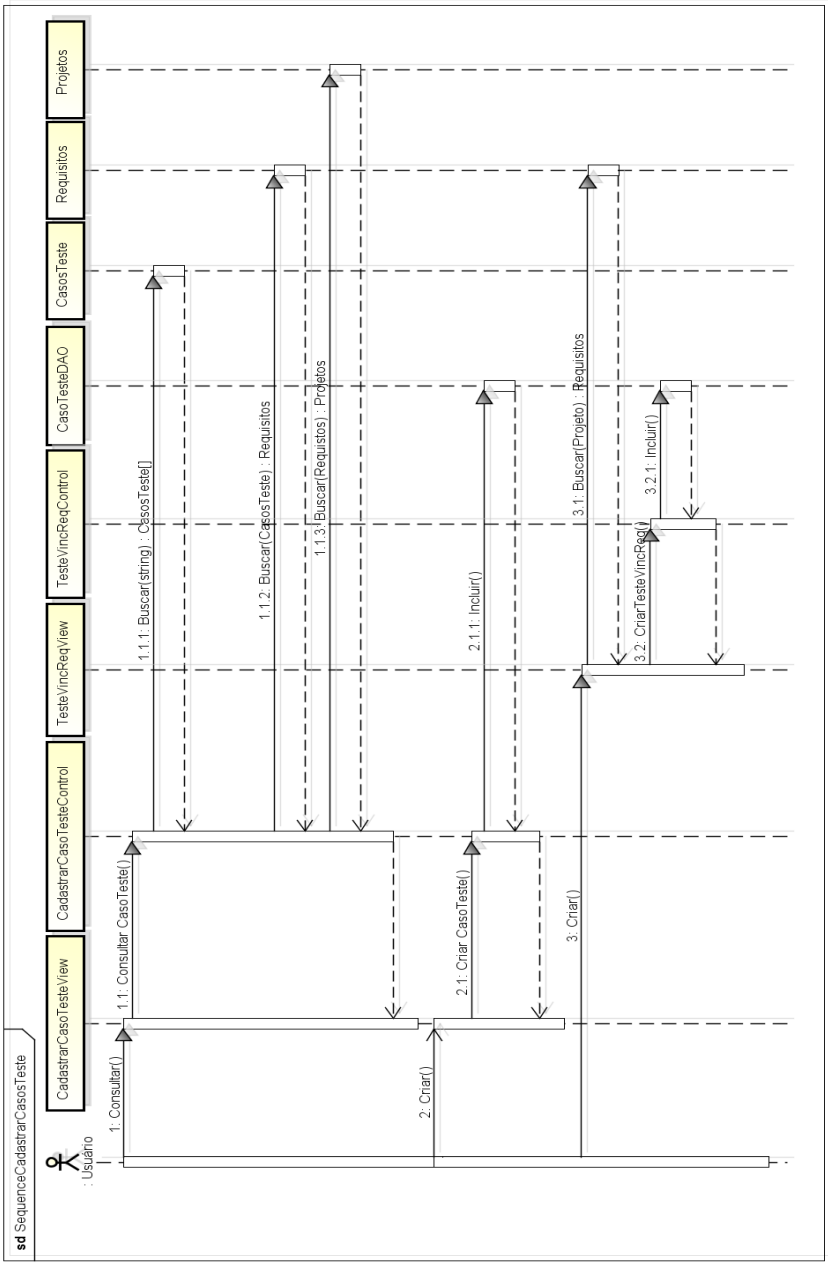
Figura 14 - Protótipo Caso de Uso, Cadastrar Caso de Teste, Requistos vinculados



Fonte: O Autor.

A figura 15 apresenta o diagrama de sequência modelado para demonstrar o fluxo do digrama de classe e do diagrama de caso de uso Cadastrar Caso de Teste.

Figura 15 - Diagrama de Sequência Cadastrar Caso de Teste



Fonte: O autor.

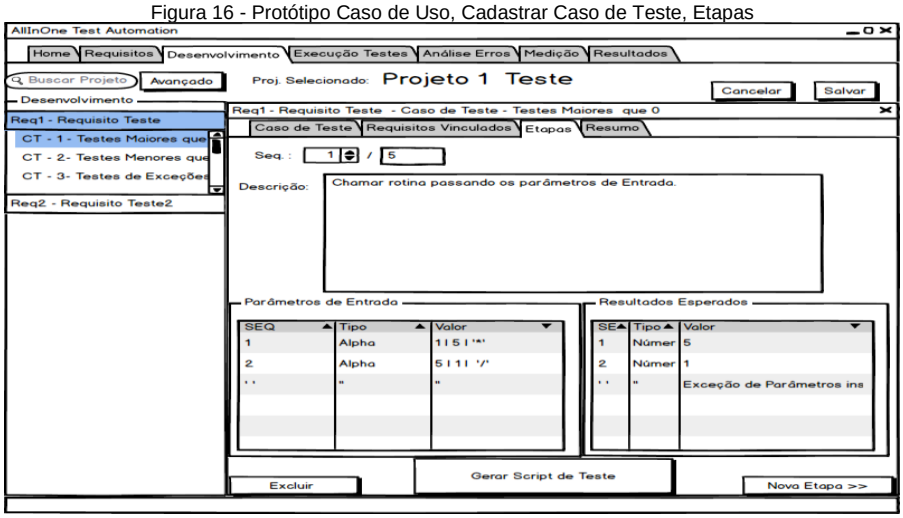
A tabela 12, descreve o caso de uso Cadastrar Etapas de Teste, indicando suas funcionalidades.

Tabela 12 - Descrição Caso de Uso, Cadastrar Etapas de Teste

Caso de Uso: Cadastrar Etapas de Teste	
Objetivo:	Permitir cadastrar etapas para casos de teste.
41 zDescrição básica:	*Etapas do teste, detalhes da etapa de teste, resultados esperados, parâmetros de entrada de dados, responsável e desenvolvedor. *Os scripts de testes poderão ser gerados a partir dos parâmetros de entrada e os resultados esperados. *Deverá ser possível, informar novos parâmetros para cada script de teste. *Resumo do cadastro de caso de teste.
Atores:	Usuário.
Pré-Condição	Cadastrar caso de teste.
Pós-Condição	1. Possibilidade de geração de scripts de teste.
Fluxo Principal	1. O usuário informa os dados. 2. O usuário informa os parâmetros. 2. O usuário clica em Gerar Script de Teste. 3. O caso de uso Gerar Script de Teste é acionado
Fluxo Alternativo A	1. O usuário clica em Nova Etapa. 2. O sistema gera uma etapa com sequência posterior.
Fluxo Alternativo B	1. O usuário clica em Excluir. 2. O sistema exclui a etapa e retorna para etapa anterior.

Fonte: O Autor.

Figura 16 - Protótipo Caso de Uso, Cadastrar Caso de Teste, Etapas



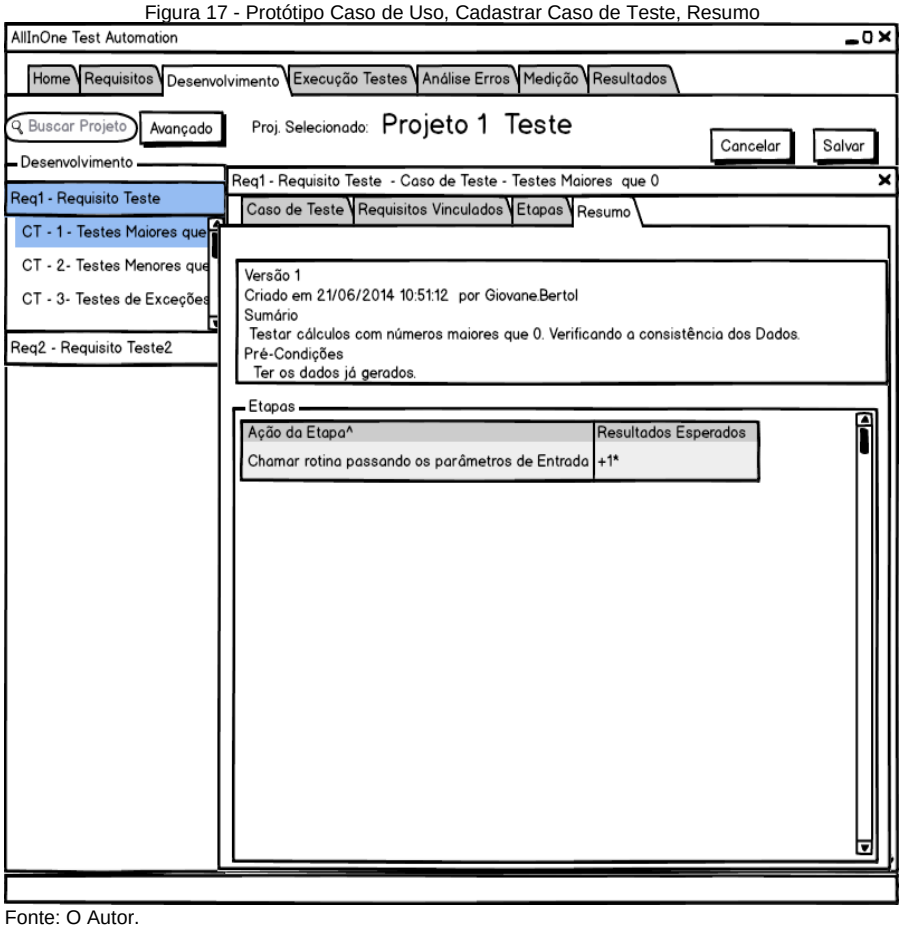
Fonte: O Autor.

A figura 16 exibiu a tela produzida com base nas funções descritas na tabela 12 - Descrição Caso de Uso, Cadastrar Etapas de Teste, sendo possível cadastrar as etapas de teste, os parâmetros de entrada e os resultados esperados para os parâmetros de entrada de mesma sequência.

A figura 17 demonstra o resumo do cadastro de caso de testes que tem a função de demostrar dados do caso de teste selecionado.

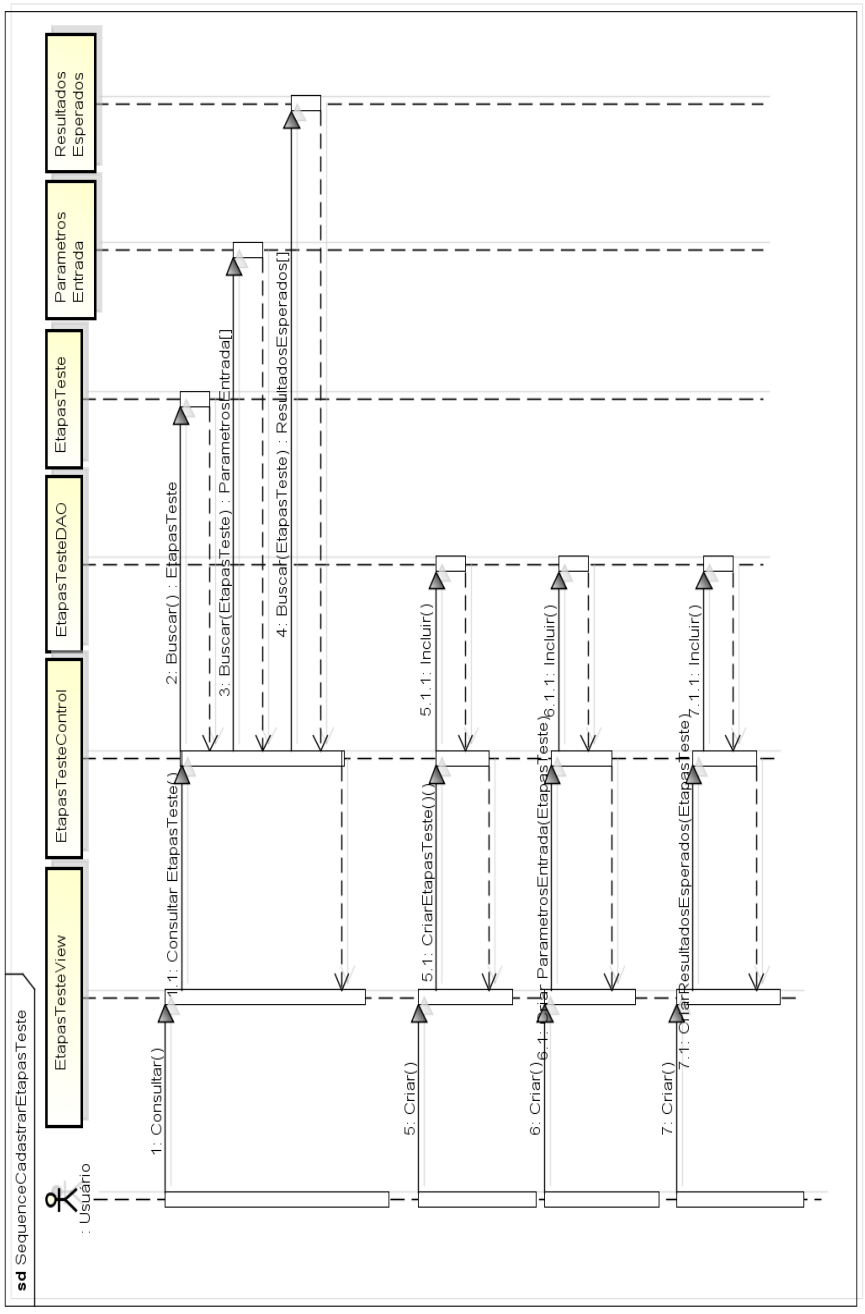
Na figura 18 é possível visualizar o Diagrama de sequência Cadastrar Etapas de Teste, é demonstrado a interação do diagrama das classes cadastrar etapas de teste.

Figura 17 - Protótipo Caso de Uso, Cadastrar Caso de Teste, Resumo



Fonte: O Autor.

Figura 18 - Diagrama de Sequência Cadastrar Etapas de Teste



Fonte: O autor.

A tabela 13 descreve o caso de uso Gerar Script de Teste.

Tabela 13 - Descrição Caso de Uso, Gerar Script De Teste

Caso de Uso: Gerar Scripts de Teste	
Objetivo:	Permitir a geração automatizada de scripts de teste. Manter scripts de teste.
Descrição básica:	• O usuário poderá gerar scripts de teste, baseados nos parâmetros de entrada e os resultados esperados de um caso de teste. • Um script poderá ter apenas uma etapa vinculada. • O script poderá ser copiado para ser utilizado como base em outra etapa. • Para geração de um script, como o protótipo irá gerar testes para a linguagem PL/SQL em banco de dados Oracle, será possível informar dados para um procedimento em específico. Ou seja parâmetros de entrada e resultados esperados no script de teste. Deverá ser informada o nome da rotina para a geração automatizada de testes, utilizando o framework utPLSQL.
Atores:	Usuário.
Pré-Condição	Ter cadastrado uma etapa de teste e seus parâmetros.
Pós-Condição	1. Possibilidade de execução de teste.
Fluxo Principal	1. Usuário informa os dados. 2. O usuário clica em gerar script. 3. O sistema gera o script com base nos dados informados.

Fonte: O Autor.

Na figura 19 é apresentada a tela responsável por gerar scripts de teste. Essa foi baseada na definição da descrição do caso de uso Gerar Scripts de Teste, apresentado na tabela 13. Cada etapa de teste deverá gerar um script baseado nos parâmetros de entrada e saídas esperados. O usuário irá informar os dados do cabeçalho, para que o programa consiga gerar o script automatizado na linguagem PL/SQL.

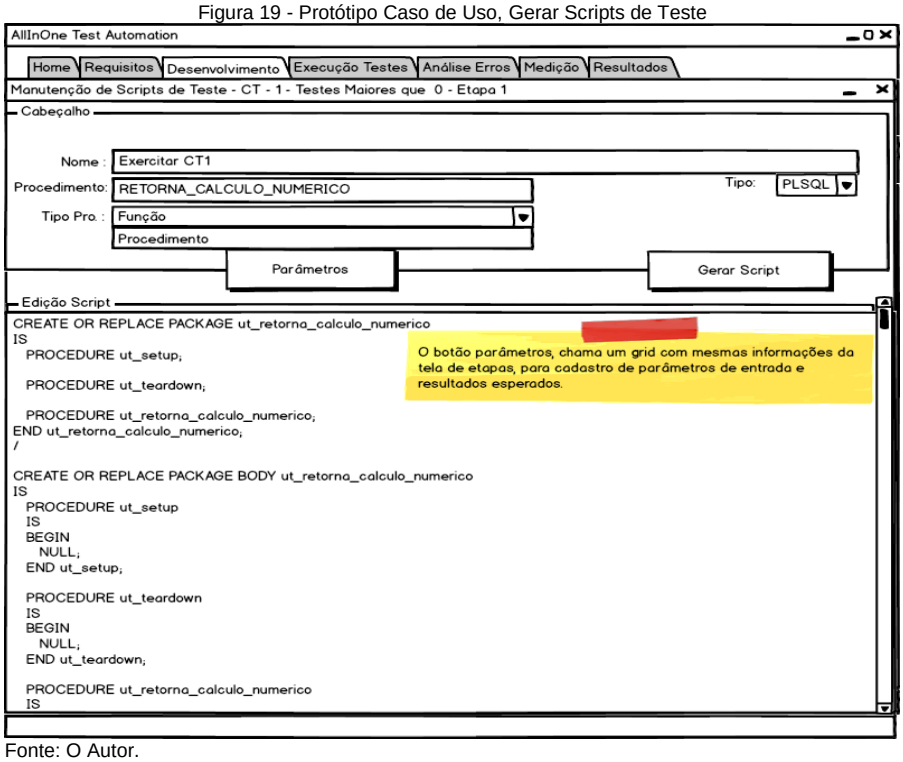
O script deve ser padronizado conforme estrutura suportada pelo framework utPLSQL, no anexo A é apresentado um exemplo dessa estrutura.

