|  |  |
| --- | --- |
|  | Logo fatec - 2016 - Denilce |

Coordenadoria de Tecnologia da Informação

Tecnologia em Análise e Desenvolvimento de Sistemas

HappyPath: Uma proposta de automação de testes utilizando Inteligência Artificial

Fernanda Pereira dos Santos Sousa

Sorocaba

Novembro – 2019

|  |  |
| --- | --- |
|  |  |
|  |  |

**Coordenadoria de Tecnologia da Informação**

Tecnologia em Análise e Desenvolvimento de Sistemas

HappyPath: Uma proposta de automação de testes utilizando Inteligência Artificial

Fernanda Pereira dos Santos Sousa

Profa. Dra. Maria das Graças Junqueira Machado Tomazela – Orientadora

Sorocaba

Novembro – 2019

**Dedicatória**

Dedico esse trabalho aos meus pais que sempre me apoiaram e incentivaram em todos os aspectos da minha vida. Dedico também a todos que contribuíram para que esse trabalho se tornasse realidade.

**AGRADECIMENTOS**

Primeiramente a Deus por todos os privilégios que recebi.

Aos meus pais Maria do Socorro e Marcílio Melo por sempre me apoiarem e incentivarem meus estudos.

Ao Maycon Douglas que me incentivou e ajudou durante toda a graduação, principalmente durante a árdua tarefa de escolha de tema para o Trabalho de Conclusão de Curso.

Ao Lucas Eduardo que auxiliou no refinamento da ideia possibilitando assim sua execução de fato, além de elucidar diversas dúvidas técnicas.

A minha orientadora Maria das Graças Junqueira Machado Tomazela, por toda a ajuda e paciência durante todo o período de criação do Trabalho de Graduação.

Ao corpo docente do curso de Análise e Desenvolvimento de Sistemas da FATEC Sorocaba por toda a base de conhecimento fornecida durante minha graduação.

Agradeço também a todos que contribuíram com ideias, sugestões e questionamentos, agregando valor ao trabalho e contribuindo para o resultado final.

*“Que nada nos defina, que nada nos sujeite. Que a liberdade seja a nossa própria substância, já que viver é ser livre.”*

*Simone de Beauvoir*

**Índice**

[Resumo 8](#_Toc25087839)

[Abstract 9](#_Toc25087840)

[1. Introdução 10](#_Toc25087841)

[2. Teste de *software* 12](#_Toc25087842)

[2.1. A Taxonomia das Técnicas de Teste 12](#_Toc25087843)

[2.2. O espectro de Teste 12](#_Toc25087844)

[2.2.1 Teste unitário 12](#_Toc25087845)

[2.2.2 Teste de integração 13](#_Toc25087846)

[2.2.3 Teste de Sistema 13](#_Toc25087847)

[2.2.4 Teste de Aceitação 13](#_Toc25087848)

[2.3. Análise Estática e Análise Dinâmica 14](#_Toc25087849)

[2.3.1 Análise Estática 14](#_Toc25087850)

[2.3.2 Análise Dinâmica 15](#_Toc25087851)

[2.4. Técnica Funcional e Técnica Estrutural 15](#_Toc25087852)

[2.4.1 Teste Funcional (Teste de Caixa Preta) 16](#_Toc25087853)

[2.4.2 Teste Estrutural (Teste de Caixa Branca) 17](#_Toc25087854)

[3. Inteligência Artificial 18](#_Toc25087855)

[3.1. Conceitos de Inteligência Artificial 18](#_Toc25087856)

[3.2. Técnicas de IA 18](#_Toc25087857)

[3.2.1 Raciocínio Baseado em Casos 19](#_Toc25087858)

[3.2.2 Redes neurais 20](#_Toc25087859)

[3.2.3 Algoritmos genéticos 22](#_Toc25087860)

[3.2.4 Lógica Fuzzy 24](#_Toc25087861)

[3.3 Regras de produção 25](#_Toc25087862)

[4. Trabalhos relacionados 29](#_Toc25087863)

[5. Metodologia 30](#_Toc25087864)

[5.1. Natureza da pesquisa 30](#_Toc25087865)

[5.1.1 Experimento de pesquisa 30](#_Toc25087866)

[6. Proposta de modelo de teste automatizado 32](#_Toc25087867)

[6.1. Utilização de regras de produção 32](#_Toc25087868)

[6.2. Aplicação HappyPath 37](#_Toc25087869)

[6.2.1. Telas 37](#_Toc25087870)

[6.2.2. Armazenamento de Dados 53](#_Toc25087871)

[7. Considerações finais 55](#_Toc25087872)

[Bibliografia 57](#_Toc25087873)

[Glossário 61](#_Toc25087874)

**Lista de Figuras**

[Figura 1 - Fluxo de informação de teste 16](#_Toc25087875)

[Figura 2 - Ciclo de RBC 19](#_Toc25087876)

[Figura 3 - Função de ativação 21](#_Toc25087877)

[Figura 4 - Saída expressa matematicamente 22](#_Toc25087878)

[Figura 5 - Expressão utilizando bias 22](#_Toc25087879)

[Figura 6 - Modelo baseado em regras Fuzzy. 24](#_Toc25087880)

[Figura 7 - Equação da lógica Fuzzy 25](#_Toc25087881)

[Figura 8 - Funcionamento de um sistema baseado em Regras de Produção 27](#_Toc25087882)

[Figura 9 - Funcionamento da Inteligência Artificial na proposta 36](#_Toc25087883)

[Figura 10 - Tela de criação de testes – Primeiro passo 38](#_Toc25087884)

[Figura 11 - Tela de criação de testes – Adição de passo 39](#_Toc25087885)

[Figura 12 - Modal de inclusão de Ponto de Partida 40](#_Toc25087886)

[Figura 13 - Tela Condições Iniciais / Resultados esperados 42](#_Toc25087887)

[Figura 14 - Modal de criação de nova Condição Inicial / Resultado Esperado 43](#_Toc25087888)

[Figura 15 - Tela Ações 45](#_Toc25087889)

[Figura 16 - Modal de criação de nova Ação 46](#_Toc25087890)

[Figura 17 - Tela Criação de Caso de Teste preenchida 48](#_Toc25087891)

[Figura 18 - Tela Casos de Testes 49](#_Toc25087892)

[Figura 19 - Modal Conversor de Casos de Teste 50](#_Toc25087893)

[Figura 20 - Janela Pop-up de Alerta 52](#_Toc25087894)

[Figura 21 - Diagrama Entidade Relacionamento 54](#_Toc25087895)

**Lista de Quadros**

[Quadro 1 - Caso de Teste Abas Principais 33](#_Toc25087896)

[Quadro 2 - CT Opções da Página Administração 33](#_Toc25087897)

[Quadro 3 - Caso de Teste Criação de Requisição de Acesso 34](#_Toc25087898)

[Quadro 4 - Caso de teste Criação de Requisição de Acesso gerado após ação da Inteligência Artificial 35](#_Toc25087899)

**HappyPath: Uma proposta de automação de testes utilizando Inteligência Artificial**

# Resumo

As novas metodologias de desenvolvimento de software vêm ganhando cada dia mais força no meio empresarial, uma vez que são capazes de agregar valor ao produto a cada entrega, trazendo sempre novas funcionalidades ao sistema do cliente. Entretanto, isso exige um maior esforço de teste, pois para garantir a integração entre os módulos e o pleno funcionamento da aplicação, testes de regressão são necessários em cada entrega, tornando o teste repetitivo e mais extenso em cada iteração. Assim, o objetivo deste trabalho foi propor uma nova metodologia de teste que utilizasse mecanismos inteligentes para auxiliar a geração de casos de teste automatizados. Por meio de uma proposta teórica, elaborou-se uma metodologia capaz de unir fragmentos de código de testes automatizados e empregar algoritmos inteligentes que possam traçar passos implícitos no Caso de Teste ao analisar o conteúdo já existente em uma base de conhecimento. Com a metodologia elaborada neste trabalho será possível o desenvolvimento de um software que implemente um mecanismo de teste automatizado, baseado em Inteligência Artificial.

Palavras-chave: Testes automatizados, Inteligência artificial, Casos de teste

**HappyPath: A proposal for test automation using Artificial Intelligence**

# Abstract

New software development methodologies are increasingly gaining strength in the business environment, as they can add value to the product with each delivery, always bringing new features to the customer system. However, this requires more testing effort, to ensure integration between modules and full application operation, regression tests are required on each delivery, making the test repetitive and more extensive on each iteration. Thus, the objective of this undergraduate thesis was to propose a new test methodology that uses intelligent mechanisms to assist the generation of automated test cases. Through a theoretical proposal, a methodology capable of assembling automated test code fragments and apply intelligent algorithms that can trace implicit steps in the Test Case by analyzing existing content in a knowledge base was developed. With the methodology elaborated in this undergraduate thesis, it will be possible to develop software that implements an automated test mechanism, based on Artificial Intelligence.

Keywords: Artificial intelligence, Automated testing, Test cases

# 1. Introdução

O cenário empresarial e tecnológico atual mostra uma grande quantidade de aplicativos e aplicações *web* sendo requisitadas e liberadas para consumo diariamente, isso gera uma grande carga de trabalho, não só para desenvolvedores como para testadores, uma vez que as atividades de testes, muitas vezes, são mais morosas que o próprio desenvolvimento.

Ao observar as metodologias usadas no desenvolvimento, percebe-se, por exemplo, que na metodologia de desenvolvimento de *software* em Cascata, a maior parte do teste será executada ao final de todo o período de desenvolvimento, gerando uma altíssima carga de testes. Já na metodologia Ágil o teste é feito em pequenos blocos, assim como o desenvolvimento, diminuindo sua carga e aumentando sua frequência, exigindo um maior número de testes de integração, repetindo várias e várias vezes o mesmo tipo de teste.

Com essa repetição, surgiram os mecanismos de automação de testes, que, ao utilizar *frameworks* próprios e linguagens de programação já usadas por desenvolvedores, são capazes de executar diversas vezes e automaticamente um mesmo Caso de Teste (CT). Porém, essa automação deve ser inteiramente construída pelo testador para cada CT criado, o que pode levar muito tempo e exigir bastante esforço.

A partir desse contexto, a pergunta de pesquisa que se coloca é:

* Como um mecanismo de Inteligência Artificial pode ser utilizado na criação de teste de *software* automatizado?

Dessa maneira, este trabalho teve como objetivo propor uma metodologia de criação de casos de testes utilizando um mecanismo de Inteligência Artificial.

Visando a atingir o objetivo declarado, uma pesquisa teórica foi realizada, analisando monografias, dissertações e artigos da área estudada, dessa forma aumentando o conhecimento específico e verificando o estado da arte.

Como maneira de ilustrar a metodologia proposta, este trabalho apresenta um exemplo de *software*, HappyPath, visando a facilitar o entendimento geral do funcionamento interno da metodologia.

Esse trabalho se justifica pela agilidade que pode prover ao processo de teste, facilitando assim o trabalho dos testadores em projetos com grande carga de testes.

Este trabalho está organizado em 7 capítulos. O Capítulo 1 é destinado a apresentar a condição atual do mercado, objetivos do trabalho e sua estrutura. O Capítulo 2 visa a conceituar testes de *software*. O Capítulo 3 tem como objetivo explicar conceitos e definições relacionadas à Inteligência Artificial. O Capítulo 4 traz trabalhos relacionados, mostrando o estado da arte do tema em estudo. O Capítulo 5 trata-se da metodologia aplicada no trabalho. O Capítulo 6 é destinado a especificação do funcionamento da metodologia, trazendo a proposta do *software* HappyPath, como forma de ilustrá-la. O Capítulo 7 apresenta as conclusões finais do trabalho.

# 2. Teste de *software*

## 2.1. A Taxonomia das Técnicas de Teste

Segundo Luo (2001), teste de *software* é uma área muito abrangente, que envolve várias outras áreas, técnicas e não-técnicas, como especificação, *design* e implementação, manutenção, processamento e gerência de problemas em engenharia de *software*.

Em diferentes publicações, a definição de teste varia de acordo com o propósito, processo e nível de teste definido (LUO, 2001). Uma boa discrição de teste é dada por Miller (1981): O objetivo geral dos testes é afirmar a qualidade dos sistemas de *software*, exercitando-os sistematicamente em circunstâncias cuidadosamente controladas. A descrição de teste de Miller (1981) vê a maioria das atividades de garantia de qualidade de *software* como teste.

Segundo Myers, Badgett e Sandler (2012) o teste deve ter como maior intenção encontrar erros. Um bom teste é aquele que tem uma grande probabilidade de encontrar um erro ainda não descoberto e, um teste bem-sucedido é aquele que o encontra.

## 2.2. O espectro de Teste

As práticas de teste estão envolvidas em todo o ciclo de vida do *software*, porém, o teste feito em cada nível de desenvolvimento de *software* é diferente em sua natureza e têm diferentes objetivos (LUO, 2001).

### 2.2.1 Teste unitário

O teste unitário é feito na unidade básica do *software*, que é o menor pedaço testável de um *software*, também chamado de “unidade”, “módulo” ou “componente” de forma intercambiável (LUO, 2001).

Os testes de unidade, segundo Myers, Badgett e Sandler (2012) são testes de falha, já que são desenvolvidos para apontar os problemas encontrados assim que o código é desenvolvido.

### 2.2.2 Teste de integração

Segundo Luo (2001), o teste de integração é performado quando duas ou mais unidades de testes são combinadas dentro de uma estrutura maior. O teste geralmente é realizado nas interfaces entre os componentes e na estrutura maior que está sendo construída, se sua propriedade de qualidade não puder ser avaliada a partir de seus componentes.

Já Orso (1998) diz que o teste de unidades individuais ajuda a remover falhas locais, mas não exerce as interações entre unidades diferentes e que o teste de integração é a atividade de exercer essas interações reunindo os diferentes módulos que compõem um sistema. Segundo Orso (1998) esse tipo de teste caracteriza-se por envolver diferentes unidades de interação que, em geral, foram desenvolvidas por diferentes programadores.

### 2.2.3 Teste de Sistema

Luo (2001) define teste de sistema como um teste que tende a afirmar a qualidade de ponta a ponta de todo o sistema. O teste do sistema geralmente é baseado na especificação funcional / de requisitos do sistema. Atributos de qualidade não funcionais, como confiabilidade, segurança e capacidade de manutenção, também são verificados.

Briand e Labiche (2002) também seguem essa linha definindo-o como uma técnica voltada a testar o sistema inteiro baseado em suas especificações e envolvendo diversas atividades como teste funcional (teste a partir de descrições comportamentais do sistema) e teste de performance (tempo de resposta e utilização de recursos), ou seja, a implementação sob teste é comparada com sua especificação pretendida.

### 2.2.4 Teste de Aceitação

O teste de aceitação é realizado quando o sistema concluído é entregue pelos desenvolvedores aos clientes ou usuários. O objetivo do teste de aceitação é fornecer confiança de que o sistema está funcionando, e não encontrar erros (LUO, 2001).

Myers, Badgett e Sandler (2012) definem que teste de aceitação consiste em comparar um sistema de *software* aos seus requisitos iniciais e às necessidades atuais de seus usuários finais ou, no caso de um programa contratado, ao contrato original.

## 2.3. Análise Estática e Análise Dinâmica

Com base na necessidade ou não da execução real do *software* em avaliação, existem duas categorias principais de atividades de garantia de qualidade, o teste estático (análise estática) e o teste dinâmico (análise dinâmica) (LUO, 2001).

De maneira geral os métodos estáticos e dinâmicos são inseparáveis, porém, podem ser discutidos separadamente. Nesse trabalho, ao se referir a teste subentende-se como teste dinâmico já que as atividades de citadas requerem execução de um *software*.

### 2.3.1 Análise Estática

Luo (2001) escreve que a análise estática se concentra na variedade de métodos usados para determinar ou estimar a qualidade do *software* sem referência às execuções reais. As técnicas nessa área incluem inspeção de código, análise de programa, análise simbólica e verificação de modelo.

Sommerville et al. (2008) também a definem como um teste sem a execução do programa, avaliando seus artefatos, tais quais como diagramas, documentos de requisitos, código fonte entre outros. Mendonça (2014) afirma que mesmo não sendo possível assegurar por meio da análise estática que um produto está suficiente seguro para ser utilizado, principalmente porque não se avalia o aspecto comportamental do produto em análise, esse tipo de teste é uma excelente técnica de verificação e cita Pressman (2011) ao afirmar que as revisões são um filtro para a gestão de qualidade.

### 2.3.2 Análise Dinâmica

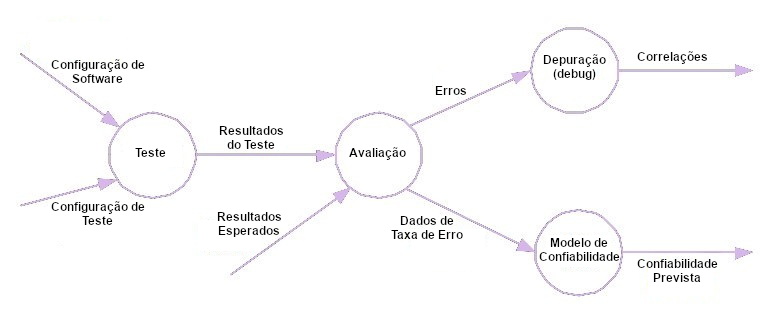
A análise dinâmica lida com métodos específicos para determinar e / ou aproximar a qualidade do *software* por meio de execuções reais, isto é, com dados reais e em circunstâncias reais (ou simuladas). As técnicas nessa área incluem a síntese de insumos, o uso de procedimentos de teste baseados na estrutura do *software* e a automação da geração do ambiente de teste (LUO, 2001).

Mendonça (2014) a define como uma avaliação do comportamento de um produto por meio de sua execução ao se utilizar casos de teste (conjunto de entradas e saídas esperadas do sistema).

## 2.4. Técnica Funcional e Técnica Estrutural

Luo (2001) define que teste envolve a configuração de entradas apropriadas, execução do *software* sobre a entrada e análise da saída. A "Configuração de *software*" inclui especificação de requisitos, especificação de *design*, código fonte e assim por diante como pode ser analisado na figura 1. A "Configuração de Teste" inclui casos de teste, plano e procedimentos de teste e ferramentas de teste.

Figura 1 - Fluxo de informação de teste

*Fonte: Adaptado de Luo (2001)*

Com base no fluxo de informações de teste, uma técnica de teste especifica a estratégia usada no teste para selecionar casos de teste de entrada e analisar os resultados do teste. Técnicas diferentes revelam aspectos diferentes da qualidade de um sistema de *software* e existem duas categorias principais de técnicas de teste, funcionais e estruturais (LUO, 2001).

### 2.4.1 Teste Funcional (Teste de Caixa Preta)

No teste funcional o programa ou sistema de *software* em teste é visto como uma "caixa preta". A seleção de casos de teste para teste funcional é baseada no requisito ou na especificação de *design* da entidade de *software* em teste. Exemplos de resultados esperados incluem especificações de requisitos / *design*, valores calculados manualmente e resultados simulados. O teste funcional enfatiza o comportamento externo da entidade de *software*. (LUO, 2001).

Segundo Mitra, Chatterjee e Ali (2011) essa técnica projeta casos de teste com base nas informações da especificação, o testador de *software* não deve ter acesso ao código-fonte interno e sim focar sua análise nas saídas apresentadas pelo sistema sob teste ao inserir as entradas necessárias, baseando-se nos requerimentos e especificações do *software*.

### 2.4.2 Teste Estrutural (Teste de Caixa Branca)

O teste estrutural é definido por Luo (2001) como se a entidade do *software* fosse vista como uma "caixa branca". A seleção de casos de teste é baseada na implementação da entidade de software. O objetivo de selecionar esses casos de teste é causar a execução de pontos específicos na entidade de *software*, como instruções específicas, ramificações ou caminhos do programa. Os resultados esperados são avaliados em um conjunto de critérios de cobertura. Exemplos de critérios de cobertura incluem cobertura de caminho, cobertura de *branch* e cobertura de fluxo de dados. O teste estrutural enfatiza a estrutura interna da entidade de *software*.

O testador que usa essa técnica (geralmente o desenvolvedor da aplicação) sabe como o código é e escreve testes executando métodos com certos parâmetros. O teste de caixa branca é focado no funcionamento interno do sistema, seu fluxo de controle ou seu fluxo de dados (SHAO; KHURSHID; PERRY, 2007, LIU; TAN, 2009, SAGLIETTI; OSTER; PINTE, 2008). Diversas pesquisas ressaltam que os testes de caixa preta e caixa branca são complementares entre si, que para um teste mais correto é essencial cobrir tanto as especificações quanto as ações do código (LIU; TAN, 2009, SHAW, 2002, JORGENSEN, 2002).

# 3. Inteligência Artificial

Existem diversas definições para o termo Inteligência Artificial, diversos autores dissertam sobre seu significado, funções e importância na sociedade.

A Inteligência Artificial pode ser definida, segundo Luger (2009), como o ramo da ciência da computação que diz respeito à automação do comportamento inteligente. O autor ressalta ainda que a IA sempre se preocupou mais em expandir as capacidades da ciência da computação do que em definir seus limites. Castrounis (2019) corrobora essa afirmação ao defini-la como um conjunto de conceitos e ferramentas que representam o verdadeiro significado de inovação e com um imenso potencial disruptivo e transformativo.

## 3.1. Conceitos de Inteligência Artificial

Como descrevem Neapolitan e Jiang (2013), é possível conceituar uma entidade artificial como inteligente usando o teste de Turing.

Alan Turing, em 1950, desenvolveu um teste operacional e empírico de inteligência artificial, que indica que um humano interrogador deve ser colocado em uma sala, um segundo ser humano colocado em uma outra sala e uma entidade artificial colocada em uma terceira. O interrogador poderia se comunicar com ambos apenas por um dispositivo textual como um terminal. Após algumas perguntas deve ser pedido para o interrogador distinguir o ser humano da entidade artificial baseando-se nas respostas adquiridas, caso o interrogador não consiga fazer isso o teste de Turing é passado e pode-se dizer que a entidade artificial é inteligente (NEAPOLITAN; JIANG, 2013).

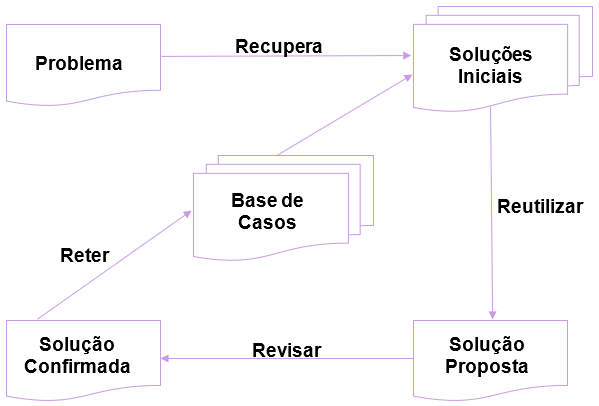
## 3.2. Técnicas de IA

Ao longo dos anos, diversas técnicas de Inteligência Artificial foram criadas e aprimoradas para diferentes usos, nas sessões a seguir seguem algumas das mais citadas na literatura (CASTROUNIS, 2019, LUGER, 2009, NEAPOLITAN; JIANG, 2013, MONARD; BARANAUKAS, 2003).