A figura 20 representa o diagrama de componentes do caso de uso Gerar Script de Teste. Esse diagrama demonstra que o componente da aplicação é dependente da implementação da interface ODBC do *driver* para conexão ao Oracle. Por sua vez, o componente utPLSQL é sujeito a implementação da interface transacional do Oracle para realizar as operações enviadas pela aplicação.

Comentado [GB2]: Essa parte é relacionada ao questionamento do Giovani.

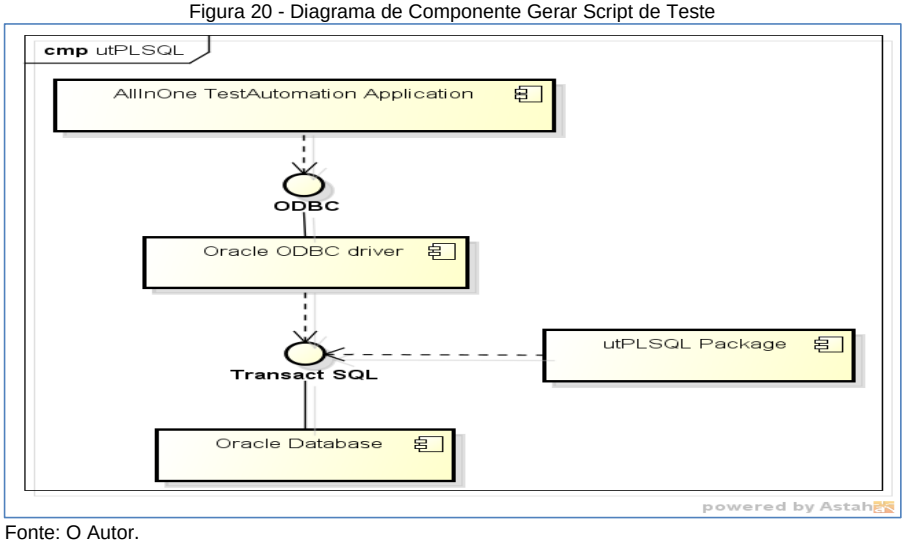
A figura 21 mostra o diagrama de sequência do caso de uso Gerar Scripts de Teste.

Figura 19 - Protótipo Caso de Uso, Gerar Scripts de Teste



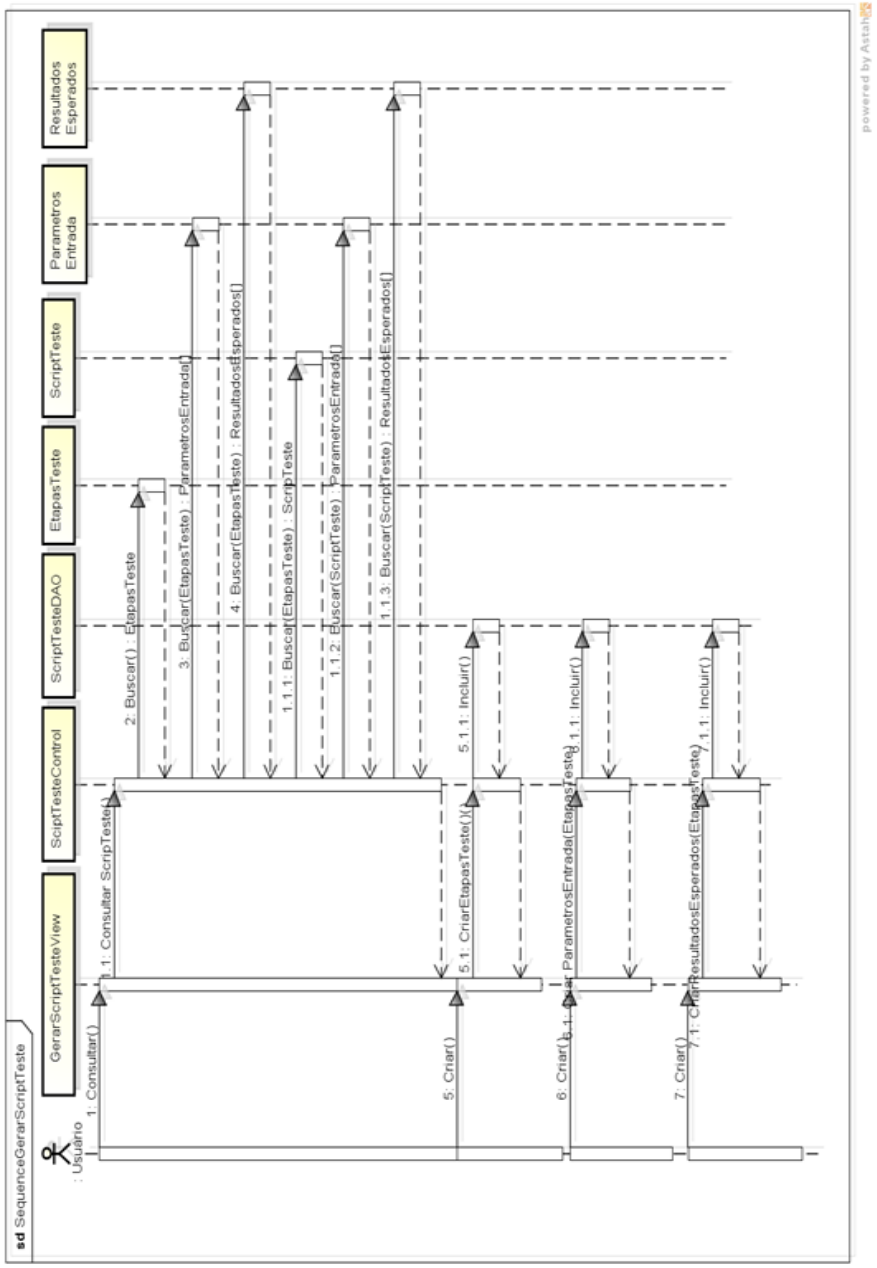
Fonte: O Autor.

Figura 20 - Diagrama de Componente Gerar Script de Teste



Fonte: O Autor.

Figura 21 - Diagrama de Sequência Caso de Uso Gerar Scripts de Teste



Fonte: Autor.

A tabela 14 descreve o caso de uso Cadastrar Roteiros, explanando suas funcionalidades.

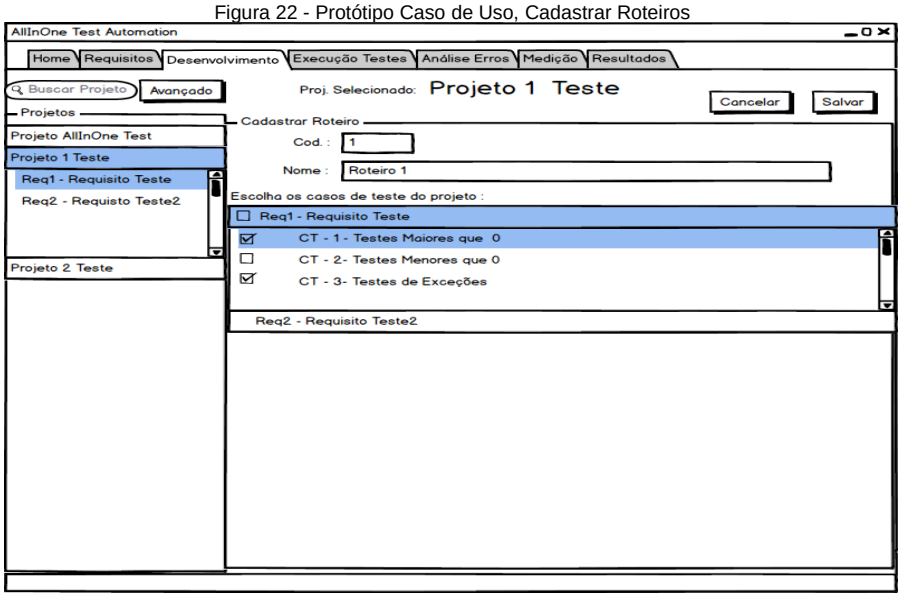
Tabela 14 - Descrição do Caso de Uso, Cadastrar Roteiros

Caso de Uso: Cadastrar Roteiros	
Objetivo:	Permitir cadastrar e manter roteiros.
Descrição básica:	O usuário poderá cadastrar roteiros para agrupamento de casos de teste. Os roteiros são vinculados sempre a um requisito. Após o cadastro de um caso de teste é possível cadastrar roteiros.
Atores:	Usuário.
Pré-Condição	1.Ter atendido ao processo do caso de uso planejar teste. 2. Ter cadastrado casos de teste.
Pós-Condição	Possibilidade de execução de um conjunto de casos de teste.
Fluxo Principal	1.O usuário informa os dados. 2.O usuário seleciona um roteiro ou casos de teste. 3.O usuário clica em salvar.

Fonte: O autor.

Na figura 22 é apresentada a tela para cadastro de roteiro, onde o usuário poderá selecionar casos de testes ou mesmo requisitos para vínculo na execução de um roteiro de teste.

Figura 22 - Protótipo Caso de Uso, Cadastrar Roteiros



Fonte: O autor.

c) Executar Testes

Neste tópico será apresentada a modelagem do protótipo para o módulo de execução de testes. A tabela 15 representa o caso de uso Executar Testes em sua forma descrita e a figura 23 demonstra a tela gráfica do mesmo caso de uso.

Tabela 15 - Descrição do Caso de Uso, Executar Testes

Caso de Uso: Executar Testes	
Objetivo:	Permitir a execução de testes de unidade e regressão.
Descrição básica:	O usuário poderá executar testes individualmente ou executar roteiros de testes. Para a execução poderão ser selecionados, casos de testes, roteiros de teste e projetos, e sua sequência de execução. * Manter os planos de execução. * Inicializar os dados, para a execução dos testes. * Após execução chamar o módulo de análise de teste. * Iniciar módulo de limpeza de dados. * Rastrear linha do lançamento da exceção, caso teste não executado com sucesso.
Atores:	Usuário.
Pré-Condição	1.Casos de testes criados.
Pós-Condição	1. Análise dos testes executados.
Fluxo Principal	1. O usuário informa os dados. 2. O usuário clica em adicionar e escolhe entre roteiro ou caso de teste. 3. O usuário clica em executar teste. 4. O sistema inicializa o módulo utPLSQL para execução de testes e salva os registros retornados. 5. O sistema inicializa o caso de uso Analisar Falhas.

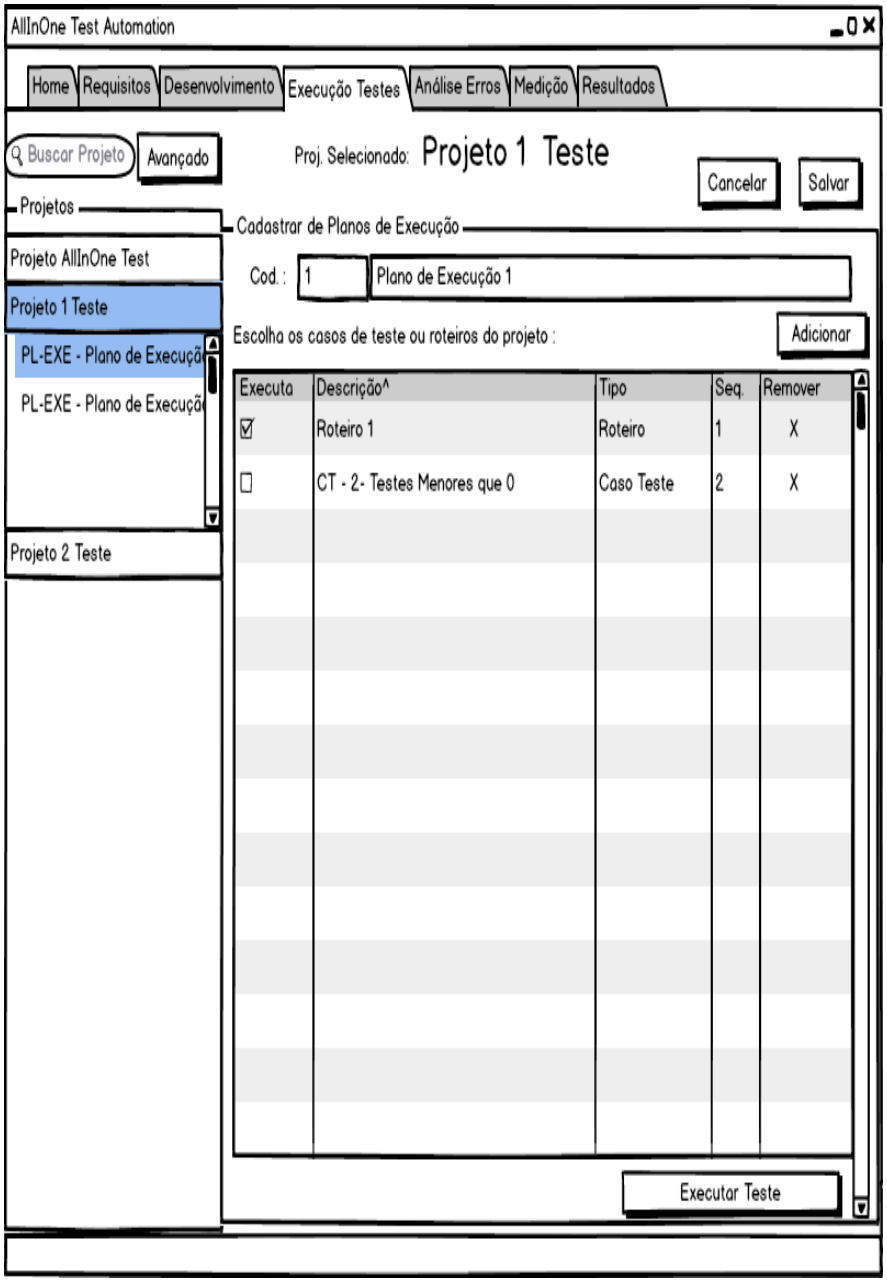
Fonte: O autor.

A figura 24 mostra o Diagrama de Sequência do Caso de Uso Executar Teste, sendo evidenciada a interação das classes e arquitetura da aplicação.

O usuário adiciona roteiros ou casos de testes ao plano de execução de testes e após seleciona aos que deseja executar. A aplicação irá solicitar todos os dados relacionados aos casos de testes vinculados e executar o teste através do framework utPLSQL.

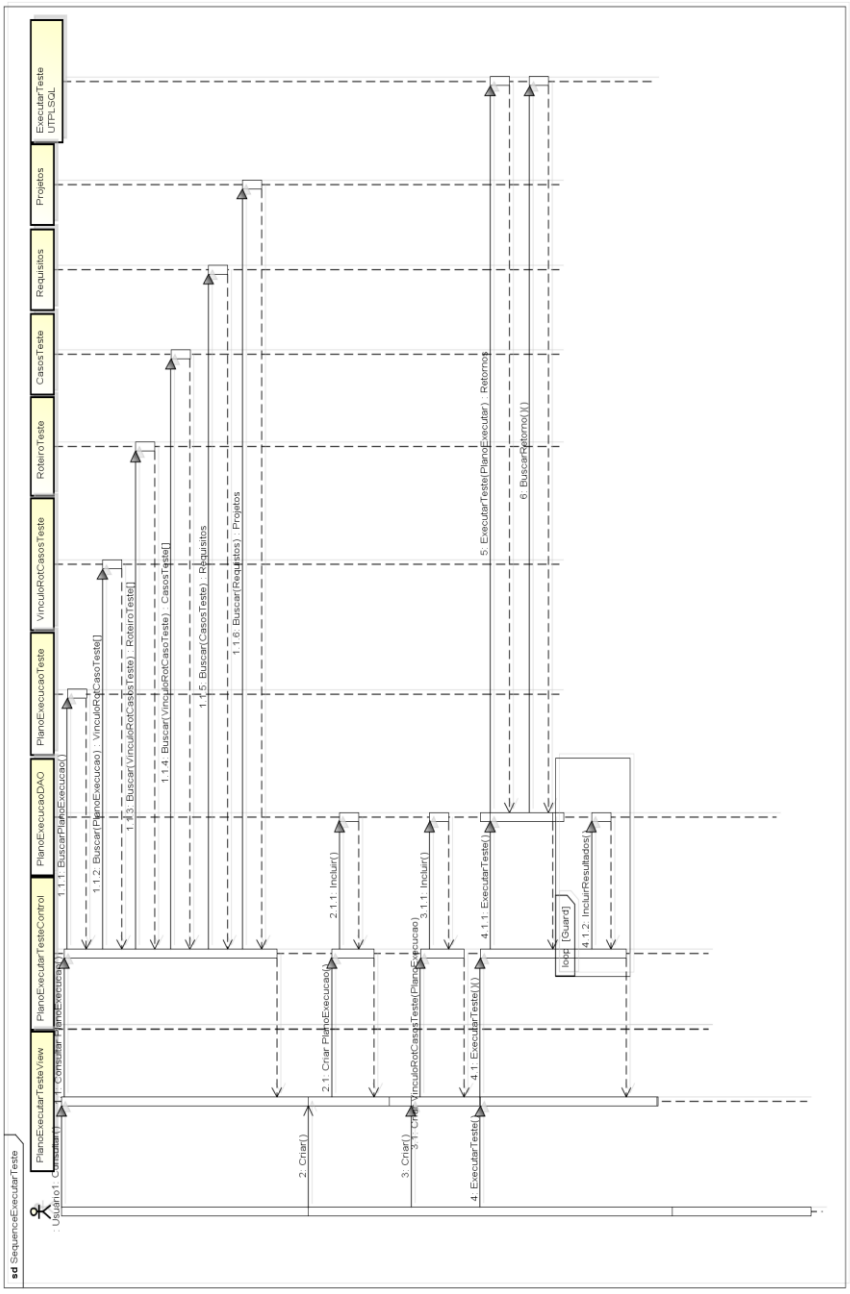
Após a execução do teste, a ferramenta irá buscar os dados armazenados pelo *framework* em tabelas do Oracle, incluindo-as no banco de dados da aplicação.