### 3.2.1 Raciocínio Baseado em Casos

A técnica Raciocínio Baseado em Casos (RBC) trata-se de uma abordagem voltada para resolução de problemas de maneira empírica ao se propor uma solução para um novo problema baseando-se em como um problema similar passado foi resolvido (VON WANGENHEIM CG, VON WANGENHEIM A, 2003). É um paradigma de Inteligência Artificial, que apresenta uma forma de representar e armazenar a experiência por meio de casos. O caso representa além do conteúdo da experiência, o contexto na qual este se passou (MIRANDA; SANTOS, 2005). É possível visualizar seu ciclo de funcionamento na figura 2.

Figura 2 - Ciclo de RBC



*Adaptado de Fernandes (2005) Apud Lee (1998).*

Os casos são utilizados ao se apresentar um novo problema; então uma recuperação na base de casos é feita no intuito de recuperar um caso que se ajuste com o novo, de acordo com as questões que sejam mais relevantes, avaliando os casos recuperados em relação à aplicabilidade para o novo problema; nesse processo é possível que haja necessidade de adaptação do caso recuperado, com a finalidade de ajuste para a solução; a solução do novo caso, já ajustada, pode então ser armazenada na base (memória de casos) juntamente com a descrição de sucessos e falhas (FERNANDES, 2005).

### 3.2.2 Redes neurais

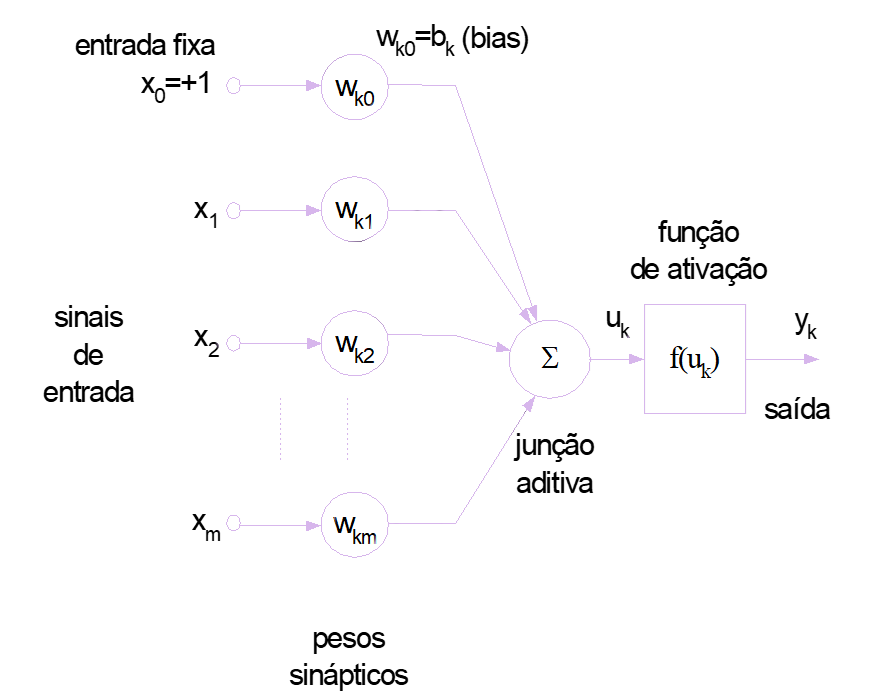
As Redes Neurais são compostas por “neurônios” ou “unidades de processamento”, células computacionais simples, que são interligadas. Elas se assemelham ao cérebro em dois aspectos (HAYKIN, 2001):

1 – O conhecimento é adquirido pela rede a partir do seu ambiente por meio de um processo de aprendizagem.

2 – Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.”

Para algoritmos de redes neurais devem ser definidos parâmetros tais como: número de camadas da rede, número de neurônios por camada, tipo de conexões (*forward*, *backward* ou lateral) e grau de conectividade entre os neurônios (um a um, total, randômica, etc.). Os neurônios artificiais têm como componentes as sinapses (entradas), com seus pesos associados, a junção somadora e a função de ativação (FERNANDES, 2005), como pode ser visualizada na figura 3.

Figura 3 - Função de ativação

*Fonte: Adaptado de Haykin (2001)*

A operação de um neurônio artificial se resume em apresentar sinais à entrada (x1 a xm), multiplicar cada sinal por um peso que indica sua influência na saída da unidade (wk), executar uma soma ponderada dos sinais para produzir um nível de atividade (uk), usar a função de ativação f(uk) que tem a função de limitar a saída e introduzir não-linearidade ao modelo e utilizar o *bias* *b*k que tem o papel de aumentar ou diminuir a influência do valor das entradas (é possível considerar o *bias* como uma entrada de valor constante 1, multiplicado por um peso igual a *b*k (MIRANDA, 2005).

O neurônio artificial pode ter sua saída representada matematicamente como ilustra a figura 4, ou, considerando o *bias* como entrada de valor *x0=1* e peso *wk0=bk*, como mostrado na figura 5.

Figura 4 - Saída expressa matematicamente

  
*Fonte: Adaptado de Haykin (2001)*

Figura 5 - Expressão utilizando bias

  
*Fonte: Adaptado de Haykin (2001)*

As camadas são geralmente classificadas em três grupos: 1 - Camada de Entrada: onde os padrões são apresentados à rede; 2 - Camadas Intermediárias ou Ocultas: onde é feita a maior parte do processamento, por meio das conexões ponderadas - podem ser consideradas como extratoras de características; 3 - Camada de Saída: onde o resultado final é concluído e apresentado (FERNANDES, 2005).

A propriedade mais importante das redes neurais é a habilidade de aprender a partir de seu ambiente e com isso melhorar seu desempenho, isso é feito por meio de treinamento, um processo iterativo de ajustes aplicado a seus pesos. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas (RUSSELL; NORVIG, 2003).

### 3.2.3 Algoritmos genéticos

Algoritmos genéticos é uma técnica da Inteligência Artificial criada ao observar a teoria da evolução proposta por Darwin. Em um conjunto de potenciais soluções para um problema, as melhores seriam selecionadas até que a melhor seja escolhida como solução (GESTAL et al., 2010).

Essa técnica se enquadra na computação evolutiva, que segundo Gestal et al. (2010) pode ser definida, de maneira geral, como uma família de modelos computacionais inspirados na evolução, ou mais formalmente, como o estudo dos fundamentos e aplicações de certas técnicas heurísticas baseadas nos princípios da evolução natural.

O algoritmo tem início com um conjunto de soluções (representado por cromossomos) gerado aleatoriamente, chamado população. Em seguida as soluções dessa população são escolhidas e usadas para formar uma nova população (reprodução) – é esperado que a nova população seja “melhor” que a anterior, as soluções que são escolhidas para formar novas soluções (descendentes) são escolhidas de acordo com uma função de adaptação (função objetivo - custo) e o processo é repetido até que uma condição seja satisfeita, Fernandes (2005) apresenta esse funcionamento com mais detalhes no algoritmo apresentado na sequência.

*Início* {Algoritmo Genético Simples}

* Gerar uma população inicial;
* Calcular a função de avaliação de cada indivíduo;

*Enquanto não* (terminou) *faça*

*Início* {produz nova geração}

*Para* (tamanho da população) / 2 *faça*

*Início* {Ciclo reprodutivo}

**Selecionar** dois indivíduos da geração anterior para cruzamento (probabilidade de seleção proporcional à função de avaliação do indivíduo);

**Cruzar** com certa probabilidade obtendo os descendentes;

**Mutação** nos descendentes com certa probabilidade;

**Calcular** a função de avaliação dos descendentes que sofreram mutação;

**Incluir** os descendentes que sofreram mutação na nova geração;

*Fim-para*

*Se* (população convergiu) *então*

Terminou := verdade;

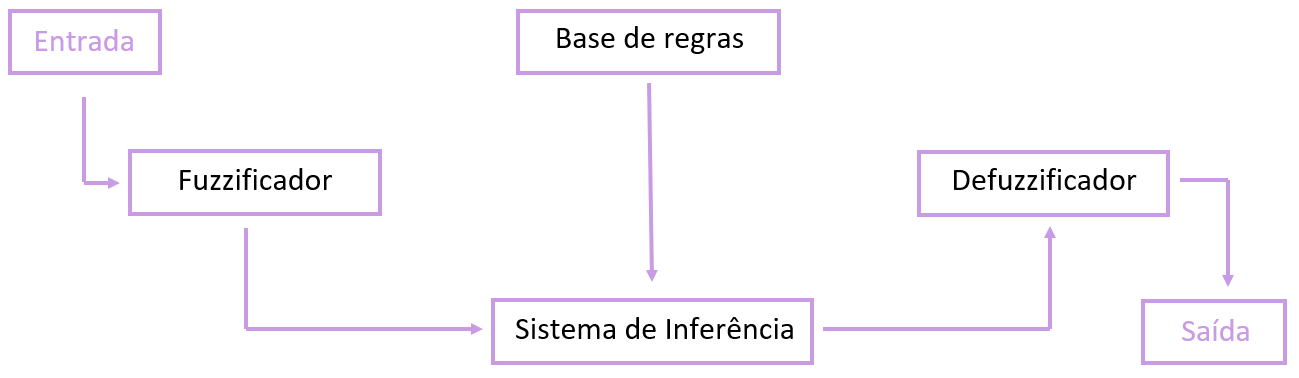
*Fim do Algoritmo*

### 3.2.4 Lógica Fuzzy

A teoria dos conjuntos *fuzzy* é uma extensão da teoria clássica de conjuntos (que trabalha apenas com grau de pertinência binária - 0 ou 1, verdadeiro ou falso). Essa teoria atribui graus intermediários de pertinência, entre 0 e 1, a cada um dos elementos dos seus conjuntos. Os modelos de inferência *fuzzy* são capazes de processar informações imprecisas e qualitativas e são adequados em processos que exigem tomadas de decisão (ALMEIDA, 2006).

É possível verificar seu fluxo de funcionamento na figura 6, em que a “Entrada” recebe valores numéricos para as Variáveis de Entrada (), o “Fuzzificador’ transforma os valores numéricos das variáveis em conjuntos *Fuzzy*, a “Base de Regras” possui o conjunto de Regras *Fuzzy* juntamente com a base de conhecimento, ou seja, conhecimento de especialistas, dados experimentais, dados numéricos, etc., o “Sistema de Inferência” aplica um raciocínio *Fuzzy* para obter uma saída *Fuzzy*, o “Defuzzificador” transforma a saída *fuzzy* em uma “Saída” numérica (PEDRYCZ; GOMIDE, 1998).

Figura 6 - Modelo baseado em regras Fuzzy.



*Fonte: Adaptado de* *Pedrycz e Gomide (1998)*

Para a representação *fuzzy* do conhecimento impreciso utilizam-se variáveis linguísticas, que admitem como valores expressões linguísticas, também denominadas de termos primários, como “muito grande”, “ruim”, “muito ruim”. Os termos primários de cada variável linguística formam a partição *fuzzy* destas variáveis (ALMEIDA, 2006).

Segundo Pedrycz e Gomide (1998), formalmente, dado um conjunto universo U, A é considerado um subconjunto *fuzzy* de U se A é representada por um conjunto de pares ordenados como os que aparecem na figura 7.

Figura 7 - Equação da lógica Fuzzy



*Adaptado de Pedrycz e Gomide (1998)*

Onde é a função de pertinência que determina com que grau .

De acordo com Pedrycz e Gomide (1998), qualquer função que associe valores entre zero e um aos elementos de um dado conjunto pode ser tomada com função de pertinência, entretanto, uma função de pertinência deve levar em conta o contexto em que esta será utilizada. Segundo Bardossy e Duckstein (1995), as regras *fuzzy* são declarações condicionais envolvendo relações entre entradas e saídas. Essas variáveis são conectadas por meio de operadores lógicos como E, OU e ENTÃO.

## 3.3 Regras de produção

Regras de produção são uma representação do conhecimento baseada nas propostas do matemático Emil Post (1943) como cita Heinzle (1995) que apresentavam os sistemas de produção como um modelo computacional para a solução de problemas, que mais tarde, na década de oitenta, passou a ser utilizada como suporte para o modelo mental (HEINZLE, 1995).

Segundo Heinzle (1995), o termo “sistema de produção” pode ser usado para definir sistemas constituídos de uma coleção de regras para descrever condições e ações, regras essas que são armazenadas como um conjunto de declarações SE-ENTÃO

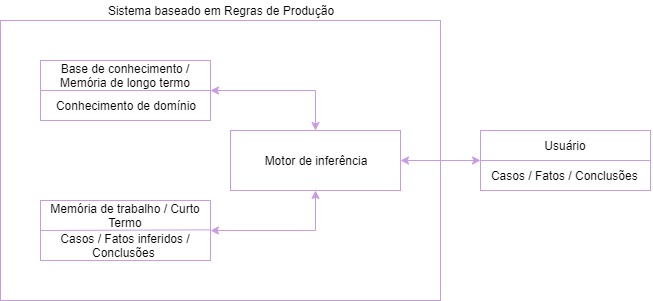
SE < premissas > ENTÃO <conclusão>

O corpo da regra delimita-se pela palavra SE, localizada à esquerda, antecedendo a premissa ou condição e deve ser avaliada em relação à base de conhecimentos como um todo (HEINZLE, 1995).

A base de conhecimentos pode ser definida, segundo Heinzle (1995), como um depósito de conhecimento acerca de um determinado assunto, inseridos por um especialista por meio de um sistema e armazenado por esse de maneira própria, permitindo posterior processamento e inferência.

A base de conhecimentos armazena suas informações de maneira estática, portanto se faz necessário o uso de uma força externa para analisá-las e processá-las para assim tirar proveito de seu conteúdo, para isso é usado um motor, máquina, ou ainda, engenho de inferência que é capaz de buscar na base o conhecimento necessário a ser avaliado em cada situação, direcionar o processo de raciocínio, gerenciar situações de incerteza e levar ao resultado final (HEINZLE, 1995), auxiliado sempre pela memória de trabalho, onde se é armazenada as informações usadas durante o processo de inferência, como pode ser observado na figura 8.

Figura 8 - Funcionamento de um sistema baseado em Regras de Produção

  
*Fonte: Adaptado de Russell e Norvig (2003)*

Para a localização das regras é necessário o uso de um mecanismo de busca, sendo a mais comumente usada a árvore de decisão que segundo Monard e Baranaukas (2003) é uma estrutura de dados definida recursivamente, sendo um nó folha correspondente a uma classe ou um nó de decisão.

Nesse mecanismo é possível observar dois métodos de encadeamento, o progressivo (*forward*) e o regressivo (*backward*), sendo o progressivo iniciado com a seleção de uma regra qualquer, ou uma determinada de acordo com o sistema que a utiliza, então a regra é analisada e se procura por uma outra regra que tenha entre as suas premissas a conclusão da regra avaliada. Já o encadeamento regressivo também se inicia ao pegar uma regra aleatoriamente ou a definida no sistema e analisá-la, porém, se procura a premissa da regra avaliada na conclusão de uma outra regra.

Quando a condição ou premissa buscada pelo sistema é encontrada na base de conhecimento a ação correspondente a ela, especificada no lado direito, consequente à condição, é executada. Se qualquer premissa falhar a ação correspondente também falha (HEINZLE, 1995).

Para ilustrar esse funcionamento Heinzle (1995) demostra regras que poderiam fazer parte, por exemplo, de um sistema especialista para orientação de candidatos ao curso de mestrado.

Regra:

SE um candidato se inscrever para o curso de mestrado

E o candidato preencher todos os requisitos exigidos

E existirem vagas disponíveis

ENTÃO o candidato passa a ser aluno do curso de mestrado

Regra:

SE o aluno de mestrado for aprovado em todas as disciplinas do currículo ENTÃO ele deve escrever uma dissertação

Regra:

SE a dissertação está concluída será submetida a uma banca examinadora

E a banca examinadora aprovar o trabalho

ENTÃO o aluno receberá o grau de mestre

As regras de produção além de ter uma interpretação natural para os humanos ainda apresenta como aspectos positivos sua modularidade e uniformidade, possibilitando manipular as regras como peças independentes e incluir novas a qualquer momento além de facilitar sua manutenção e interpretação por pessoas não familiarizadas com o sistema (HEINZLE, 1995).

# 4. Trabalhos relacionados

Durante a pesquisa realizada poucos obras recentes foram encontrados com um objetivo similar ao apresentado neste trabalho, é possível citar entre esses o trabalho de Ferreira (2016) que teve como objetivo apresentar uma abordagem para a automatização testes de regressão de *software* para aplicações empresariais desenvolvidas no contexto *web* com metodologias ágeis, para isso, além da pesquisa teórica, uma aplicação é apresentada como forma de atingir os objetivos declarados. Como resultado o autor observa que um agente artificial inteligente pode ser usado na automatização da execução de testes de regressão, pela reutilização dos testes manualmente escritos para cada uma das histórias de uso. O agente proposto tem também capacidade de reagir a não conformidades do sistema em teste, aplicando sequências de atividades para a correção das anomalias ou para a coleta de mais informação de diagnóstico.

Frota (2017) apresenta uma metodologia para a criação de testes de *software* automatizados tendo como base a utilização de Aprendizado de Máquina, através de Rede Neural Artificial, para o reconhecimento de padrões em páginas HTML. Através de um levantamento bibliográfico e aplicação prática o autor observa que é possível simplificar a realização dos casos de teste através de uma linguagem de alto nível. Observou-se também a possibilidade de viabilizar um ambiente de alta performance para realização de testes automatizados de software e interpretação dos resultados através de Processamento Natural de Linguagem (NLP).

É possível correlacionar os trabalhos citados com diversos aspectos do trabalho apresentado tendo em vista os objetivos abordados por ambos os trabalhos sendo esses focados em testes automatizados para aplicações web, diferindo em sua intenção final sendo um criar uma automação de ponta a ponta baseando-se em um caso de teste (Ferreira, 2016), outro focado em automatizar casos de teste utilizando aprendizado de máquina (Frota, 2017) e a terceira auxiliar o testador durante a construção da automação.

# 5. Metodologia

## 5.1. Natureza da pesquisa

Segundo Gil (2008) essa pesquisa pode ser definida quanto a sua natureza como uma pesquisa aplicada e quanto a seu propósito com uma pesquisa exploratória dado que sua principal finalidade é esclarecer e conceitos e ideias, tendo em vista a formulação de hipóteses pesquisáveis e aplicáveis de maneira prática em estudos posteriores. Sua abordagem pode ser definida como qualitativa utilizando procedimentos de revisão de literatura especializada, tanto nacional quanto estrangeira, sendo utilizados livros, revistas, artigos científicos, dissertações de mestrado e teses de doutorado.

### 5.1.1 Experimento de pesquisa

Este experimento de pesquisa propõe uma metodologia focada na possibilidade de criação de códigos de automação de testes a partir de casos de teste padronizados, atribuindo linhas de código já cadastradas a passos semelhantes de diversos casos de teste. Dessa forma o testador pode cadastrar cada tipo de ação apenas uma vez na base de conhecimento e reutilizá-la em todos os testes subsequentes.

Para o desenvolvimento de tal metodologia se torna necessário o uso de tecnologias de padronização de escrita de casos de teste, para deixar as informações mais completas e uniformes para análise, ferramentas de inteligência artificial para o reconhecimento de passos necessários para complementar a automação do teste, além de uma base de conhecimento para manter e consultar os passos já criados em cada CT.

O paradigma de IA proposto foi o uso de Regras de Produção utilizando *frameworks* e linguagens de programação voltadas ou adaptáveis para automação de testes, sendo essas livres para uso como convir ao usuário.

Esse trabalho teve como objetivo propor uma metodologia capaz de facilitar a automação de testes, visando a apoiar os testadores na programação desses scripts e agilizar o processo de teste, podendo ser facilmente empregada em projetos de desenvolvimento *web*.

# 6. Proposta de modelo de teste automatizado

A metodologia proposta tem como objetivo auxiliar os testadores durante a criação de testes automatizados ao armazenar pedaços de código reutilizáveis e organizá-los de maneira a atender as necessidades de cada CT.

A proposta é composta por cinco elementos: 1 - utilização de CT padronizado; 2 - codificação referente a cada passo deste CT; 3 - utilização de um banco de dados para armazenar a Base de Conhecimento, ou seja, as informações referentes aos CTs e automações inseridas no sistema; 4 - utilização de uma memória de trabalho ou curto termo utilizada pela IA durante o processo de inferência e; 5 - utilização de um mecanismo de IA capaz de analisar os passos existentes na base de conhecimentos e propor uma codificação para cobrir passos implícitos anteriores aos informados.

Com o intuito de ilustrar o funcionamento da metodologia foi sugerida uma aplicação fictícia onde cada CT deve possuir um “Ponto de partida” – delimitando a tela em que o CT deve iniciar – e um ou mais passos a serem executados, ou seja, as ações necessárias. O “Ponto de partida” deve ser selecionado em um *drop box* e os passos serão adicionados um a um registrando uma “Condição inicial”, uma “Ação” e um “Resultado esperado / Conclusão” selecionando-os também em *drop boxes*. Ao selecionar a função desejada (condição inicial, ação ou resultado esperado) um número de caixas de texto igual ao de parâmetros necessários serão exibidos e não devem ser deixados em branco.

Em seu primeiro uso o testador deverá alimentar a aplicação com as funções desejadas incluindo seu Nome, Número de parâmetros e Código da função.

## 6.1. Utilização de regras de produção

A técnica Regras de Produção será utilizada na criação dos testes automatizados para traçar um caminho partindo da Página Inicial até o Ponto de Partida definido em um CT, para isso serão verificados todos os Casos de Teste observando suas Condições Iniciais e Resultados Esperados. Considerando um cenário em que haja Casos de Teste em todas as páginas da aplicação, uma espécie de mapeamento será produzido partindo do Ponto de Partida desejado e “fazendo o caminho inverso” até chegar na página inicial.

A aplicação usará as Regras de Produção da seguinte forma: Cada passo dos Casos de Testes será considerado como uma regra (Condição Inicial = SE, Ação = ENTÃO, Resultado Esperado = Conclusão / Verdade), ao iniciar no Ponto de Partida a aplicação considera-o como verdade e busca qual Condição Inicial levou até àquele resultado.

Para exemplificar o funcionamento é possível considerar os seguintes Casos de Teste registrados na aplicação:

O Quadro 1 apresenta um CT em que um dos passos (o primeiro) leva da página inicial (Home) à página Administração.

Quadro 1 - Caso de Teste Abas Principais

|  |  |  |
| --- | --- | --- |
| Caso de Teste: Abas Principais | | |
| Condição Inicial | **Ação** | **Resultado Esperado** |
| Página [Home] visível | **Clicar em aba [Administração]** | **Página [Administração] visível** |
| Página [Administração] visível | Clicar em aba [Coordenação] | Página [Coordenação] visível |
| Página [Coordenação] visível | Clicar em aba [Usuário] | Página [Usuário] visível |
| ... | ... | ... |

*Fonte: A autora*

Já o Quadro 2 apresenta um CT no qual um dos passos leva da página Administração à página Requisição de acesso.

Quadro 2 - CT Opções da Página Administração

|  |  |  |
| --- | --- | --- |
| Caso de Teste: Opções da Página Administração | | |
| Condição Inicial | **Ação** | **Resultado Esperado** |
| Página [Administração] visível | **Clicar em [Requisição de acesso]** | **Página [Requisição de acesso] visível** |
| Página [Requisição de acesso] visível | Clicar em aba [Administração] | Página [Administração] visível |
| Página [Administração] visível | Clicar em [Usuários] | Página [Usuários] visível |
| ... | ... | ... |

*Fonte: A autora*

O Quadro 3 apresenta um CT em que um dos passos tem como Condição Inicial a página Requisição de acesso.