Figura 23 - Protótipo Caso de Uso, Executar Testes, Cadastro de Planos de Execução



Fonte: O autor.

Figura 24 - Diagrama de Sequência do Caso de Uso Executar Testes



Fonte: O autor.

d) Analisar Falhas

A tabela 16 representa a descrição do caso de uso analisar testes. E a tabela 17 as informações do caso de uso Relatório de Falhas.

Tabela 16 - Descrição do Caso de Uso, Analisar Testes

Caso de Uso: Analisar Testes	
Objetivo:	Permitir a análise de testes já executados, buscando identificar e mostrar ao usuário se executou com sucesso ou não através de relatórios.
Descrição básica:	O sistema irá inicializar o módulo de análise após a execução do plano de teste, retornando os valores passíveis de serem analisados. * Os valores dos retornos, serão comparados com o cadastro dos casos de teste. Se os valores corresponderem ao cadastrado, o módulo de análise deverá, indicar que o teste foi executado com sucesso. * Deverá ser determinado, o horário de execução e data, usuário e tempo de execução a cada caso de teste.
Atores:	Sistema (Software)
Pré-Condição	1.Casos de testes criados. 2. Plano de execução cadastrado.
Pós-Condição	1. Relatório de análises.
Fluxo Principal	1. O usuário seleciona o plano de execução. 2. O sistema carrega os planos de execução disponíveis. 3. O usuário seleciona o plano e clica em relatório. 4.O sistema chama o caso de uso Analisar Testes Anteriores.

Fonte: O autor.

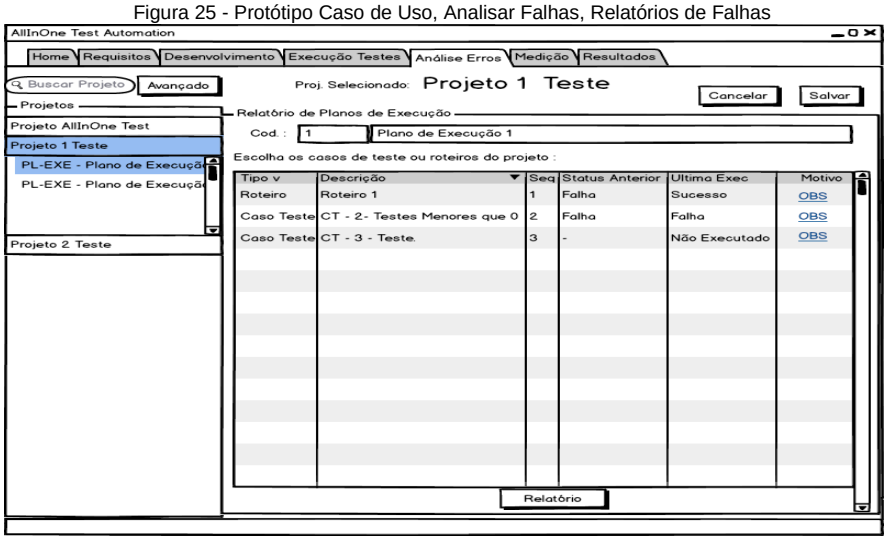
Tabela 17 - Descrição do Caso de Uso, Relatório de Falhas

Caso de Uso: Relatório de Falhas	
Objetivo:	Permitir visualizar os resultados esperados e erros.
Descrição básica:	Visualizar erros do caso de uso em questão.
Atores:	Usuário
Pré-Condição	-
Pós-Condição	1. Relatório de análises.
Fluxo Principal	1. O usuário clica no link OBS, do caso de uso Analisar Falhas. 2. O sistema abre a tela de relatórios de falhas, observações.

Fonte: O autor.

A figura 25 representa a prototipagem de tela contendo informações com o resultado dos casos de testes executados. Nessa tela é possível identificar qual teste teve sucesso e qual falha, analisando assim o motivo no caso de erro.

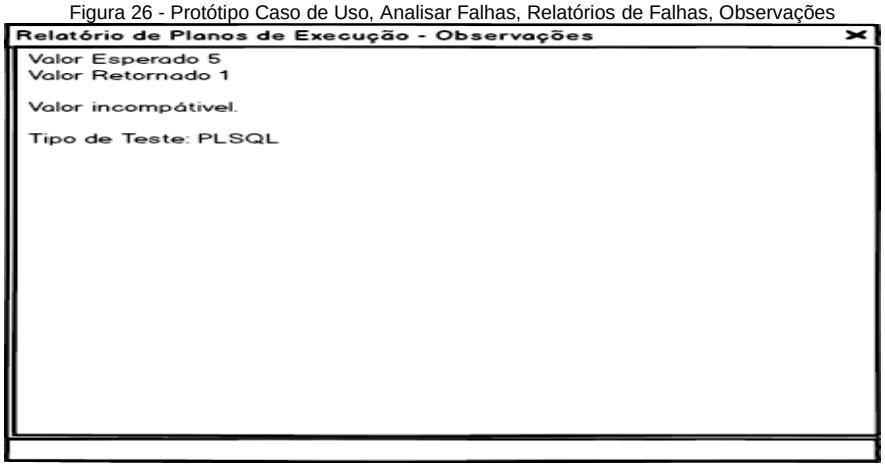
Figura 25 - Protótipo Caso de Uso, Analisar Falhas, Relatórios de Falhas



Fonte: O autor.

A figura 26 representa a tela do caso de uso Relatório de falhas. Essa tela é acionada pelo link OBS da figura 25 e tem o objetivo de demonstrar qual o motivo da falha de um teste.

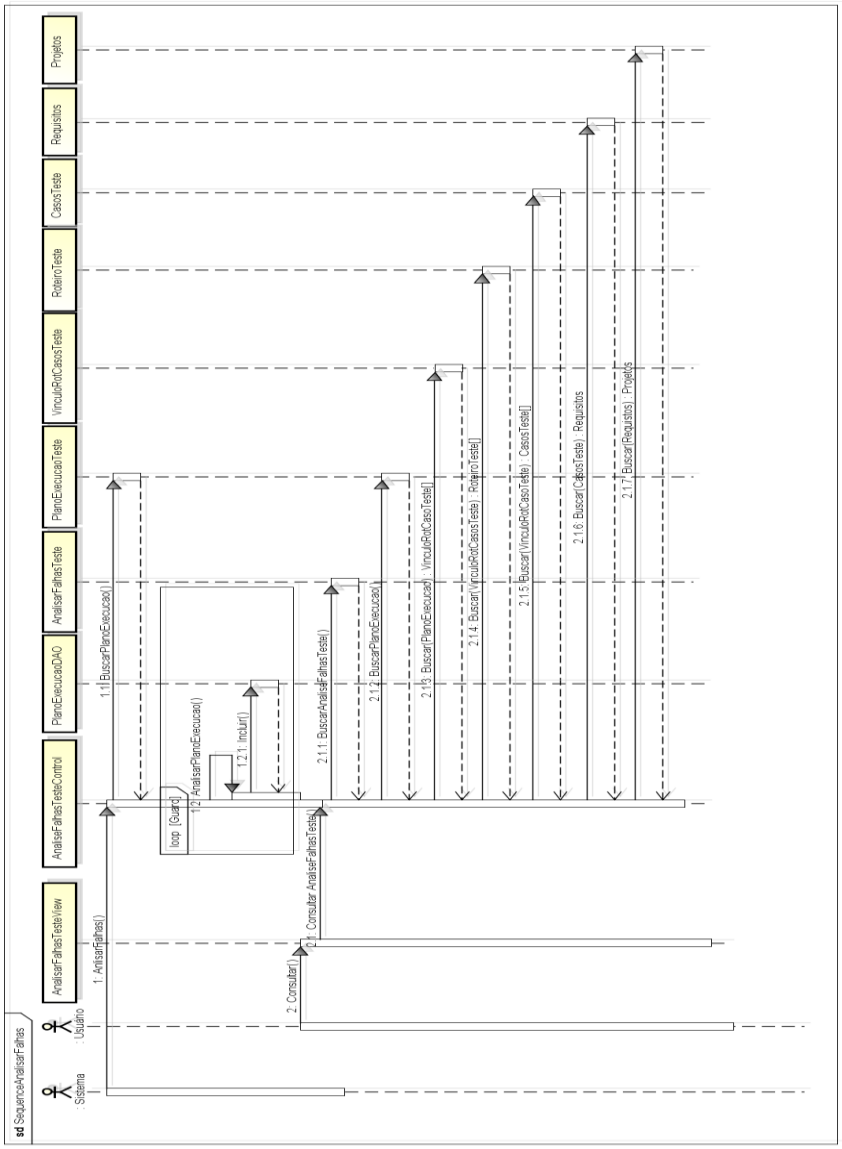
Figura 26 - Protótipo Caso de Uso, Analisar Falhas, Relatórios de Falhas, Observações



Fonte: O autor.

A figura 27 mostra o Diagrama de Sequência Analisar Falhas e expressa a comunicação das classes do caso de uso Analisar Testes.

Figura 27 - Diagrama de Sequência do Caso de Uso Analisar Falhas



Fonte: O autor.

A tabela 18 representa a descrição do caso de uso Analisar Testes Anteriores.

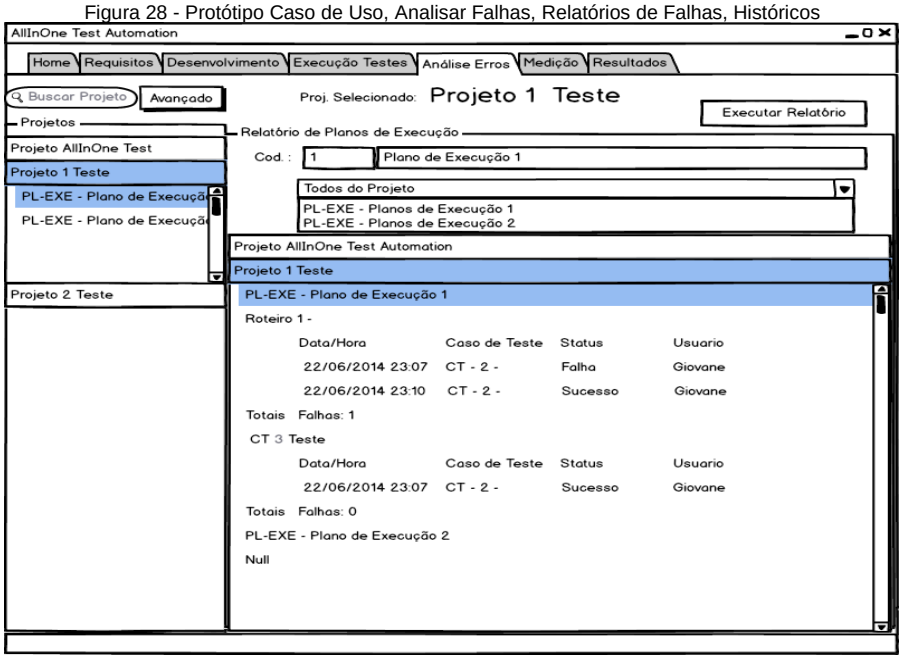
Tabela 18 - Descrição do Caso de Uso, Analisar Testes Anteriores

Caso de Uso: Analisar Testes Anteriores	
Objetivo:	Permitir verificar todo o histórico de planos de execução.
Descrição básica:	* O usuário poderá consultar histórico completo de execução de testes de um plano de execução.
Atores:	Usuário
Pré-Condição	1. Análise de testes já efetuados.
Pós-Condição	-
Fluxo Principal	1. O usuário seleciona o plano de execução. 2. O usuário clica em executar relatório.
Fluxo Alternativo	-

Fonte: O autor.

A figura 28 exibe a tela com detalhes do plano de execução, com as informações temporais para análise do usuário.

Figura 28 - Protótipo Caso de Uso, Analisar Falhas, Relatórios de Falhas, Históricos



Fonte: O autor.

e) Medir Testes

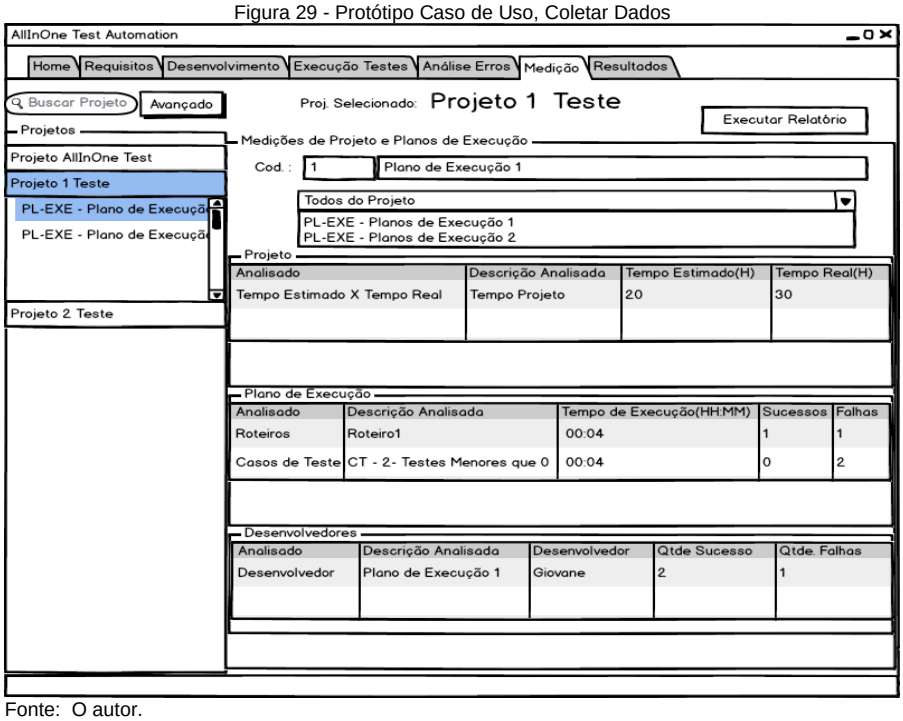
Este tópico apresentará a modelagem do protótipo para o módulo de medição de falhas. A tabela 19 descreve o caso de uso Coletar Dados.

Tabela 19 - Descrição do Caso de Uso, Coletar Dados

Caso de Uso: Coletar Dados	
Objetivo:	Permitir a visualização de dados do projeto sendo como resumo do mesmo.
Descrição básica:	O usuário poderá executar um relatório para visualizar: * Tempo para conserto da falha após a execução. * Falhas por programas e responsáveis. * Tempo utilizado para o plano de teste. * Relação final do tempo estimado x tempo efetivo
Atores:	Usuário.
Pré-Condição	1. Execução de Plano de teste já efetuada.
Pós-Condição	1. Análise dos testes executados.
Fluxo Principal	1. Executar relatório.
Fluxo Alternativo	-

Fonte: O autor.

Figura 29 - Protótipo Caso de Uso, Coletar Dados



Fonte: O autor.

A figura 29 apresentou o protótipo de tela para a aba medir testes. Essa tem a função de exibir os dados do projeto, dos planos de execução e sobre erros encontrados por desenvolvedores.

f) Gerenciamento de configuração dos artefatos de teste

Neste tópico será apresentada a modelagem do protótipo para o módulo de gerenciamento de configuração dos artefatos de teste.

O módulo tem o objetivo de atender ao requisito de rastreabilidade e com base neste é possível identificar casos de testes ou scripts específicos.

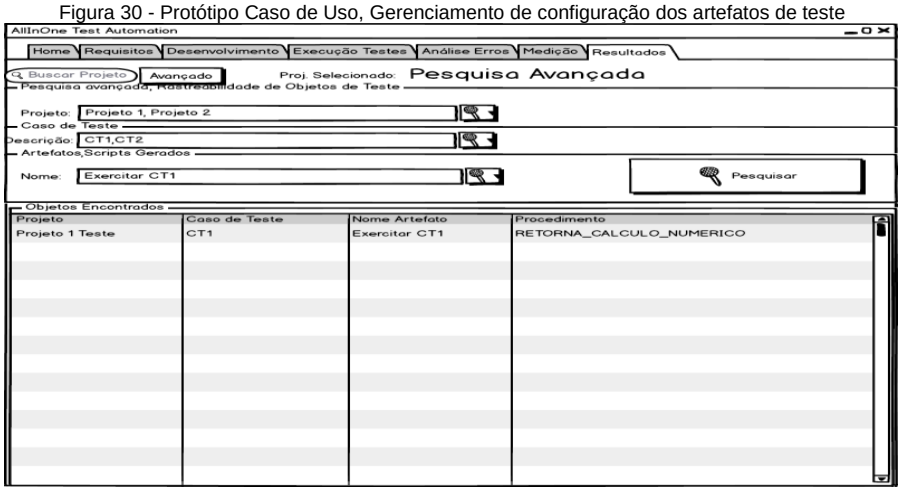
A tabela 20 descreve o caso de uso Rastrear Objeto e a figura 30 mostra a tela para identificação de objetos já criados, com o fim de sua reutilização.

Tabela 20 - Descrição do Caso de Uso, Rastrear Objeto

Caso de Uso: Rastrear Objeto	
Objetivo:	Permitir rastrear objetos já criados.
Descrição básica:	O usuário poderá executar uma pesquisa buscando os dados de um determinado: * Projeto, * Caso de Teste, * Nome de scripts já criados.
Atores:	Usuário.
Pré-Condição	1. Ter scripts de teste já gerados.
Pós-Condição	-
Fluxo Principal	1. Filtrar objetos.