Quadro 3 - Caso de Teste Criação de Requisição de Acesso

|  |  |  |
| --- | --- | --- |
| Caso de Teste: Criação de Requisição de Acesso | | |
| Condição Inicial | **Ação** | **Resultado Esperado** |
| Página [Requisição de acesso] visível | Clicar em [Nova requisição de acesso] | Página [Criar requisição de acesso] visível |
| Página [Criar requisição de acesso] visível | Preencha o campo [País] com [Brasil] | Campo [País] preenchido com [Brasil] |
| ... | ... | ... |

*Fonte: A autora*

Tendo em vista que o CT escolhido para conversão em código automatizado tem como Ponto de Partida a página “Requisição de acesso” (Caso de Teste Criação de Requisição de Acesso – Quadro 3), então “Página [Requisição de acesso] visível” é considerada verdade e procura-se a Condição Inicial que leva a esse resultado, encontrando assim a Condição Inicial “Página [Administração] visível” (Caso de Teste Opções da Página Administração – Quadro 2). O código relacionado a esse passo é reservado na memória de trabalho, a Condição Inicial é comparada com a página inicial buscada e não sendo igual o processo continua. Se inicia a busca pela Condição Inicial que tem como resultado esperado “Página [Administração] visível” encontrando assim a Condição Inicial “Página [Home] visível” (Caso de Teste: Abas Principais – Quadro 1), o código relacionado a esse passo é reservado acima do anterior e ao comparar a página encontrada com a página inicial buscada se tem um resultado positivo e o processo é encerrado.

Nesse processo, dois trechos de código foram adicionados antes do começo do CT escolhido. Dessa forma, o CT convertido terá a seguinte estrutura:

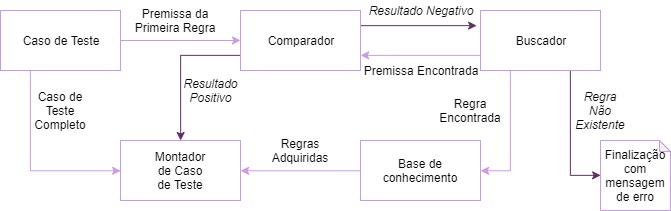
Quadro 4 - Caso de teste Criação de Requisição de Acesso gerado após ação da Inteligência Artificial

|  |  |  |
| --- | --- | --- |
| Condição Inicial | Ação | Resultado Esperado |
| Página [Home] visível | Clicar em aba [Administração] | Página [Administração] visível |
| Página [Administração] visível | Clicar em [Requisição de acesso] | Página [Requisição de acesso] visível |
| Página [Requisição de acesso] visível | Clicar em [Nova requisição de acesso] | Página [Criar requisição de acesso] visível |
| Página [Criar requisição de acesso] visível | Preencha o campo [País] com [Brasil] | Campo [País] preenchido com [Brasil] |
| ... | ... | ... |

*Fonte: A autora*

Esse fluxo de funcionamento pode ser verificado na figura 9 , na qual a premissa da Primeira Regra do CT é mandada para um Comparador onde se verifica se a premissa recebida é igual à página Inicial configurada, em caso negativo essa premissa é enviada para o Buscador que procurará na Base de Conhecimento uma regra que possua como resultado esperado a mesma premissa recebida, caso o Buscador encontre uma regra que satisfaça a busca, esse a salva na memória de trabalho e devolve a premissa encontrada para o Comparador, sendo essa premissa equivalente ao Home, a busca é finalizada, as regras encontradas pelo Buscador são enviadas para o Montador de Caso de Teste e as regras do CT inicial são anexadas a elas gerando o CT final. Caso o Buscador não seja capaz de encontrar uma regra que satisfaça a busca então a operação é finalizada apresentando um erro.

Figura 9 - Funcionamento da Inteligência Artificial na proposta

*Fonte: A autora*

Para possibilitar esse funcionamento torna-se necessário o uso de uma árvore de busca utilizando o encadeamento regressivo, ou *backward*, na busca de regras, dado que a regra analisada terá como foco sua conclusão e esta será procurada entre as premissas das demais regras.

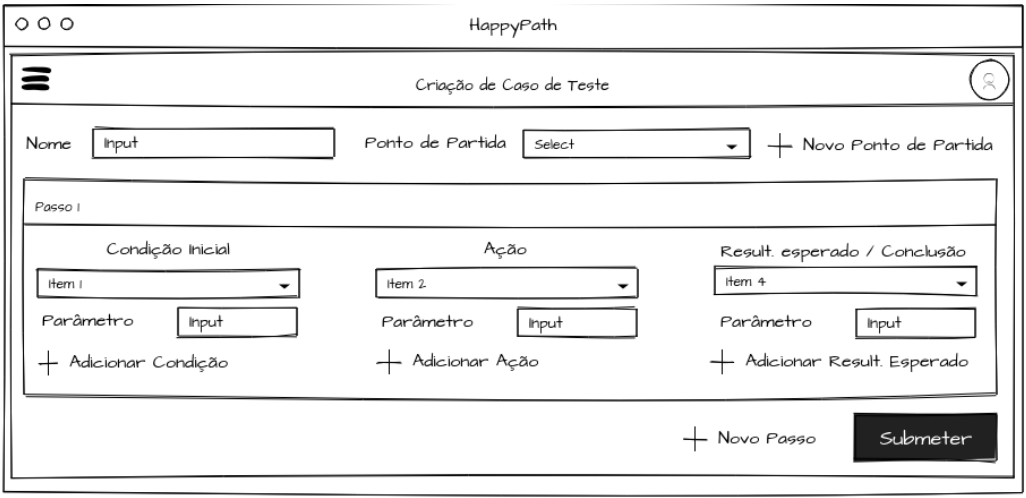
## 6.2. Aplicação HappyPath

### 6.2.1. Telas

Como forma de exemplificar a metodologia, o protótipo HappyPath foi apresentado. Segue as telas da aplicação com suas funções.

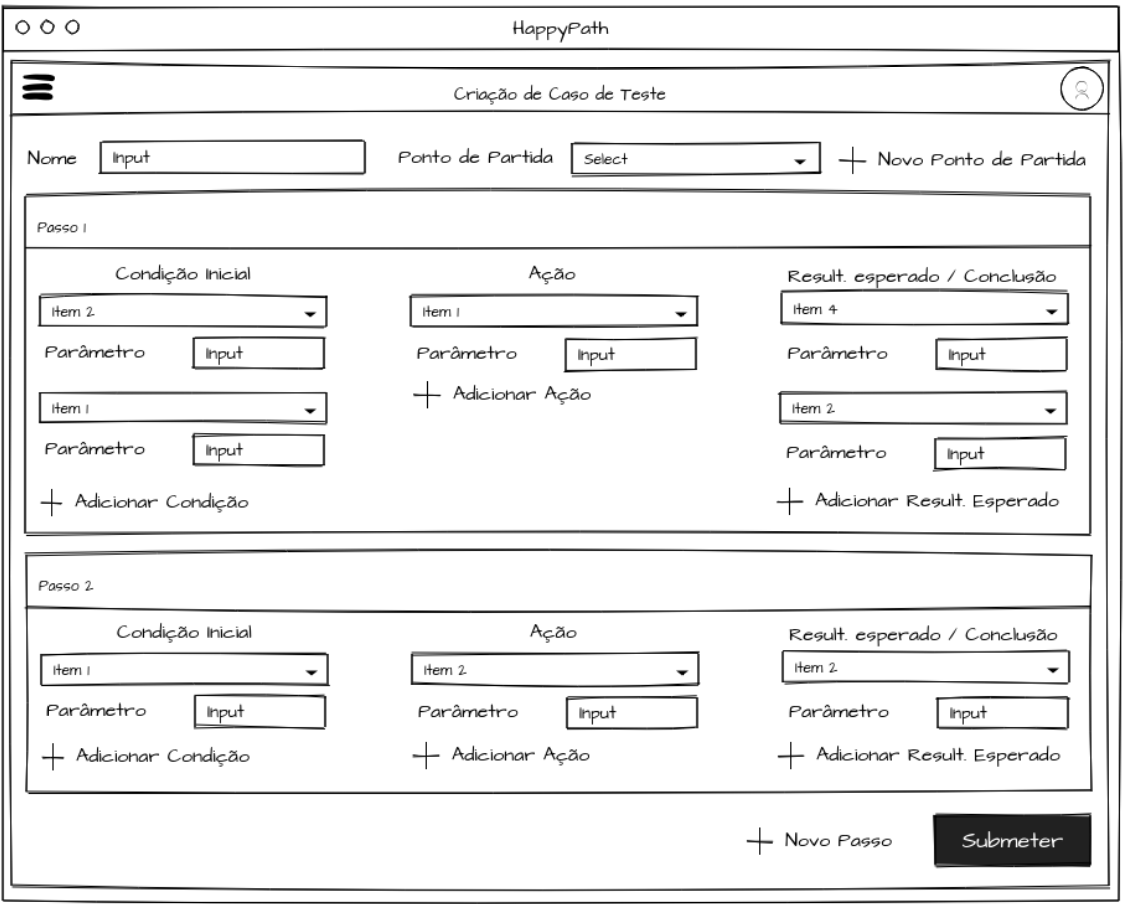
A tela de Criação de Casos de Testes apresenta um campo de texto para a inclusão do Nome do Caso de Teste, um campo de seleção para escolher o Ponto de Partida e por padrão o primeiro passo do CT, contendo os campos de seleção da Condição Inicial, Ação e Resultado Esperado/Conclusão, cada um exibindo um número de caixas de texto para a inclusão de Parâmetros igual à registrada na opção escolhida, como ilustra a figura 10.

Figura 10 - Tela de criação de testes – Primeiro passo

*Fonte: A autora*

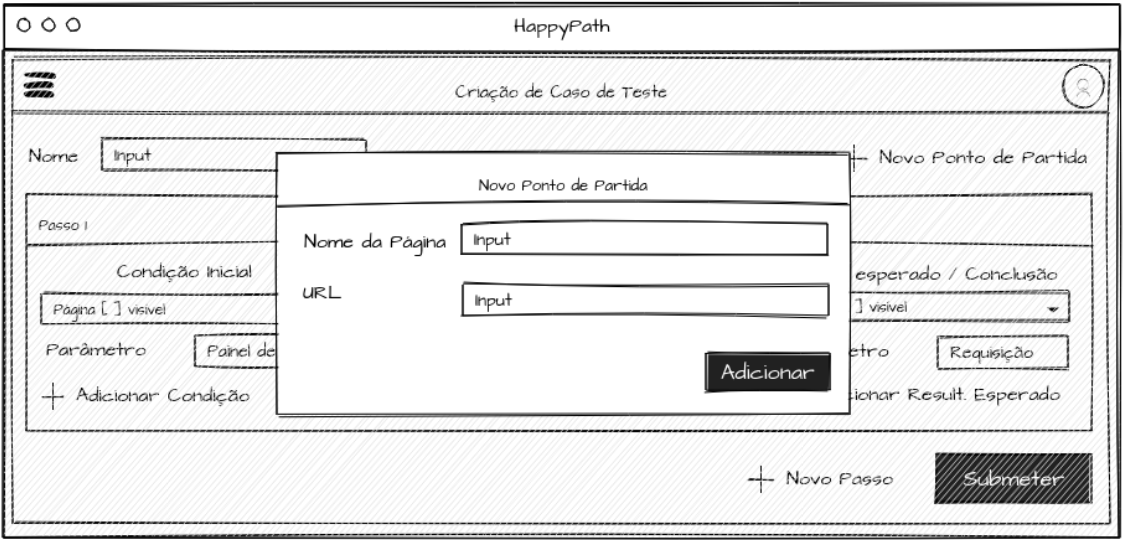
Será possível incluir mais Condições Iniciais, Ações e Resultados Esperados / Conclusões ao clicar nos sinais de adição abaixo de cada campo e adicionar novos passos ao clicar no sinal de adição ao lado do botão Submeter, conforme ilustra a figura 11.

Figura 11 - Tela de criação de testes – Adição de passo

 *Fonte: A autora*

Caso o Ponto de Partida desejado não esteja cadastrado será possível criá-lo através do sinal de adição ao lado do campo de seleção, conforme mostrado na figura 12.

Figura 12 - Modal de inclusão de Ponto de Partida

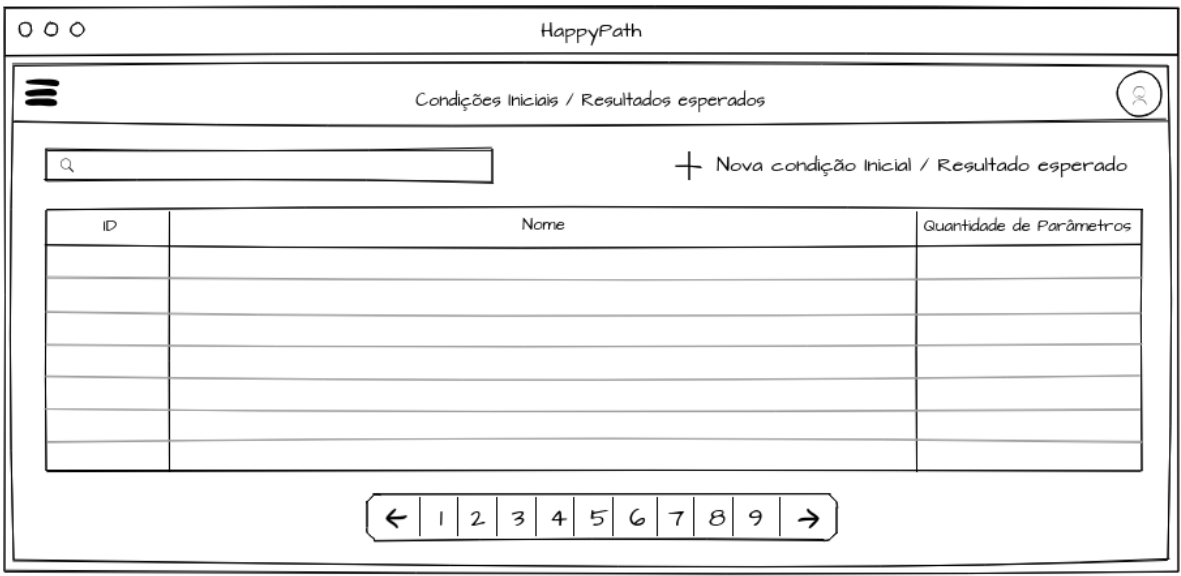
 *Fonte: A autora*

As opções de Condições Iniciais, Ações e Resultados Esperados/Conclusões devem ser criadas em telas diferentes.

As Condições Iniciais e Resultados Esperados/Conclusões serão adicionadas na mesma tela (Condições Iniciais / Resultados Esperados) e armazenadas na mesma tabela da Base de Conhecimento por se tratar de situações semelhantes. Nesta tela serão exibidos todos os itens criados identificados por ID, Nome e Quantidade de Parâmetros, será possível pesquisar por esses identificadores e navegar entre as páginas.

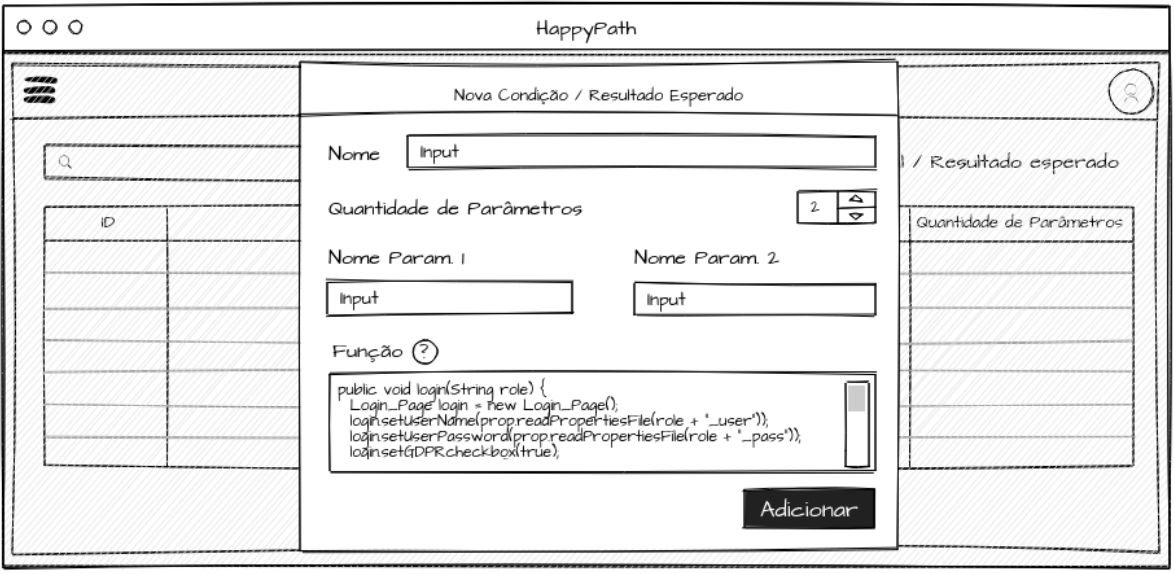
Será possível adicionar um novo item clicando no sinal de adição Nova condição inicial / resultado esperado, como visto na figura 13.

Figura 13 - Tela Condições Iniciais / Resultados esperados

 *Fonte: A autora*

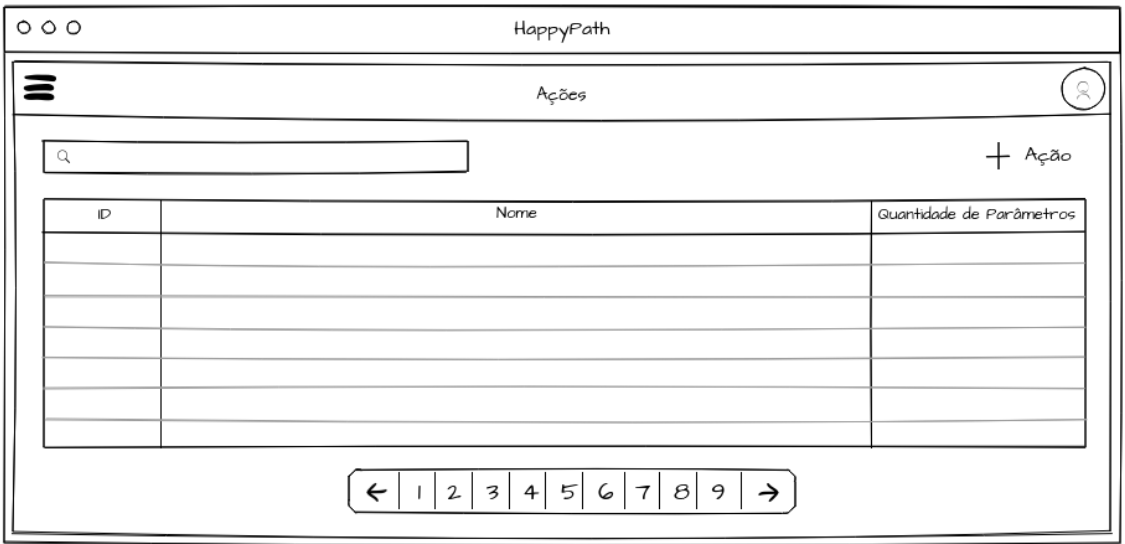
Na tela Nova Condição / Resultado Esperado será apresentado um campo de texto para a inclusão do nome da condição inicial ou resultado esperado / conclusão, um campo para seleção da quantidade de parâmetros que devem ser passados, caixas de texto para o nome dos parâmetros e uma área de texto para a inclusão do código relacionado a essa regra, como ilustrado na figura 14.

Figura 14 - Modal de criação de nova Condição Inicial / Resultado Esperado

 *Fonte: A autora*

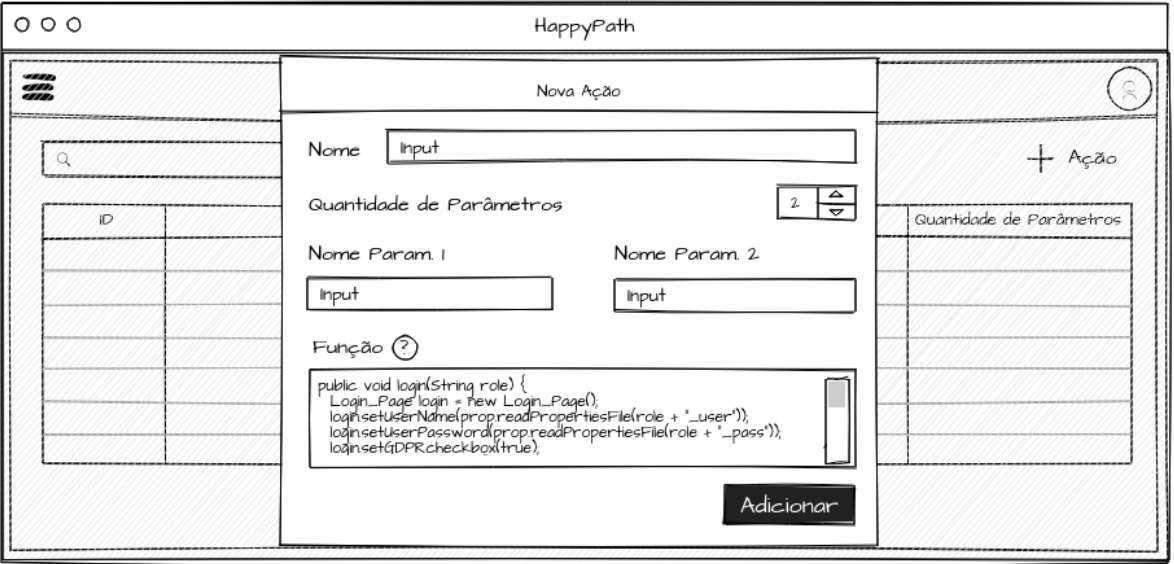
Da mesma forma funcionará com as Ações, uma tela conterá todas as ações cadastradas identificadas por ID, Nome e Quantidade de Parâmetros, nessa tela será possível pesquisá-las por esses identificadores ou navegar entre as páginas. Para adicionar uma nova Ação deve-se clicar sobre o sinal de adição no topo da página, como visto na figura 15.

Figura 15 - Tela Ações

 *Fonte: A autora*

A tela Nova Ação possuirá um campo de texto para a inclusão do nome da ação, um campo para seleção da quantidade de parâmetros que devem ser passados, caixas de texto para o nome dos parâmetros e uma área de texto para a inclusão do código relacionado a essa regra, assim como mostrado na figura 16.