Fonte: O autor.

Figura 30 - Protótipo Caso de Uso, Gerenciamento de configuração dos artefatos de teste



Fonte: O autor.

3.5 IMPLEMENTAÇÃO DO PROTÓTIPO

Essa seção irá apresentar o desenvolvimento da ferramenta com base na metodologia definida neste trabalho. Será ilustrado como foi estruturada, quais os requisitos para instalação e como parametrizar para o uso.

3.5.1 **Estrutura da aplicação**

O desenvolvimento iniciou-se com a estruturação de projetos na ferramenta de desenvolvimento(*IDE*), Visual Studio 2013. A criação de projetos é um método utilizado como forma de divisão da aplicação por camadas, para assim ser possível a visualização e aplicação do modelo *MVVM.*

A figura 31 representa a estrutura definida para o desenvolvimento da ferramenta. Existe uma solução que agrupa todos os projetos da aplicação, e cada projeto tem um objetivo em uma das camadas do modelo MVVM.

No projeto de apresentação são armazenadas todas as telas da ferramenta (*Views*), bem como os objetos *ViewModel,* que são responsáveis por comunicar componentes de tela, por *DataBinding,* ao projeto de modelos e também acionar procedimentos do projeto de negócio. O *ViewModel* pode ser considerado com um controlador e ajuda a retirar o código lógico da visão (*View*).

Por sua vez, o projeto de modelos contém todas as classes de projeto que expõe suas propriedades para gerenciamento de dados. Este projeto é vinculado ao *ViewModel,* que está localizado no projeto de apresentação, e ao projeto de negócio aonde esse irá aplicar as regras de negócio.

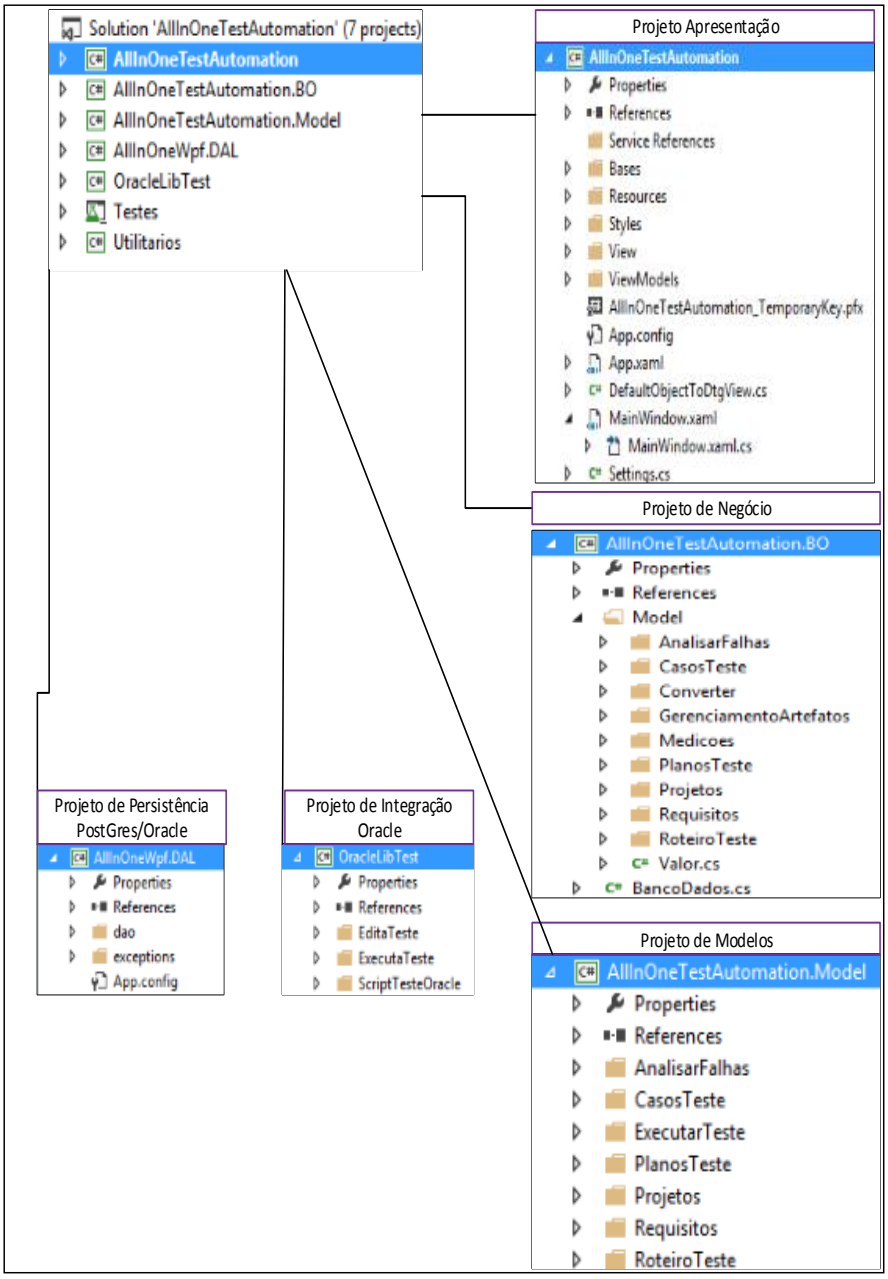
Já o projeto de negócio processa a regra, comunica-se com o projeto de persistência e também com o projeto de apresentação, esse através do *ViewModel*.

O projeto de persistência tem a função de gravar registros e realizar consultas do banco de dados. Seus procedimentos e funções são acionados pelo projeto de negócio.

O projeto de integração Oracle é utilizado para comunicar-se com o framework utPLSQL, o qual irá executar os scripts de testes e retornar os resultados.

A figura 9 deste trabalho representa a arquitetura da aplicação aqui explanada.

Figura 31 - Estruturação de Projetos da Solução AllInOne Test Automation



Fonte: O autor.

3.6 REQUISITOS DE INSTALAÇÃO

Para a utilização da ferramenta como requisitos mínimos são necessários:

a) A utilização do Sistema Operacional *Windows Vista* ou Superior, com 4GB de Memória RAM.
b) Instalação do *NetFramework* versão 4.5 ou superior.
c) Instalação do servidor de banco de dados PostGres na sua última versão ou pode ser utilizado um servidor disponível na rede.
d) Instalação do servidor de banco de dados Oracle na sua última Release, versão XE ou pode ser utilizado um servidor disponível na rede.
e) Instalação e configuração do *Framework* de testes utPLSQL[14].
f) Instalação do Setup.exe da aplicação. Este irá criar todos os registros necessários no sistema operacional.
g) Criação da base de dados no banco de dados PostGres com a identificação de “allinoneautomation” como padrão, ou qualquer desde que configurada no arquivo de configuração da aplicação: “AllInOneTestAutomation.exe.config”.
h) Configuração da *String de Conexão* com o banco de dados Oracle.

Para configurar a S*tring* de conexão[15] é necessário informar os dados de acordo com o driver de comunicação de acesso para cada banco de dados. O anexo B ilustra a parametrização do arquivo de configuração para o banco de dados PostGres e Oracle.

[14] A documentação de instalação pode ser encontrada no endereço: Instalação utPLSQL
[15] String de conexão é o endereço utilizado pelo driver de conexão para comunicação com o banco de dados.

4 SIMULAÇÃO DA FERRAMENTA ALLINONE TEST AUTOMATION

A simulação da ferramenta tem o objetivo de validar os requisitos elencados e encontrar ajustes na ferramenta.

Os objetivos da simulação foram norteados pelos seguintes itens:

a) Validar os requisitos propostos.
b) Revelar falhas de desenvolvimento e lógica divergentes dos requisitos da ferramenta.
c) Avaliar a utilização da ferramenta.

Esta seção irá demonstrar a utilização da ferramenta conduzida por um usuário chave. Esse usuário é conhecedor dos processos de desenvolvimento na linguagem PL/SQL com procedimentos armazenados no SGBD Oracle, porém não possui conhecimento prático sobre a ferramenta.

A simulação foi aplicada em um projeto simulado baseado em um projeto real, com tempo de desenvolvimento total de 73 horas.

4.1 ESPECIFICAÇÕES DO PROJETO PARA SIMULAÇÃO

O projeto escolhido para a simulação tem o objetivo de vincular ordens de fabricação com itens do pedido de venda, na execução do planejamento de recursos de manufatura, do Inglês *Manufacturing Resource Planning (MRP)*.

O processo inicia-se no cadastro da demanda do cliente gerando assim o pedido de venda. A partir da sua liberação, este fica disponível para planejamento e o próximo passo é executar o MRP, para calcular as estimativas de demandas e previsões de entrega, planejando assim as ordens de compra e fabricação.

O projeto encaixa-se na execução do MRP, vinculando as ordens planejadas e ordens já liberadas ao pedido de venda, considerando também os itens da estrutura. Para a simulação, as estruturas dos itens devem estar previamente cadastradas e os dados já gerados em tabelas de dados no Oracle.

Com base nestes aspectos do projeto, a simulação com a ferramenta AllInOne Test Automation deverá validar os procedimentos desenvolvidos para a geração dos vínculos, testando os seguintes itens:

a) Se para um item do pedido de venda não existir vínculo com ordem e antes da execução do MRP, deverá existir após o cálculo do planejamento.
b) Caso já existam ordens liberadas no processo, testar se o vínculo foi efetuado com alguma ordem.
c) Existe uma tabela no banco de dados onde é gerada a informação da quantidade de vínculo da ordem com o pedido, sendo assim, a ferramenta deverá somente validar se a quantidade do vínculo é a mesma do pedido, independentemente se forem ordens planejadas ou liberadas.

4.2 SIMULAÇÃO DOS CASOS DE USO

Serão utilizadas capturas de telas que representarão a simulação e suas respectivas considerações, explanadas de forma objetiva, iniciando assim a validação de acordo com a sequência dos casos de uso explicados neste trabalho.

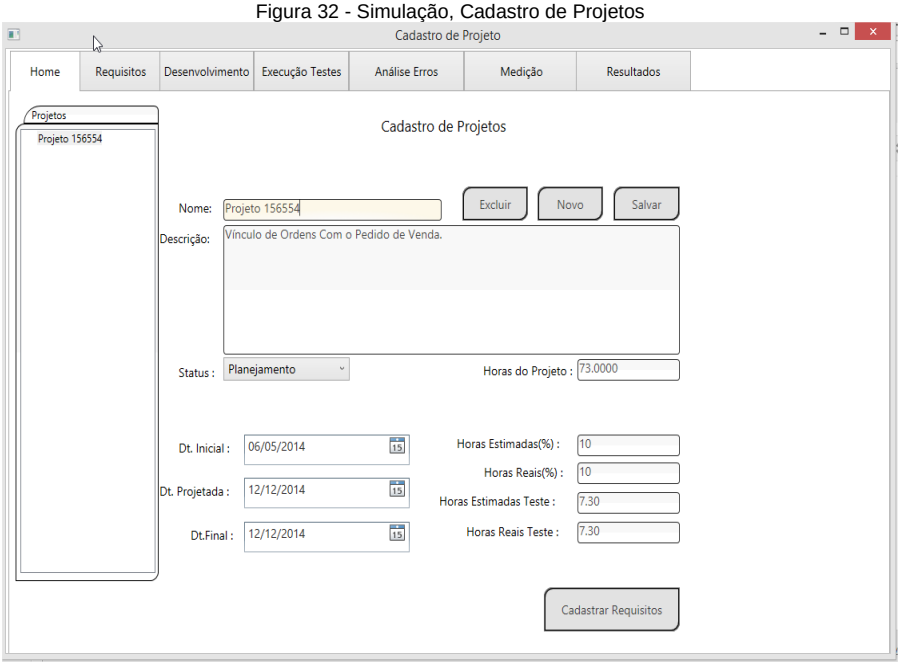
4.2.1 **Cadastro do Projeto**

No cadastro de projeto foi necessário informar os dados para cada campo testando a inclusão, deleção e alteração dos dados. Na ferramenta foram informados os dados conforme a figura 32 e para salvar foi acionado o botão salvar.

O projeto 156554 possui 73 horas de desenvolvimento e conforme os dados apresentados 10% serão atribuídos para os testes.

Ao clicar em “Salvar”, a ferramenta efetiva a transação de dados, no banco de dados Postgres, atualizado a tela e a árvore localizada a esquerda identificando os projetos e o selecionado.

Figura 32 - Simulação, Cadastro de Projetos



Fonte: O autor.

Analisando a figura 32 pode-se observar que ela atendeu os requisitos definidos para o caso de uso Cadatrar Projetos. Após este cadastro a próxima sequência da simulação é o caso de uso, Cadastrar Requisitos.

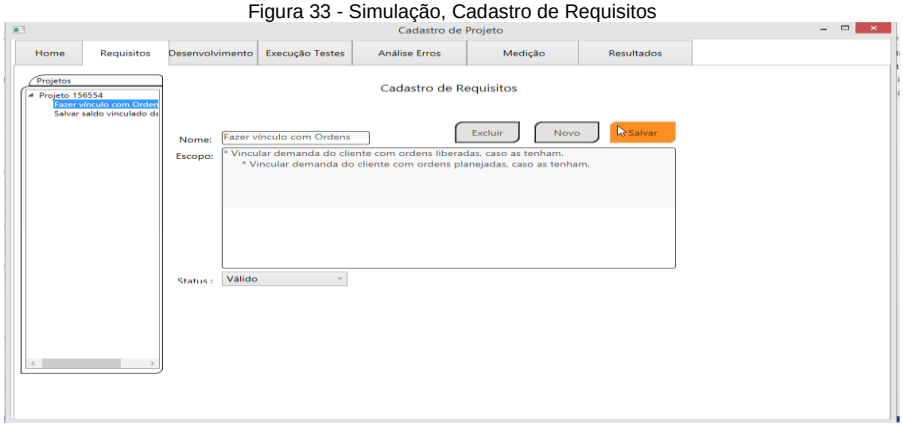
4.2.2 **Cadastro de Requisitos**

O cadastro de requisitos é similar ao processo de cadastro de projetos. O usuário informa os dados do requisito e salva o mesmo. Caso queria inserir um novo requisito, com o foco sobre o projeto na árvore de projetos, é possível informando os dados desse requisito e salvar.

A figura 33 mostra dois requisitos salvos no projeto, com as informações sobre o vínculo com o pedido. O primeiro requisito informado, conforme figura 34, tem como escopo o vínculo de itens do pedido com ordens liberadas e caso não as tenham, os vínculos devem existir com ordens planejadas.

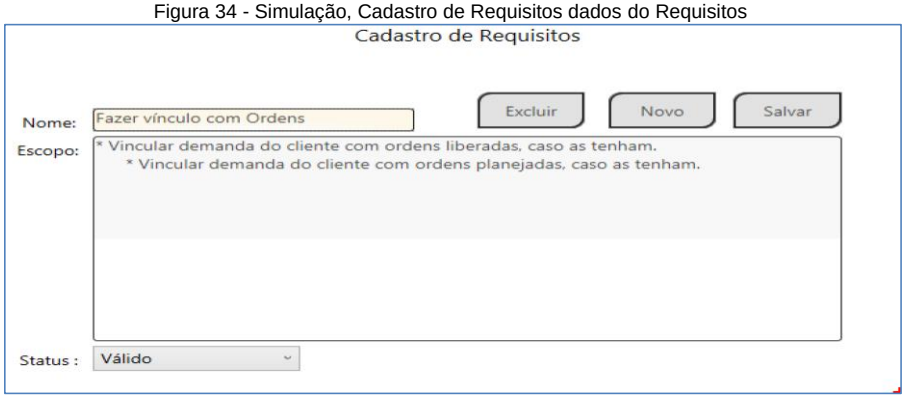
A figura 35 apresenta o segundo requisito do projeto, identificando que cada ordem deve ter a quantidade do vínculo para cada item.

Figura 33 - Simulação, Cadastro de Requisitos



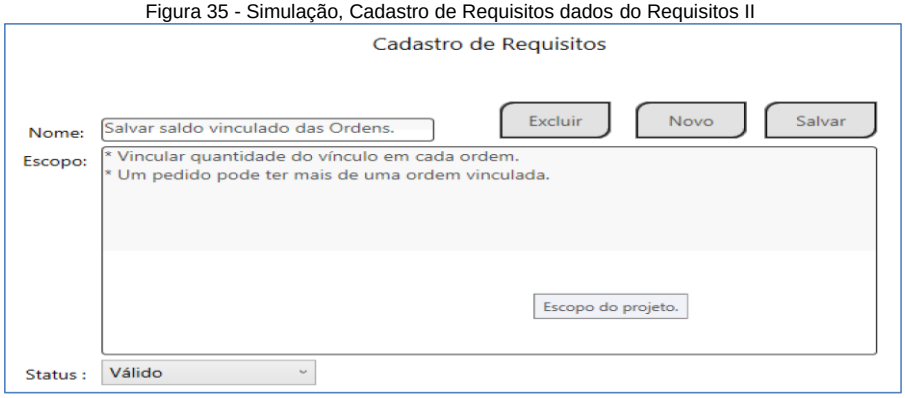
Fonte: O autor.

Figura 34 - Simulação, Cadastro de Requisitos dados do Requisitos



Fonte: O autor.