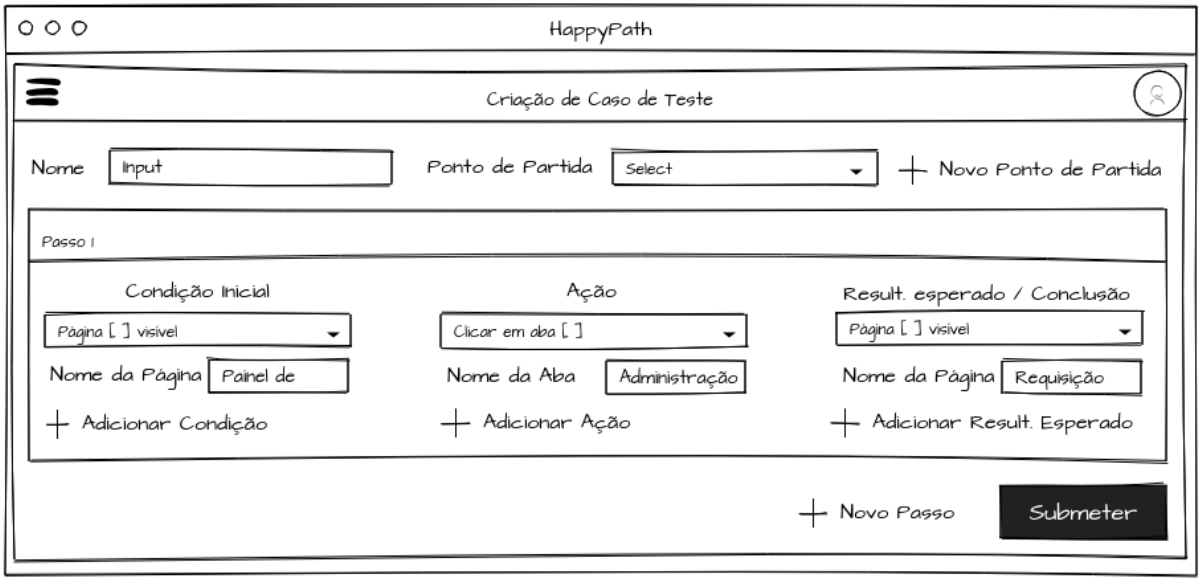
Figura 16 - Modal de criação de nova Ação

 *Fonte: A autora*

A criação das condições iniciais, ações e resultados esperados devem seguir um padrão para se manterem genéricas, e assim poderem ser reutilizadas.

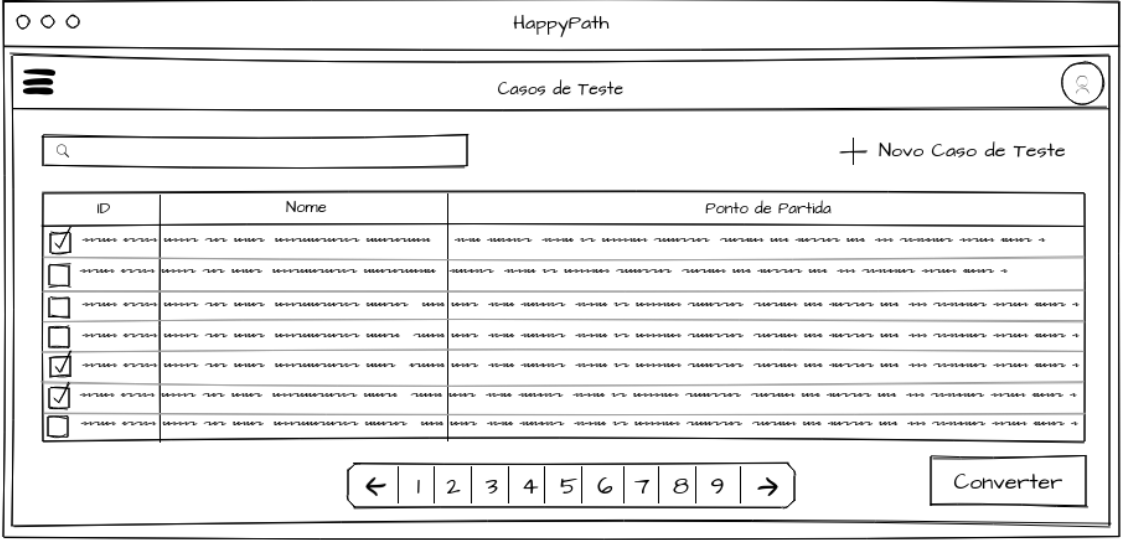
Os nomes devem ser intuitivos para que o usuário entenda a função executada pelo código, como ilustrado na figura 17, em que o nome da condição inicial é “Página [ ] visível” e os colchetes (“[ ]”) indicam onde o parâmetro passado será incluído. Nesse caso, a condição inicial será “Página [Painel de Controle] visível”, caso haja mais de um parâmetro a ser passado, a ordem considerada será a presente no nome da condição / resultado esperado / ação, por exemplo “Preencher campo [ ] com [ ]”, os parâmetros Nome e Marina seriam passados, e respeitando a ordem de inserção a regra seria entendida como “Preencher campo [Nome] com [Marina].

Figura 17 - Tela Criação de Caso de Teste preenchida

 *Fonte: A autora*

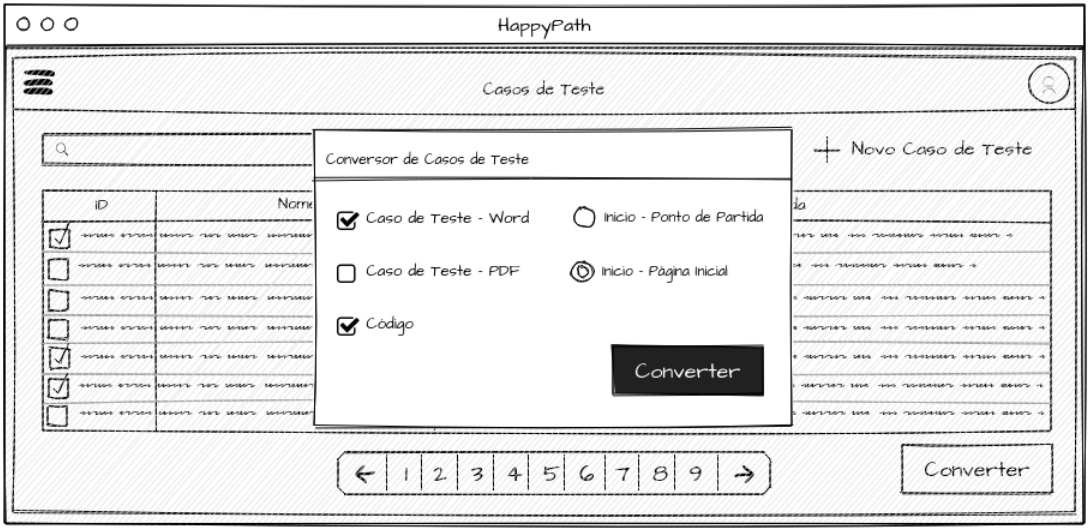
Na tela de Casos de Teste, conforme figura 18, serão exibidos em uma *grid* todos os CTs já criados, identificados pelo ID, Nome e Ponto de Partida. Há um botão para a criação de novos CTs e um para a conversão dos Casos de Teste selecionados.

Figura 18 - Tela Casos de Testes

  
*Fonte: A autora*

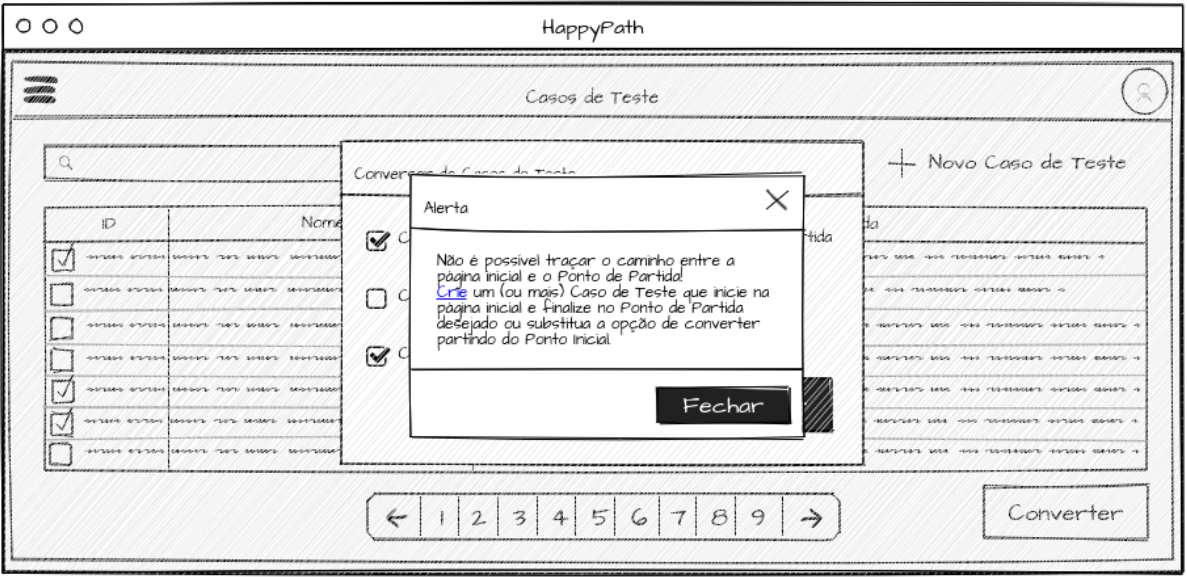
Ao selecionar os CTs desejados e clicar em Converter um modal, será apresentado para a seleção do tipo de conversão desejada, em forma de CT em Word ou PDF ou em forma de Código, também é dada a opção do início do CT no ponto de partida indicado ou na página inicial da aplicação. A Figura 19 ilustra esse cenário.

Figura 19 - Modal Conversor de Casos de Teste

 *Fonte: A autora*

Caso a opção escolhida for iniciar a partir da Página Inicial, ao clicar em converter a aplicação verificará, utilizando a Inteligência Artificial (Regras de Produção), se existem Casos de Teste que levam da página inicial até o Ponto de Partida declarado no CT, caso não haja, uma mensagem de erro será apresentada indicando que não é possível traçar um caminho da Página Inicial até o Ponto de Partida e que, nesse caso, será necessário ou criar um ou mais Casos de Teste que tenham como Ponto de Partida a página inicial e levem até o Ponto de Partida do CT (funcionamento explicado no capítulo 6.2, ou selecionar a opção de conversão partindo do Ponto de Partida do CT, como mostrado na figura 20.

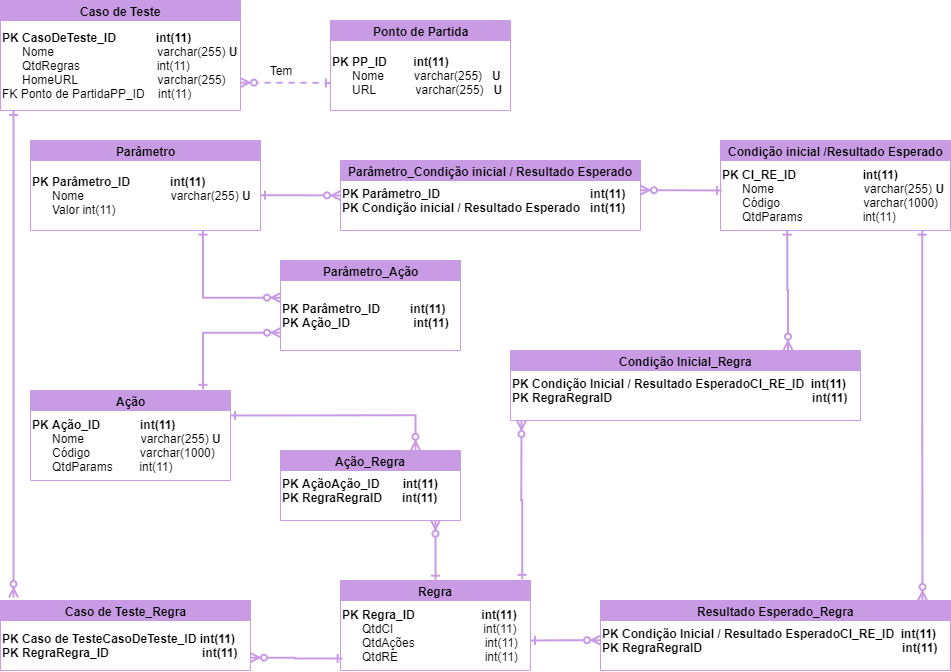
Figura 20 - Janela Pop-up de Alerta

  
*Fonte: A autora*

### 6.2.2. Armazenamento de Dados

Seguindo o protótipo apresentado, o seguinte diagrama de entidade e relacionamento, figura 21, representa como a Base de Conhecimento seria armazenada em um banco de dados.