Figura 35 - Simulação, Cadastro de Requisitos dados do Requisitos II

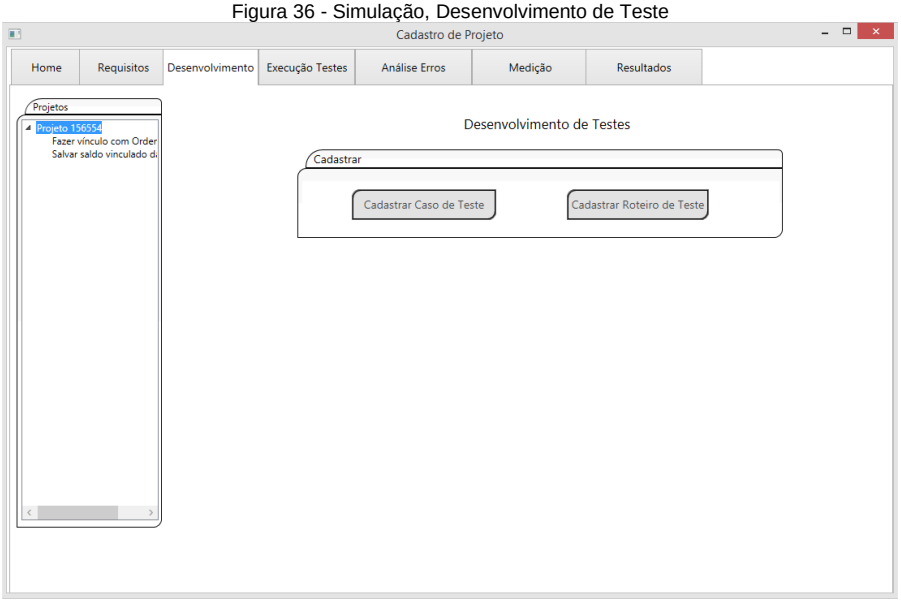


Fonte: O autor.

4.2.3 **Desenvolvimento de Testes**

Para o teste do caso de uso Desenvolver Teste, foi necessário respeitar a pré-condição do cadastro de projetos, pois caso não o tenha, o sistema invalida o fluxo. A figura 36 apresenta a tela para os cadastros seguintes, neste caso é necessário executar o cadastro do caso de teste, pois o roteiro necessita do mesmo.

Figura 36 - Simulação, Desenvolvimento de Teste



Fonte: O autor.

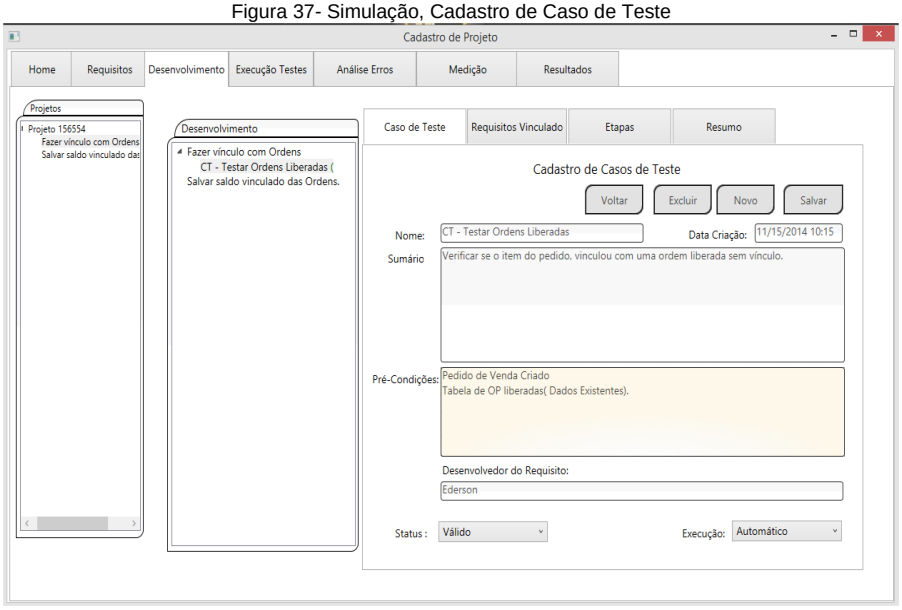
4.2.4 **Cadastro de Casos de Teste**

A simulação do processo para o caso de uso Cadastrar Casos de Teste, inicia-se ao clicar no botão Cadastrar Caso de Teste conforme a figura 36.

Os dados informados na tela disponibilizada são apresentados na figura 37. Foi informado para o requisito, “Fazer vínculo com Ordens”, que este caso de teste irá verificar se o item do pedido, vinculou com ordem já liberada e que essa ordem esteja sem vínculo. Ou seja, se a ordem não tiver vínculo com nenhum outro item de pedido, então deverá vincular com o pedido em questão.

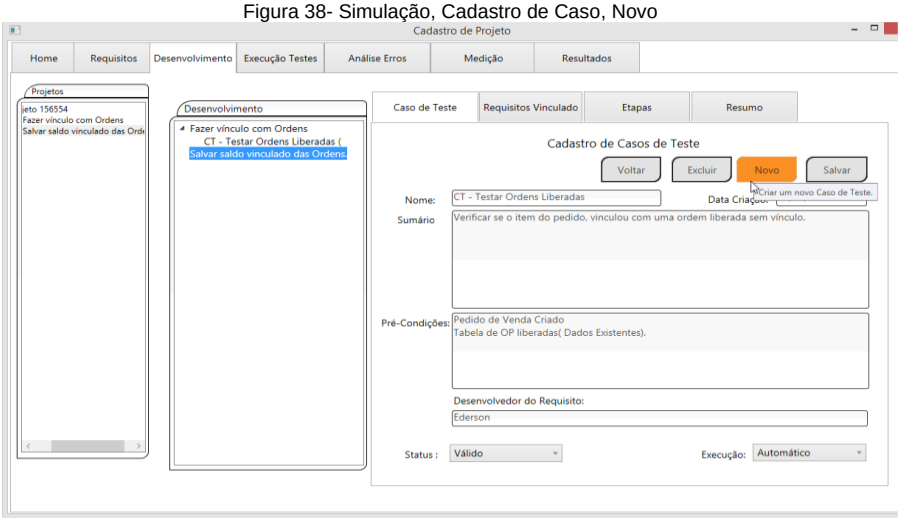
Para criar um novo caso de teste no mesmo requisito é necessário selecionar o requisito na árvore “Desenvolvimento” e acionar o botão “Novo”, conforme a figura 38.

Figura 37- Simulação, Cadastro de Caso de Teste



Fonte: O autor.

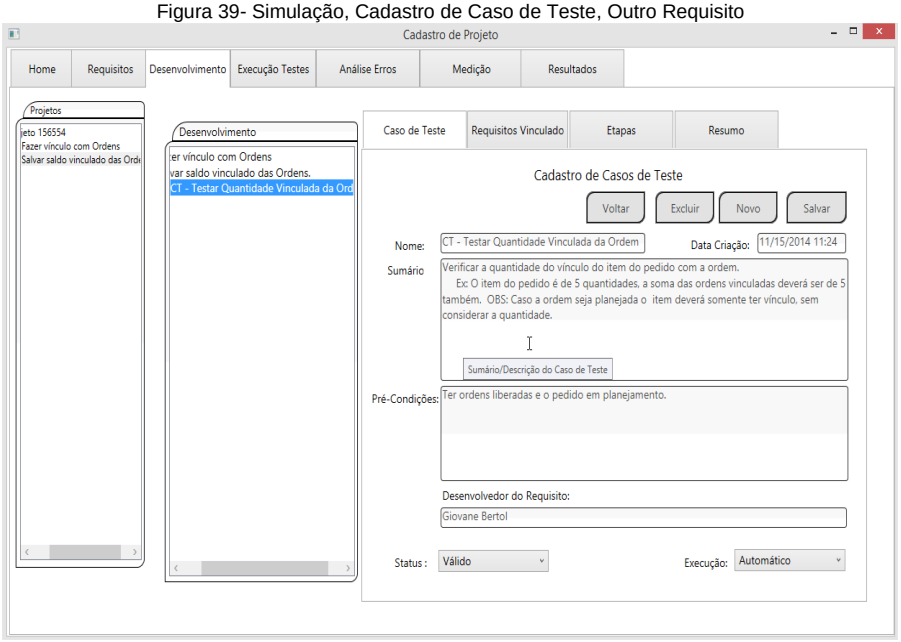
Figura 38- Simulação, Cadastro de Caso, Novo



Fonte: O autor.

A figura 39 apresenta outro caso de teste criado na simulação. Esse tem o objetivo de validar a quantidade vinculada do item do pedido com as ordens liberadas.

Figura 39- Simulação, Cadastro de Caso de Teste, Outro Requisito

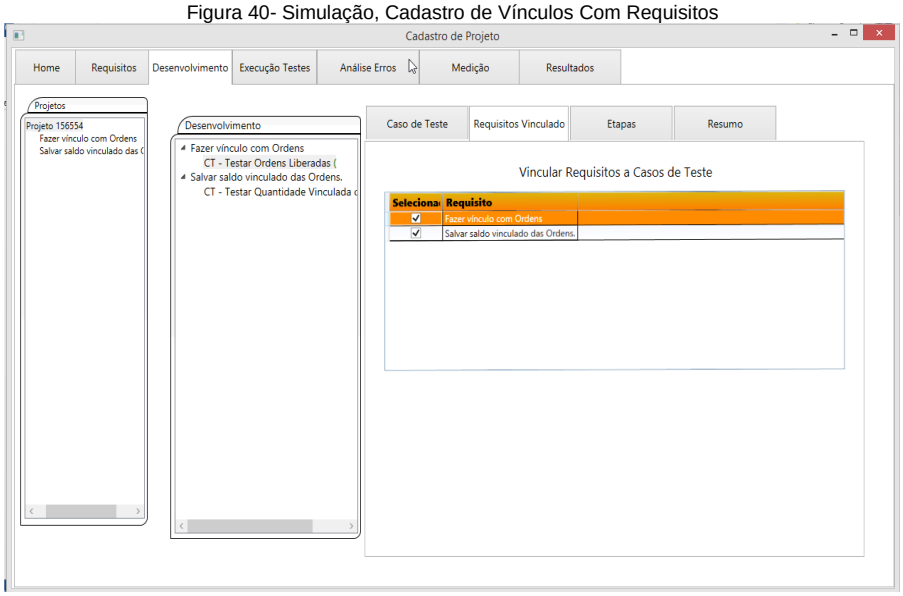


Fonte: O autor.

Após o cadastro de casos de teste é necessário vincular o mesmo a um requisito, pois somente vinculando o caso de teste a um requisito, esse ficará disponível para ser incluído em um roteiro de teste. O mesmo caso de teste pode ser também atrelado a outro requisito que não deu a sua origem. Assim, quando cadastrado um roteiro será possível incluir casos de teste de outros requisitos.

Na figura 40 é apresentado o cadastro de vínculos com requisitos. O usuário selecionou os dois requisitos existentes no projeto, sendo assim para cada roteiro atrelado ao requisito, estará ativo o caso de teste. O próximo passo é cadastrar as etapas de teste.

Figura 40- Simulação, Cadastro de Vínculos Com Requisitos



Fonte: O autor.

4.2.5 **Cadastro de Etapas de Teste**

O cadastro de etapas de teste é acionado ao selecionar a tab “Etapas”, localizado na janela do desenvolvimento de testes, essa pode ser visualizada na figura 41. Para cadastrar uma nova etapa deverá ser selecionado um caso de teste da árvore e clicar no botão novo.

Ainda, na mesma figura é demonstrado os dados informados pelo usuário e pode-se concluir que nesta etapa, o teste deverá validar se um determinado item do pedido de venda esteja vinculado a uma ordem liberada.

Nesta etapa de teste foi informado apenas um parâmetro de entrada e um parâmetro de saída, esse o resultado esperado. Julga-se que o procedimento que buscará essas informações seja compatível com esses parâmetros.

A figura 42 mostra outra etapa de teste cadastrada, porém nessa, o teste é para um item que não esteja vinculado. Sendo assim uma das duas etapas deverá invalidar o teste em questão, pois o item só poderá ter um dos dois status, vinculado ou não.

Figura 41- Simulação, Cadastro de Etapas de Teste

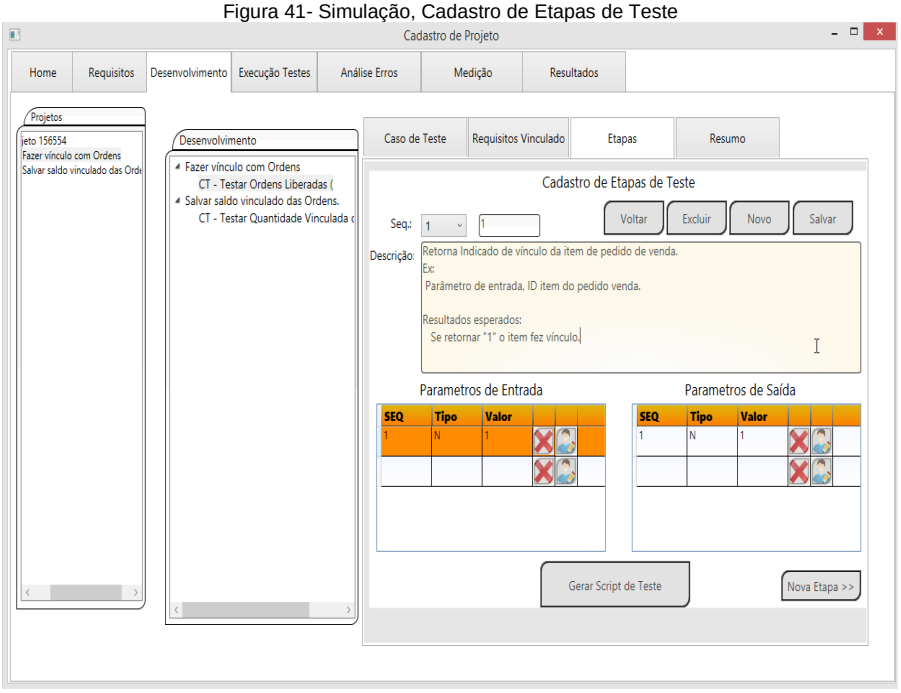
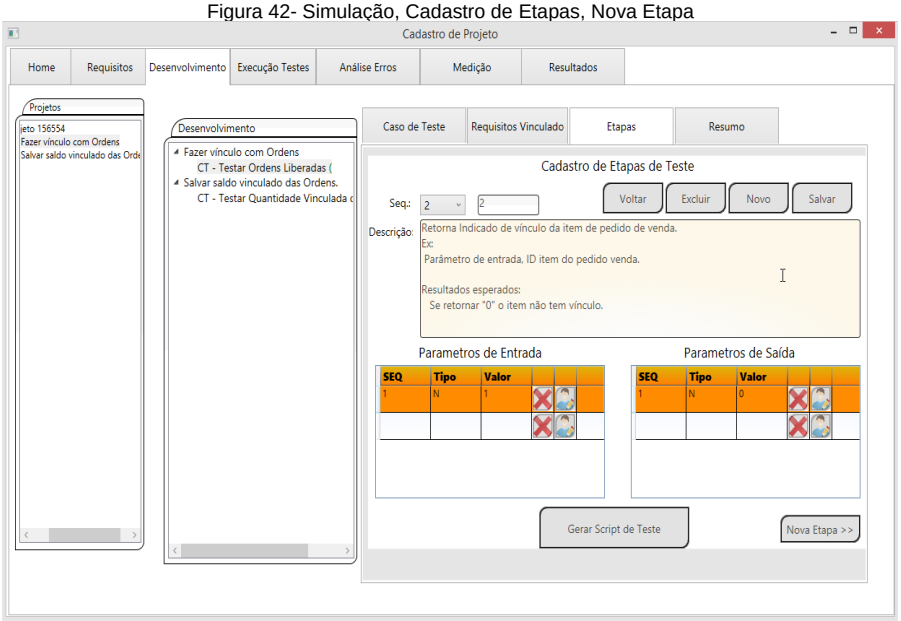


Figura 42- Simulação, Cadastro de Etapas, Nova Etapa

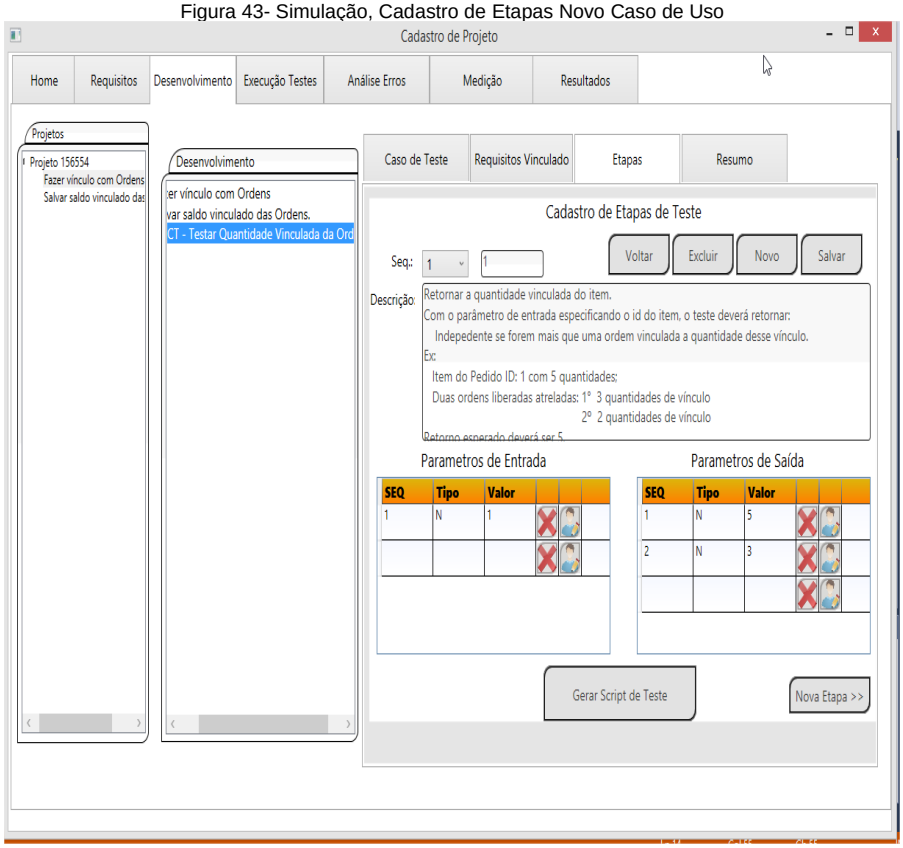


Fonte: O autor.

A figura 43 apresenta o cadastro de uma etapa para o segundo caso de teste. Lembrando que o objetivo desse é validar a quantidade vinculada as ordens de um item em questão. Para cadastrar esta etapa o usuário precisa selecionar o caso de teste e navegar até a tab “Etapas”.

Ainda, analisando os parâmetros da etapa conforme a figura 42, pode-se verificar que para o item de ID 1 (Primeiro parâmetro de entrada informado) os resultados esperados para a 1º sequência é 5 e para a 2º sequência 3. Como o sistema valida um caso de teste, logo, esse irá ter o resultado de falho pelo fato de um item poder ter apenas um valor vinculado.

Figura 43- Simulação, Cadastro de Etapas Novo Caso de Uso



Fonte: O autor.

Nessa validação foram encontradas pelo usuário algumas observações na ferramenta, apresentadas pela tabela 21.

Tabela 21 – Cadastro de Etapas de Teste, alterações sugeridas pela simulação

Identificação	Descrição	Finalidade
Inclusão de Parâmetros de Entrada e Saída	Para a Inclusão de parâmetros de entrada e saída no campo “Tipo” seria mais usual que a ferramenta utilize uma lista de seleção dos mesmos.	Diminuir as chances de erros do usuário no cadastro dos parâmetros.
Usabilidade	Para voltar e cadastrar um roteiro é necessário navegar até a Tab. Caso de Teste. Criar um botão voltar em todas as Tabs. Do cadastro.	Agilizar o processo no uso da ferramenta.

Fonte: O autor.

O próximo passo é acionar o caso de uso Gerar Scripts de Teste. Esse é acionado ao clicar no botão “Gerar Script de Teste” e pode-se analisar a descrição do caso de uso na tabela 13.

4.2.6 Geração de Scripts de Teste

Na geração do script de teste é imprescindível informar todas os campos apresentados pelo sistema. A figura 44 demonstra o exemplo utilizado pela simulação e na tabela 22 é explanado cada campo do sistema para facilitar a operação do usuário.

Tabela 22 - Descrição e finalidade de cada campo da Geração de Scripts de teste

Campo	Descrição	Finalidade
Nome	Texto que será apresentado pelo teste após a execução.	Facilitar análise de teste.
Procedimento	Identifica o nome do procedimento a ser testado.	O sistema irá gerar o script de teste com o nome do procedimento, enviando os parâmetros de entrada e saída.
Tipo Procedimento	Se é procedimento ou função.	O sistema irá gerar os testes retornando valores diretamente quando função ou quando procedimento irá declarar variáveis para que cada parâmetro de saúda seja testado.

Campo	Descrição	Finalidade
Tipo de dado	Indispensável informar quando função.	O sistema deverá realizar as conversões corretas.
Etapa	Qual etapa que será atrelada.	Buscar os parâmetros da etapa.
Tipo	Qual linguagem será testada.	Gerar o script de acordo com o tipo escolhido.
Edição de Script	O script de teste gerado.	Script que será utilizado para os testes (Teste de Unidade).
Gerar Script	Botão para gerar o script de teste.	Sempre será gerado um script novo, baseado nas informações do cadastro de geração de script e dos parâmetros informados para a etapa.
Salvar	Salvar os dados informados no cadastro.	Sempre quando for necessário a edição do script de teste, será necessário salvar por este botão. Pois ele não irá regerar o teste e sobrescrever as informações do script alterado.
Excluir	Excluir o *script*.	O sistema irá excluir o cadastro do script de teste referente a etapa selecionada.

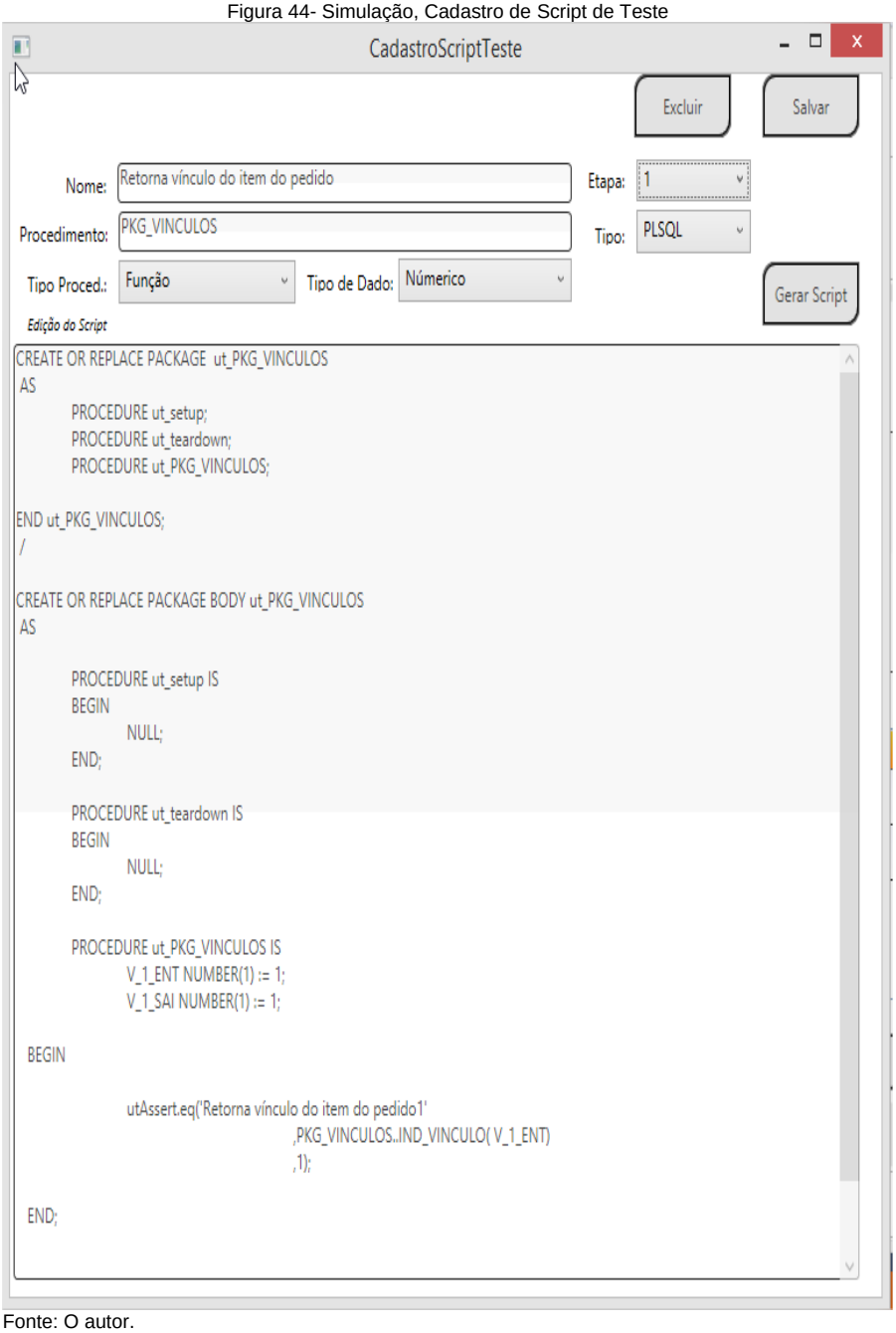
Fonte: O autor.

Conforme os dados informados pelo usuário, o processo gerou o código contendo: A descrição do procedimento informado, como sendo o nome do teste; O nome do *script* de teste, como sendo a descrição do teste ou também chamada de mensagem identificadora do teste; Os testes do procedimento onde é testado cada parâmetro de saída.

A figura 44 exemplifica bem o funcionamento da ferramenta para geração de testes, pois o exemplo foi gerado para avaliar um pacote de procedimentos. Um pacote de procedimentos, na linguagem PLSQL, é conhecido como *Package.* Onde primeiramente é declarado o nome dessa *Package* e após o nome do procedimento acoplado.

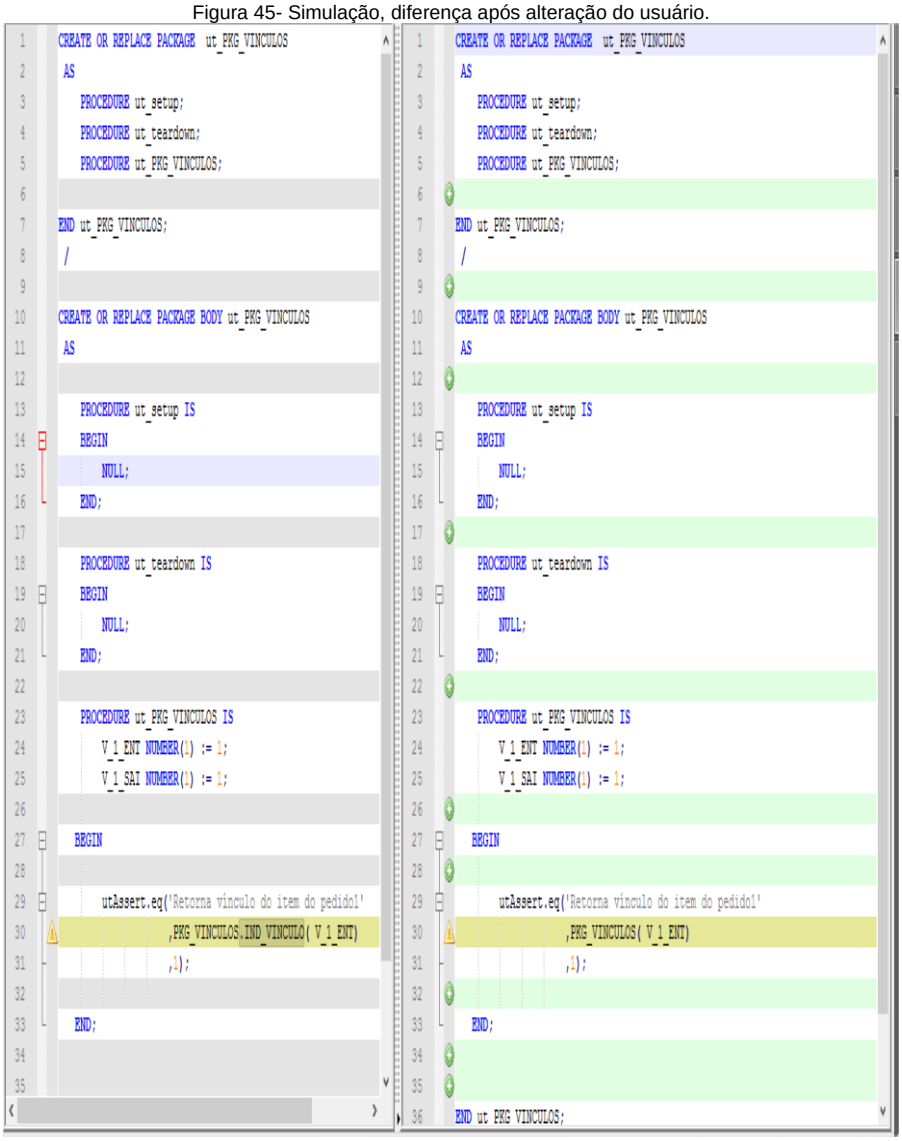
Ainda com base na figura 44, o código gerado pelo processo foi modificado pelo usuário, pois a função que será testada está armazenada em uma *Package* e a ferramenta não monta o código de testes quando utilizadas rotinas nessas *Packages*. Portanto, foi incluído manualmente o código sendo possível averiguar na figura 45 a diferença dos testes gerados pela ferramenta (à direita) e após a alteração do mesmo pelo usuário (à esquerda).

Figura 44- Simulação, Cadastro de Script de Teste



Fonte: O autor.

Figura 45- Simulação, diferença após alteração do usuário.



```
CREATE OR REPLACE PACKAGE  ut_PKG_VINCULOS
 AS
    PROCEDURE ut_setup;
    PROCEDURE ut_teardown;
    PROCEDURE ut_PKG_VINCULOS;

END ut_PKG_VINCULOS;
 /

CREATE OR REPLACE PACKAGE BODY ut_PKG_VINCULOS
 AS

    PROCEDURE ut_setup IS
    BEGIN
       NULL;
    END;

    PROCEDURE ut_teardown IS
    BEGIN
       NULL;
    END;

    PROCEDURE ut_PKG_VINCULOS IS
       V_1_ENT NUMBER(1) := 1;
       V_1_SAI NUMBER(1) := 1;

  BEGIN

       utAssert.eq('Retorna vínculo do item do pedido1'
                  ,PKG_VINCULOS.IND_VINCULO( V_1_ENT)
                  ,1);

    END;
```

```
CREATE OR REPLACE PACKAGE  ut_PKG_VINCULOS
 AS
    PROCEDURE ut_setup;
    PROCEDURE ut_teardown;
    PROCEDURE ut_PKG_VINCULOS;

END ut_PKG_VINCULOS;
 /

CREATE OR REPLACE PACKAGE BODY ut_PKG_VINCULOS
 AS

    PROCEDURE ut_setup IS
    BEGIN
       NULL;
    END;

    PROCEDURE ut_teardown IS
    BEGIN
       NULL;
    END;

    PROCEDURE ut_PKG_VINCULOS IS
       V_1_ENT NUMBER(1) := 1;
       V_1_SAI NUMBER(1) := 1;

  BEGIN

       utAssert.eq('Retorna vínculo do item do pedido1'
                  ,PKG_VINCULOS( V_1_ENT)
                  ,1);

    END;

END ut_PKG_VINCULOS;
```

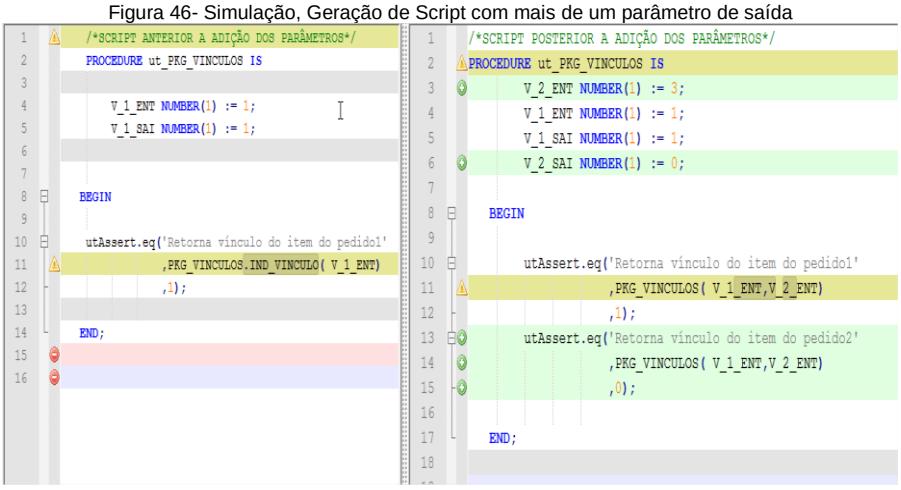
Fonte: O autor.

Caso sejam informados outros parâmetros de saída ou entrada, o *script* irá somente aderir-se a essas modificações se acionado o botão Gerar Script. Contudo todas alterações realizadas manualmente serão sobrescritas.

Para exemplificar a figura 46 mostra o mesmo *script* de teste, ilustrado na figura 45, demostrando que após a inclusão de novos parâmetros de saída e

entrada, ao clicar no botão Gerar *Script* a ferramenta irá regerar o código sobrescrevendo o código manipulado pelo usuário.

Figura 46- Simulação, Geração de Script com mais de um parâmetro de saída



Fonte: O autor.

Para gerar o *script* de outras etapas, basta selecionar no *combo box* a etapa requerida e incluir ou modificar os campos que forem necessários. Também é importante reportar que as validações geradas podem ser alteradas, desde que restrinjam-se a estrutura do *framework* utPSLQL.

A execução de testes funciona de acordo com o código do *script* da primeira etapa.

A tabela 23 apresenta sugestões de melhorias verificada pelo usuário.

Tabela 23 – Gerar Script de Teste, alterações sugeridas pela simulação

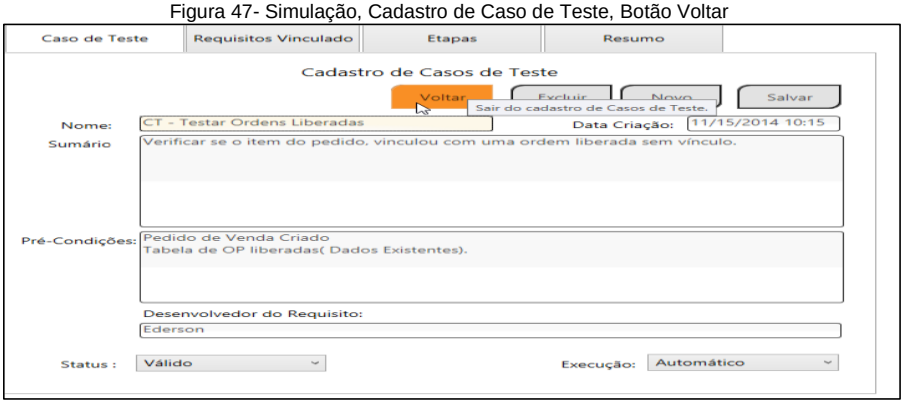
Identificação	Descrição	Finalidade
Gerar Script de Teste	Quando selecionada outra etapa, a ferramenta deveria trazer os dados da 1° etapa para a próxima.	Agilizar o cadastro e gerar maior assertividade. Isso porque ao cadastrar um script de outra etapa, o mesmo sobrescreve os dados de todas as outras.
Usabilidade	Não existe mensagem indicando as alterações dos dados do script. Criar essa mensagem para que o usuário não gere indevidamente um novo script, podendo comprometer o anterior	Prevenir erros de usuários.

	modificado.	
Funcionalidade	Prover versões de scripts anteriores.	Caso um script ser alterado e der algum problema, facilitar a escolha de versões anteriores.