Figura 21 - Diagrama Entidade Relacionamento

  
*Fonte: A autor*

# 7. Considerações finais

Este trabalho que teve como pergunta de pesquisa: “Como um mecanismo de Inteligência Artificial pode ser utilizado na criação de teste de *software* automatizado?” chega ao seu final com as seguintes considerações:

Inicialmente foi realizada uma pesquisa bibliográfica em literatura especializada visando a explorar o conceito de teste de *software* utilizando automações além de métodos de IA.

Baseando-se nessas pesquisas a metodologia de teste proposta neste trabalho se apresenta através de uma interface com o usuário, a aplicação exemplo HappyPath, a qual é utilizada para alimentar a Base de Conhecimentos. Nesta, o usuário é capaz de armazenar funções comuns em suas automações e assim utilizá-las em CTs subsequentes, além de ter a possibilidade de solicitar a criação automática de passos implícitos que levem da Página Inicial da aplicação sob teste até o Ponto de Partida do CT. Para isto se faz necessário o uso do método Regras de Produção, o motor de inferência se inicia coletando a primeira regra do CT em questão e analisando-a para verificar se esta atende a condição desejada, ou seja, se ela é capaz de levar da Página Inicial da aplicação testada até o Ponto de Partida definido no CT, caso essa regra não atenda esta condição o motor de inferência procurará na Base de Conhecimentos uma outra regra que possua em sua conclusão a premissa da regra sob análise, essa nova regra e sua codificação é salva na memória de trabalho e o processo se segue até que uma regra atenda a condição possibilitando assim a montagem de um novo CT contendo as regras salvas na memória de trabalho durante o processo de inferência adicionado as remais regras do CT inicial.

Portanto essa metodologia de teste mostrou-se capaz de cumprir o papel para qual foi planejada dado que é possível utilizá-la em testes longos e repetitivos, deixando o processo mais ágil e intuitivo. Sua relevância se dá pelo uso de uma interface amigável que permite armazenar funções comumente usadas na construção de automações de teste dando a liberdade ao testador de usar a linguagem de programação que melhor lhe convir, dado que não há limitações na metodologia quanto a esse aspecto.

É esperado que essa metodologia possa ser amplamente usada em projetos de desenvolvimento de *software* que exijam testes longos e/ou repetitivos, potencializando assim a capacidade e efetividade do testador ou da equipe de testes.

# Bibliografia

ALMEIDA, Rui Jorge. **Comparison of fuzzy clustering algorithms for classification.** 2006. Proceedings of the 2006 International Symposium on Evolving Fuzzy Systems, EFS'06. Disponível em: <https://ieeexplore.ieee.org/abstract/document/4016702/authors#authors>. Acesso em: 24 nov. 2019.

BARDOSSY, Andras; DUCKSTEIN Lucien; BOGARDI Istvan. **Fuzzy rule‐based classification of atmospheric circulation patterns**. International journal of climatology 15.10 (1995): 1087-1097.

BRIAND, Lionel; LABICHE, Yvan. A UML-Based Approach to System Testing. **Software And Systems Modeling**, [s.l.], v. 1, n. 1, p.10-42, set. 2002. Springer Science and Business Media LLC. <http://dx.doi.org/10.1007/s10270-002-0004-8>.

CASTROUNIS, Alex. **AI for People and Business**. Sebastopol: O’reilly Media, Inc., 2019. Disponível em: <https://learning.oreilly.com/library/view/ai-for-people/9781492036562/copyright-page01.html>. Acesso em: 16 mar. 2019.

FERNANDES, Anita Maria da Rocha. **Inteligência Artificial:** noções gerais. 2. ed. Florianópolis: VisualBooks, 2005. 160 p.

FERREIRA, Frederico Alexandre. **Inteligência Artificial na Verificação e Teste de Software para Desenvolvimento Ágil.**2016. 101 f. Dissertação (Mestrado) - Curso de Mestre em Engenharia Informática e de Computadores, Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores, Instituto Superior de Engenharia de Lisboa, Lisboa, 2016.

FROTA, Francisco Vitor Lopes da. **USO DE APRENDIZADO DE MÁQUINA PARA A AUTOMAÇÃO DE TESTES DE SISTEMAS WEB**. 2017. 85 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia ElÉtrica, Departamento de Engenharia ElÉtrica, Universidade de BrasÍlia Faculdade de Tecnologia, Brasília, 2017. Disponível em: <https://repositorio.unb.br/bitstream/10482/32003/1/2017\_FranciscoVitorLopesdaFrota.pdf>. Acesso em: 19 nov. 2019.

GESTAL, Marcos et al. **Introducción a los Algoritmos Genéticos y la Programación Genética**. A Coruña: Universidade da Coruña, Servizo de Publicacións, 2010. Disponível em: <http://www.galeon.com/dantethedestroyer/algoritmos.pdf>. Acesso em: 16 mar. 2019.

GIL, Antonio Carlos. **Métodos e Técnicas de Pesquisa Social**. 6. ed. São Paulo: Editora Atlas S.a., 2008. 220 p. Disponível em: <http://197.249.65.74:8080/biblioteca/bitstream/123456789/707/1/M%C3%A9todos%20de%20Pesquisa%20Social.pdf>. Acesso em: 19 nov. 2019.

HAYKIN, S. **Redes neurais**. Princípios e prática. 2.ed Porto Alegre: Bookman, 2001.

JORGENSEN P. **Software testing**: a craftman’s approach, CRC Press, 2002. p. 359.

LIU H.; TAN B. Kuan. **Covering code behavior on input validation in functional testing** Information and Software Technology, vol. 51, no. 2, pp. 546–553, Feb. 2009.

LUGER, George F. **Artificial intelligence**: structures and strategies for complex problem solving. 6. ed. Boston: Pearson Education, Inc, 2009. Disponível em: <http://dspace.fue.edu.eg/xmlui/bitstream/handle/123456789/3379/2387.pdf?sequence=1>. Acesso em: 16 mar. 2019.

LUO, Lu. **Software Testing Techniques:** Technology Maturation and Research Strategy. Pittsburgh: Institute For Software Research International Carnegie Mellon University, 2001. 19 p. Disponível em: <http://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf>. Acesso em: 27 out. 2019.

MENDONÇA, Vinícius Rafael Lobo de. **Estudo, Definição e Implementação de um Sistema de Recomendação para Priorizar os Avisos Gerados por Ferramentas de Análise Estática.** 2014. 109 f. Dissertação (Mestrado) - Curso de Ciência da Computação., Instituto de Informática da Universidade Federal de Goiás, Universidade Federal de Goiás, Goiânia, 2014. Cap. 2. Disponível em: <https://repositorio.bc.ufg.br/tede/bitstream/tede/4338/5/Disserta%c3%a7%c3%a3o%20-%20Vin%c3%adcius%20Rafael%20Lobo%20de%20Mendon%c3%a7a%20-%202014.pdf>. Acesso em: 31 out. 2019.

MILLER, Edward F. **Introduction to Software Testing Technology**, Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16.

MIRANDA, Elisângela Maschio de; SANTOS, Fernando Gomes dos. Raciocínio Baseado em Casos. In: FERNANDES, Anita Maria da Rocha. **Inteligência Artificial:** noções gerais. 2. ed. Florianópolis: Visualbooks, 2005. Cap. 3, p. 18,19.

MITRA P.; CHATTERJEE S.; ALI N. **Graphical analysis of MC/DC using automated software testing**, in Electronics Computer Technology (ICECT), 2011 3rd International Conference on, 2011, vol. 3, pp. 145 –149.

MONARD, Maria Carolina; BARANAUKAS, José Augusto. “Indução de regras e árvores de decisão”. **Sistemas Inteligentes Para Engenharia**. Rezende: So Editora Manole Ltda, 2003. Disponível em: <http://dcm.ffclrp.usp.br/~augusto/publications/2003-sistemas-inteligentes-cap5.pdf>. Acesso em: 16 mar. 2019.

MORE: Mecanismo online para referências, versão 2.0. Florianópolis: UFSC Rexlab, 2013. Disponível em: ‹ http://www.more.ufsc.br/ › . Acesso em: 16 mar. 2019.

MYERS, Glenford J.; BADGETT, Tom; SANDLER, Corey. **The Art Software Testing.** 3. ed. New Jersey: John Wiley & Sons, 2012. 256 p.

NEAPOLITAN, Richard E.; JIANG, Xia. **Contemporary Artificial Intelligence**. Boca Raton: Crc Press - Taylor & Francis Group, 2013. Disponível em: <https://learning.oreilly.com/library/view/contemporary-artificial-intelligence/9781466573192/OEBPS/9781466559400\_cvi.htm>. Acesso em: 16 mar. 2019.

ORSO, Alessandro. **Integration Testing of Object-Oriented Software.** 1998. 119 f. Tese (Doutorado) - Curso de Dottorato di Ricerca In Ingegneria Informatica e Automatica, Politecnico di Milano, Milão, 1998.

PEDRYCZ, Witold; GOMIDE, Fernando. **An introduction to fuzzy sets**: analysis and design. Mit Press, 1998.

PRESSMAN, R. **Engenharia de Software**. AMGH, 2011. ISBN 9788563308337. Disponível em: <https://books.google.com.br/books?id=eRIOuQAACAAJ>.

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial Intelligence:**A Modern Approach. 2. ed. New Jersey: Pearson Education, Inc., Upper Saddle River, 2003. 1112 p.

SHAO D.; KHURSHID S.; PERRY D. E. **A Case for White-box Testing Using Declarative Specifications Poster Abstract**, in Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, 2007, p. 137.

SAGLIETTI, F.; OSTER, N.; PINTE F. **White and grey-box verification and validation approaches for safety- and security-critical software systems**, Information Security Technical Report, vol. 13, no. 1, pp. 10–16, 2008.

SHAW, M. **What makes good research in software engineering?**, International Journal on Software Tools for Technology Transfer (STTT), vol. 4, no. 1, pp. 1–7, 2002.

SOMMERVILLE, I. et al. **Software Engineering**. ADDISON WESLEY BRA, 2008. ISBN 9788588639287. Disponível em: <https://books.google.com.br/books?id=ifIYOgAACAAJ>.

VON WANGENHEIM, Christiane Gresse; VON WANGENHEIM, Aldo. **Raciocínio baseado em casos**. Barueri: Manole Ltda., 2003.

# Glossário

***Backward*** – Para trás.

**Base de Conhecimentos** - Local onde se armazenam fatos, heurísticas, crenças etc.

***Bias*** - Viés.

***Branch*** - Ramo ou ramificação de algo.

**CT** - Caso de teste, conjunto de condições usadas para teste de software.

***Design*** - No contexto de *software* trata-se do projeto, comumente ligado ao visual do mesmo.

***Dropbox*** - Tipo de campo usado em uma aplicação onde seleciona uma opção entre as fornecidas.

***Flag*** *-* Bandeira, no contexto de *software* refere-se à sinalização de algum estado na aplicação.

***Forward*** - Frente ou para frente.

***Frameworks*** -Estrutura. É uma abstração na qual o *software* que fornece funcionalidade genérica pode ser alterado seletivamente por código escrito pelo usuário adicional, fornecendo software específico ao aplicativo.

***Grid*** - Grade. No contexto de *software* trata-se do componente que apresenta informações em forma de tabela.

***Happy Path*** - No contexto de *software* é o cenário padrão de funcionamento, sem erros ou exceções.

***Pop-up*** - No contexto de *software* trata-se de uma janela apresentada posteriormente em uma aplicação com alguma informação.

***Primary Key*** – Chave primária. Valor único utilizado para identificar uma informação em um banco de dados.

**RBC** - Raciocínio Baseado em Casos