Fonte: O autor.

A próxima etapa da simulação é validar o caso de uso Cadastrar Roteiro e para acioná-lo, caso esteja no cadastro de casos de testes, é necessário que a "*Tab.*" selecionada seja a de "Casos de Teste" conforme figura 47. Assim, estará ativo o botão voltar que redirecionará para o caso de uso "Desenvolver Teste".

Figura 47- Simulação, Cadastro de Caso de Teste, Botão Voltar

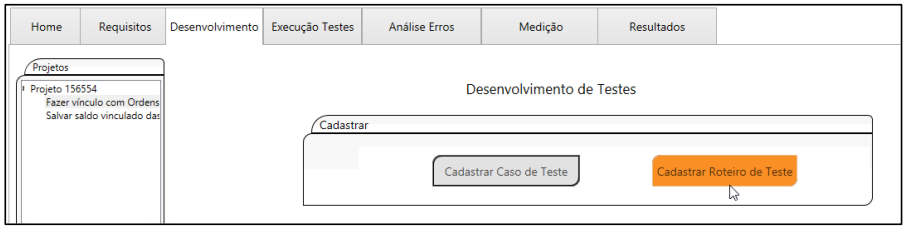


Fonte: O autor.

4.2.7 **Cadastro de Roteiros**

Após ser adicionado os casos de teste, os roteiros poderão ser cadastrados. A figura 48 apresenta como acessar o Cadastro de Roteiros, onde a Tab. "Desenvolvimento" deverá estar selecionada.

Figura 48- Simulação, Desenvolvimento de Testes, Botão Cadastrar Roteiro

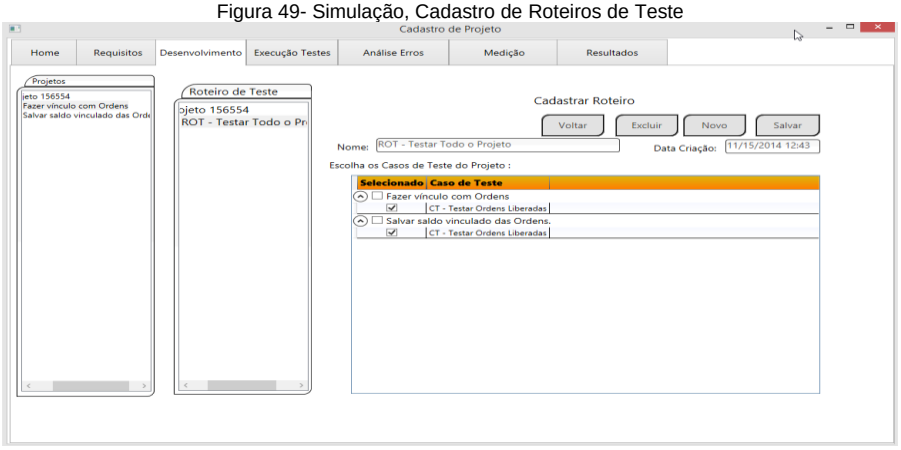


Fonte: O autor.

No cadastro de roteiros é possível cadastrar inúmeros roteiros para cada requisito e cada requisito, poderá executar todos os casos de testes que estão vinculados a ele (Esse vínculo é realizado no cadastro de casos de testes, figura 40).

Na simulação a figura 49 ilustra o roteiro criado para a execução dos casos de testes incluídos pelo usuário. Analisando os dados informados, pode-se perceber que para este roteiro serão vinculados todos os requisitos com o mesmo caso de teste.

Figura 49- Simulação, Cadastro de Roteiros de Teste



Fonte: O autor.

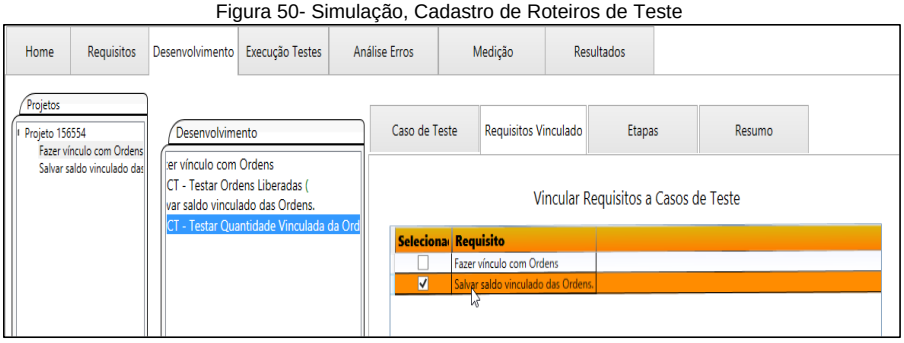
Ainda na figura 49, é passível de visualização apenas um dos casos de testes. Isso acontece pelo fato do usuário não ter vinculado o segundo caso de teste cadastrado a um requisito.

Foi necessário acessar a tela de vínculos no cadastro de casos de teste, para vincular o segundo caso de teste cadastrado. Assim o cadastro ficou conforme é ilustrado nas figuras 50 e 51.

Conforme descrito anteriormente, a figura 50 demonstra que o vínculo para o segundo caso de teste fora realizado, agora estará disponível para seleção no cadastro de roteiros de teste.

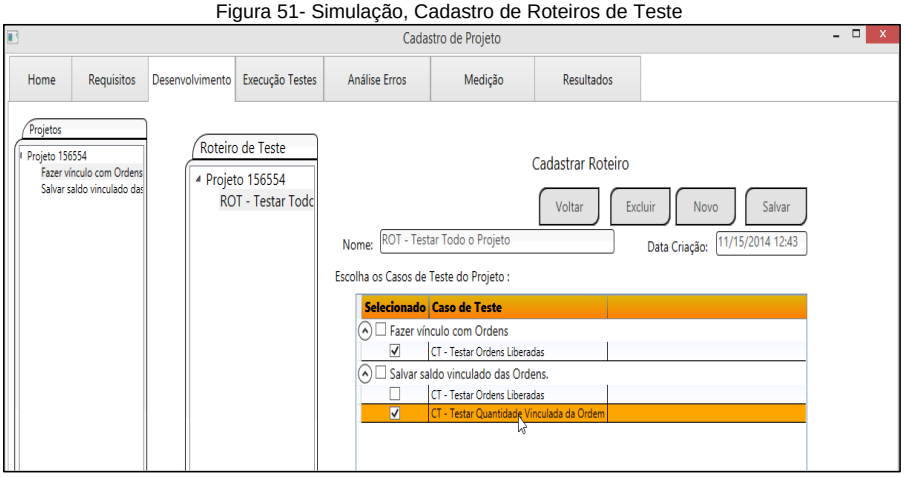
A figura 51 mostra que para o requisito “Salvar saldo vinculado das ordens”, apenas o “CT – Testar Quantidade Vinculada da Ordem” será executado e para o requisito “Fazer vínculo com Ordens” somente o caso de teste “CT – Testar Ordens Liberadas”.

Figura 50- Simulação, Cadastro de Roteiros de Teste



Fonte: O autor.

Figura 51- Simulação, Cadastro de Roteiros de Teste



Fonte: O autor.

Realizado esse processo, o próximo passo é acionar o caso de uso Executar Testes, onde o mesmo está disponível através da Tab. “Execução Testes”.

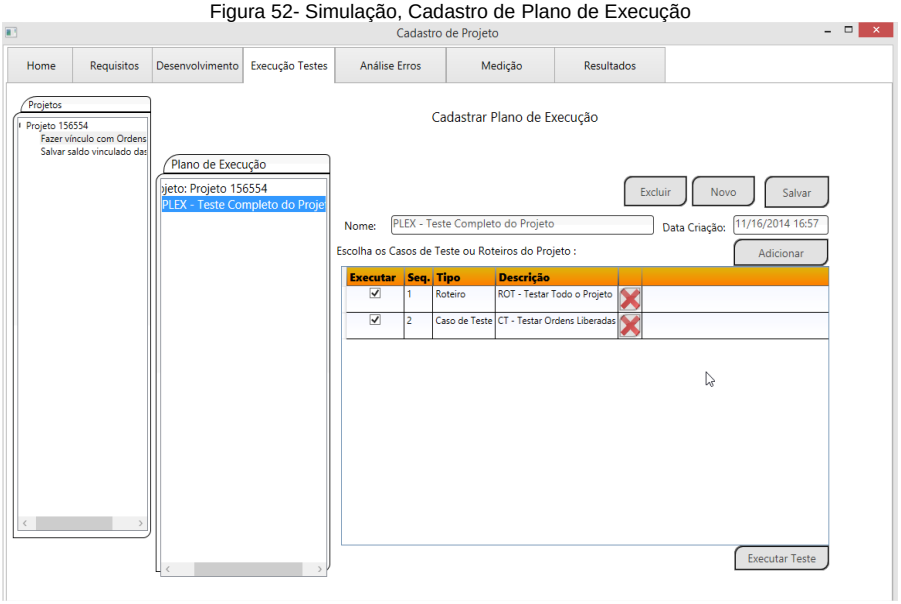
4.2.8 **Execução de Testes**

Para executar um teste primeiramente é necessário o cadastro de um plano de execução. Esse tem o objetivo de agrupar roteiros e casos de testes em uma única execução.

Na simulação do processo, figura 52, foram informados os dados com a descrição do plano de execução, adicionado um roteiro de testes conforme a figura 51 e um caso de teste. Pode-se notar que o caso de teste “CT – Testar Ordens

Liberadas” será executado duas vezes, pois o mesmo também está atrelado ao roteiro de teste adicionado.

Figura 52- Simulação, Cadastro de Plano de Execução

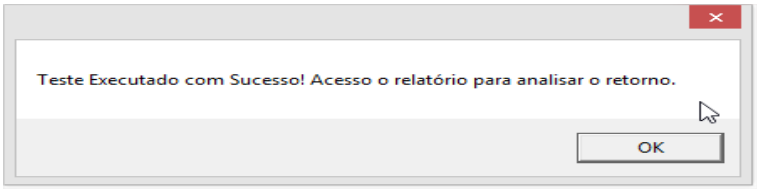


Fonte: O autor.

A execução do plano de teste selecionado inicia-se após acionar o botão Executar Teste. Quando acionado o mesmo, a ferramenta irá iniciar o módulo de execução de testes integrando com o *framework* utPLSQL. Conforme já explanado o *framework* irá executar o conjunto de scripts de testes (etapas de cada caso de testes) e retornar o resultado para a ferramenta.

A mensagem de retorno no caso de sucesso da execução pode ser visualizada na figura 53 e o resultado da execução poderá ser visualizado clicando na “Tab” Análise Erros.

Figura 53- Simulação, Sucesso de Execução do Plano de Execução



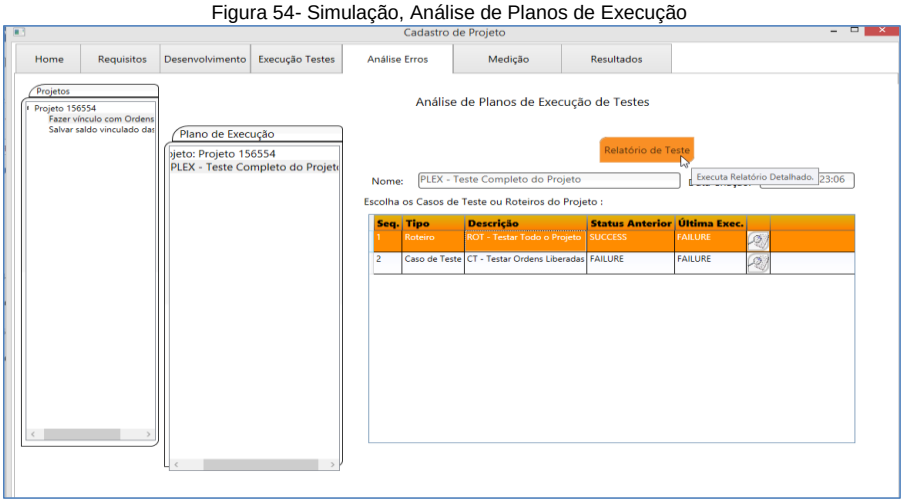
Fonte: O autor.

4.2.9 **Análise de Falhas**

Na análise de falhas é possível verificar quais etapas deram erro e por qual motivo e quais obtiveram sucesso. Essa etapa do fluxo da ferramenta e da simulação, verifica e valida as funcionalidades do caso de uso Analisar Testes.

Na primeira tela da análise de falhas exibida pela figura 54 e após a execução do plano de execução referenciado na figura 52, é possível evidenciar o status da última execução do teste e também do anterior.

Figura 54- Simulação, Análise de Planos de Execução



Fonte: O autor.

Pode-se verificar pela figura 54 que o teste do roteiro inicia com sucesso, retorna esse valor pois não havia nenhuma execução anterior. Já como status da última execução o roteiro consta com falha, pois conforme a parametrização das etapas para o caso de teste “CT – Testes Ordens Liberadas”, uma retornaria com sucesso, porque o item pode ou não estar vinculado e outra com erro.

A mesma situação ocorre com a sequência de execução 2, pois testa o mesmo caso de teste. Contudo essa execução já inicia com falha, dado que o mesmo caso de teste foi executado através do roteiro primeiramente.

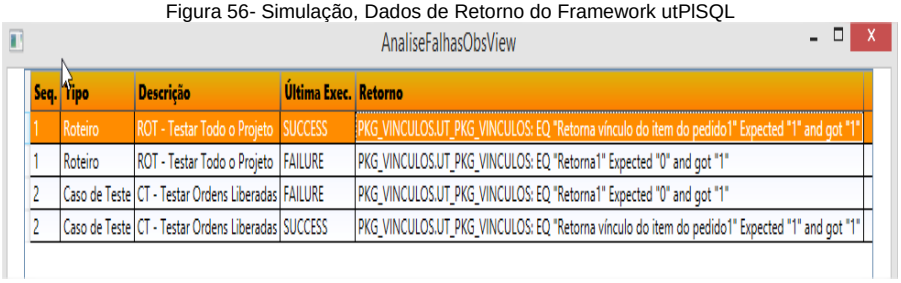
Para visualizar o motivo do erro é necessário acionar o botão no grid conforme figura 55. Esse botão irá abrir uma nova janela, contendo as informações explicativas da falha ou sucesso.

Figura 55- Simulação, Abrir Informações Sobre Execução dos Testes

Fonte: O autor.

A figura 56 demonstra o retorno da execução dos testes de forma detalhada, destacando: A rotina de execução; a *string* nome definida em cada etapa para identificação da mesma; O resultado esperado e encontrado.

Figura 56- Simulação, Dados de Retorno do Framework utPlSQL



Fonte: O autor.

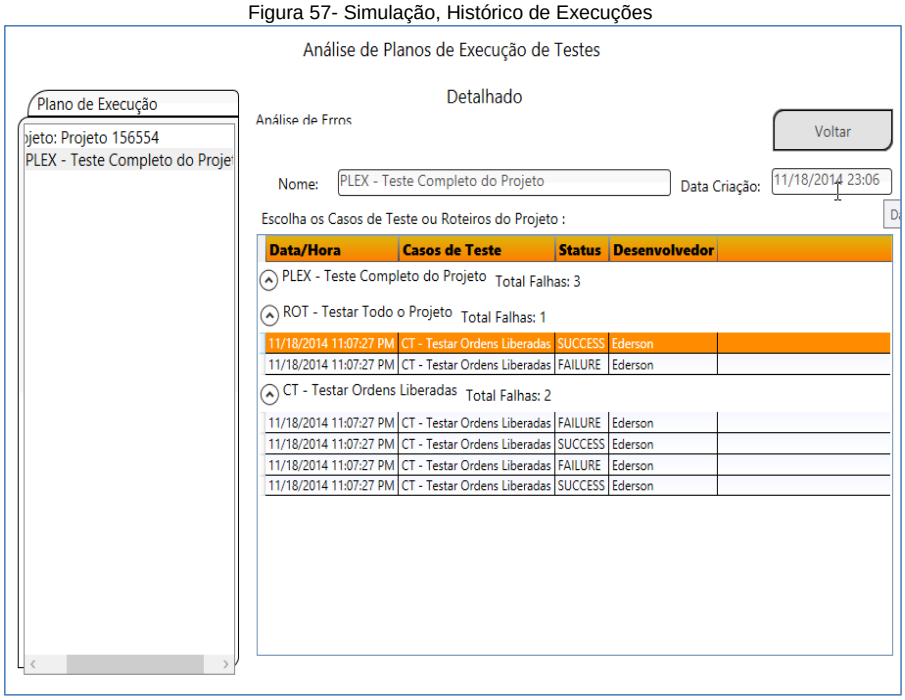
Com base na figura 56, conclui-se que o item está vinculado a uma ordem, portanto seu retorno é igual a 1. Como em uma das etapas esperava-se 0, ou seja não vinculado, o erro foi constatado pela ferramenta.

Todos estes dados estão disponíveis em tabela de dados da ferramenta (Postgres), tanto como na base de dados Oracle.

Sempre estará disponível para visualização, o histórico de execuções anteriores do plano de execução selecionado. Para visualiza-lo é necessário acionar o botão “Relatório de Testes”, de acordo com a figura 54.

A figura 57 apresenta o histórico das execuções, esse ainda demonstra a data e hora da execução, qual caso de teste envolvido, o status da execução e o desenvolvedor. Nessa mesma tela é possível identificar a quantidade total de falhas do plano de execução selecionado, agrupando por plano de execução, roteiros e casos de testes.

Figura 57- Simulação, Histórico de Execuções



Fonte: O autor.

Analisando a figura 57 é possível concluir que o caso de teste ficará com 4 execuções, pois o mesmo foi executado pelo roteiro também.

A próxima etapa da simulação é validar o caso de uso Medir Testes. Para acessar o mesmo é necessário selecionar a Tab. “Medição”.

4.2.10 **Medição de Testes**

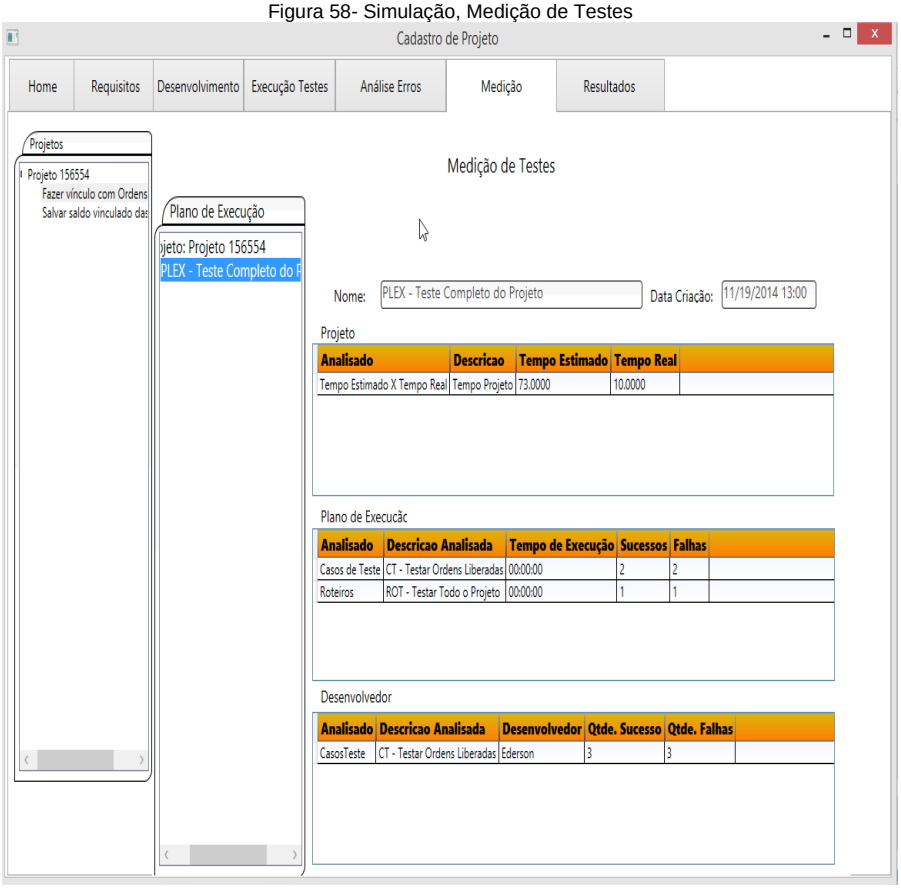
A medição de testes tem por função relatar ao usuário resumos de todo o projeto de testes. Sendo assim ela poderá ser útil em decisões gerenciais, como por exemplo tempo e dedicação da equipe voltada a testes.

A figura 58 apresenta a tela responsável por prover os dados da medição para:

a) Tempo estimado do projeto X Tempo efetivo.
b) Sequências dos planos de execução, contendo tempo de execução e totalizador de falhas e sucessos.

c) Desenvolvedor do caso de teste, contendo totalizador de falhas e sucessos.

Figura 58- Simulação, Medição de Testes



Fonte: O autor.

Nesta simulação selecionamos o plano de execução “PLEX – Teste Completo do Projeto” através da árvore de planos de execução. A ferramenta carregou os dados para cada item de medição.

Os dados do projeto são recuperados pelo cadastro do projeto. Já as informações do plano de execução e desenvolvedor, são retornados da análise de falhas.

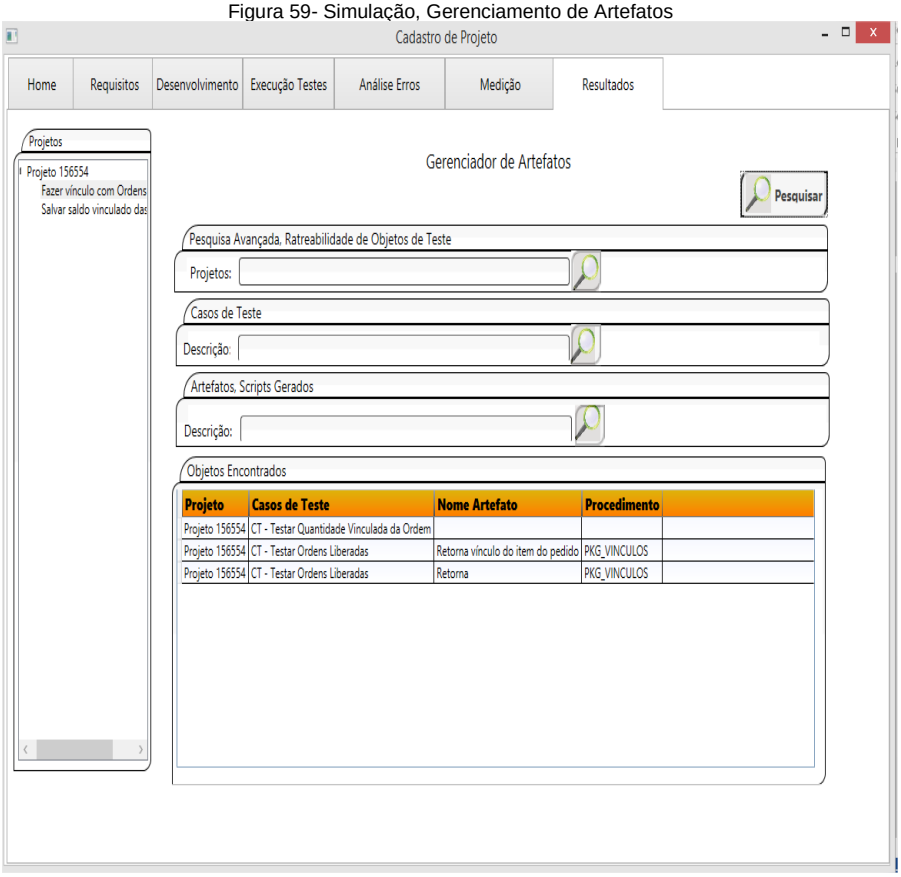
O próximo passo da simulação é atender ao caso de uso Gerenciamento de Configuração dos Artefatos de Teste. Para acessar o mesmo é necessário selecionar a Tab. “Resultados”.

4.2.11 **Resultados**

A Tab. “Resultados” possibilita ao usuário encontrar artefatos já criados para sua reutilização, ou até mesmo buscar casos de testes e projetos vinculados a um script de teste.

A figura 59 mostra a tela de gerenciamento de artefatos e a funcionalidades de seus campos são descritas na tabela 24.

Figura 59- Simulação, Gerenciamento de Artefatos



Fonte: O autor.

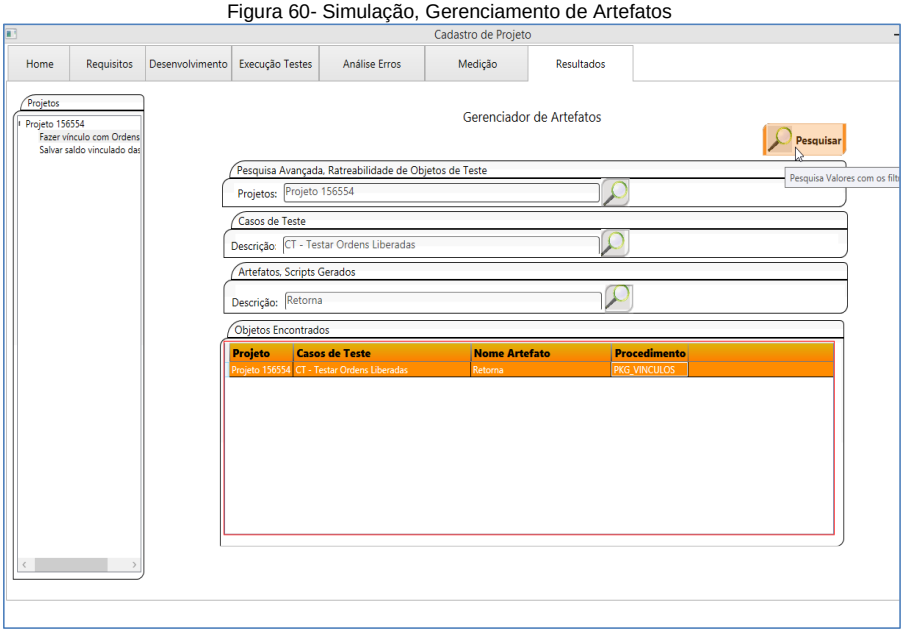
Tabela 24 – Simulação, Funcionalidades dos Campos de Resultados

Campo	Descrição	Finalidade
Projeto	Projetos cadastrados na ferramenta.	Filtrar por projetos.
Descrição Caso de Teste	Casos de Testes cadastrados na ferramenta.	Filtrar por descrição dos casos de testes.
Descrição, artefatos, scripts gerados	Nome da mensagem dos scripts de teste, do cadastro de etapas de teste.	Filtrar por nome dos scripts de testes.
Grid de Resultados	Apresentação dos dados após a execução da busca.	Retornar os projetos, casos de teste, nome de scripts e o procedimento da pesquisa.
Botão de Pesquisa	Todos os botões que possuam a imagem da lupa.	Abrir uma nova janela com um grid de pesquisa.

Fonte: O autor.

Foi realizada uma pesquisa utilizando todos os campos disponíveis para efeito da simulação. Através do botão de pesquisa é possível escolher os critérios para a ferramenta retornar as informações. A figura 60 ilustra o exemplo com os critérios escolhidos.

Figura 60- Simulação, Gerenciamento de Artefatos



Fonte: O Autor.

4.3 CONSIDERAÇÕES FINAIS

O presente capítulo apresentou uma simulação explicativa das funcionalidades da ferramenta AllInOneTest Automation, demonstrando em etapas as operações do usuário com o objetivo de validar os casos de uso.

Todos os casos de uso foram desenvolvidos, aplicando-se aos fluxos propostos da modelagem.

Além da simulação, em alguns dos tópicos foi citado situações de melhorias abordadas pelo usuário, essas representadas por tabelas. Porém, tais ajustes não poderão ser realizados neste trabalho ficando pendentes para a continuidade futura da ferramenta.

Usa-se o conceito de teste de unidade, em cada *script* salvo pela ferramenta e automatizado por ser possível a geração dos scripts e sua execução em forma de suítes de testes por um plano de execução.

Com a ferramenta foi possível gerar *scripts* de teste com facilidade e agilidade, além de documentar com maiores detalhes cada procedimento utilizado no desenvolvimento.

5 CONCLUSÃO

Com base na metodologia apresentada para o desenvolvimento e os objetivos propostos desse trabalho, conclui-se que os mesmos foram atingidos com sucesso. A ferramenta implementada e validada através de simulação, comprova que as funcionalidades requeridas pelos casos de uso foram atendidas de forma satisfatória.

Além disso, a ferramenta foi dividida em seu desenvolvimento na forma de projetos, podendo ser estendida para a criação de outros tipos de testes, não somente os PL/SQL, atingindo assim um dos objetivos da proposta.

Os testes de unidades puderam ser exercitados e criados rapidamente na geração dos *scripts* de teste, facilitando a integração com projetos de desenvolvimento. Esses testes são também armazenados na ferramenta garantindo o acesso e manipulação dos objetos de maneira simples e objetiva.

Os testes de regressão podem ser exercitados após a criação de novos projetos que alterem objetos antes criados. Ou seja, o teste sendo executado após uma nova alteração em um objeto já existente, já é considerado como teste de regressão.

Para aprimoramento futuro da ferramenta para testes de regressão é sugerida uma nova implementação. Sendo essa, a criação de um módulo específico para teste de regressão. Esse módulo manteria uma tabela de dados gerados para testes e a cada nova funcionalidade de um artefato, a comparação poderia ser realizada, validando se esse artefato mudou o comportamento dos métodos antigos.

Os testes com a linguagem PL/SQL foram validados pelo *framework* utPLSQL e no desenvolvimento sentiu-se dificuldade para a integração com o mesmo, pelo motivo no qual a única forma de retorno dos testes seja a consulta em tabelas do *framework*. Como uma nova sugestão de melhoria futura, o teste para a linguagem PL/SQL poderá ser realizado sem o uso do *framework* utPLSQ, podendo ser criado uma nova estrutura para testes e geração de scripts, acoplado ao projeto.

ANEXO A – ESTRUTURA DO FRAMEWORK UTPLSQL

```
CREATE OR REPLACE PROCEDURE calc_secs_between (
   date1 IN DATE,
   date2 IN DATE,
   secs OUT NUMBER)
IS
BEGIN
   -- 24 hours in a day,
   -- 60 minutes in an hour,
   -- 60 seconds in a minute...
   secs := (date2 - date1) * 24 * 60 * 60;
END;
/
```

```
CREATE OR REPLACE PACKAGE ut_calc_secs_between
IS
   PROCEDURE ut_setup;
   PROCEDURE ut_teardown;

   -- For each program to test...
   PROCEDURE ut_CALC_SECS_BETWEEN;
END ut_calc_secs_between;
/
CREATE OR REPLACE PACKAGE BODY ut_calc_secs_between
IS
   PROCEDURE ut_setup
   IS
   BEGIN
      NULL;
   END;

   PROCEDURE ut_teardown
   IS
   BEGIN
      NULL;
   END;

   -- For each program to test...
   PROCEDURE ut_CALC_SECS_BETWEEN IS
   BEGIN
      CALC_SECS_BETWEEN (
            DATE1 => ''
            ,
            DATE2 => ''
            ,
            SECS => ''
       );

      utAssert.this (
         'Test of CALC_SECS_BETWEEN',
         '<boolean expression>'
         );
   END ut_CALC_SECS_BETWEEN;
```

ANEXO B – EXEMPLO DO ARQUIVO DE STRING DE CONEXÃO

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <connectionStrings>
    <add name="StringConexaoPostGres"
         connectionString="Server=ALLINONEDATABASE;
                           Port=5432;
                           User Id = postgres;
                           Password = pacu;
                            database = allinonewpftest"/>
    <add name="StringConexaoOracle"
       connectionString="Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
                         (HOST=allinonedatabase)
                         (PORT=1521)))
                         (CONNECT_DATA=(SERVICE_NAME=xe)));
                         User Id=allinone;
                         Password=pacu"/>
  </connectionStrings>
</configuration>
```

Arquivo *AllInOneTestAutomation.exe.config*

REFERÊNCIAS

ASSESPRO – Associação das Empresas Brasileiras de Tecnologia da Informação. Disponível em:
< http://www.assespropr.org.br/programas-especiais/122-certificacao-de-qualidade-de-software.html> Acesso em 23 de mar. 2014, às 17:30.

COSTA, Mozart Guerra. Estratégia de Automação em Testes: Requisitos, Arquitetura e Acompanhamento de sua Implantação, São Paulo, 2004.

GUEDES, Gilleanes T. A. UML 2 Uma abordagem prática. 2. ed. São Paulo: Novatec Editora, 2011.

MARTIN, Robert C.; MARTIN, Micah. Princípios, Padrões e Práticas Ágeis em C#. Porto Alegre: Bookman, 2011. 736 p.

MEDEIROS, Ernani Sales. Desenvolvendo Software com UML 2.0. São Paulo: Pearson Makron Books, 2004. 259 p.

MOLINARI, Leonardo. Inovação e Automação de Testes de Software. São Paulo: Editora Érica Ltda., 2014. 140 p.

PFLEEGER, Shari Lawrence. Engenharia de software: teoria e prática / Shari Lawrence Pfleeger ; tradução Dino Franklin; revisão técnica Ana Regina Cavalcanti da Rocha, --2. Ed. – São Paulo: Prentice Hall, 2004.

PRESSMAN, Roger S., Egenharia de Software / Roger S. Pressamn; tradução Rosângela Delloso Penteado, revisão técnica Fernão Stella R. Germano, José Carlos Maldonato, Paulo Cesar Masiero. – 6.ed. – São Paulo: MCGraw-Hill,2006. 720 pp.

RIOS, Emerson; MOREIRA FILHO, Trayahú. Testes de Software 3° Edição revisada e atualizada. 3. ed. Rio de Janeiro: Alta Books, 2013. 304 p.

SOCOLOWSKI, Camila; ALARCON, André; ANTONIO, André Temple de. Flextest – um framework flexível para a automação de teste funcional de software. 2012. Disponível em: <http://www.imago.ufpr.br/csbc2012>Acesso em: 29 mar. 2014.

SOMMERVILLE, Ian. Engenharia de Software. 8. ed. São Paulo: Pearson Addison - Wesley, 2007. 551 p.

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. 521 p.

USP – Universidade de São Paulo. Disponível em:
<http://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf >
Acesso em 23 de mar. 2014, às 16:35.