APRENDIZAGEM POR REFORÇO NA EXECUÇÃO DE TESTES EXPLORATÓRIOS EM INTERFACES GRÁFICAS

Trabalho de Conclusão de Curso

Engenharia da Computação

Vitor Felix Oliveira da Silva

Orientador: Prof. Joabe Bezerra de Jesus Júnior

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

VITOR FELIX OLIVEIRA DA SILVA

APRENDIZAGEM POR REFORÇO NA EXECUÇÃO DE TESTES EXPLORATÓRIOS EM INTERFACES GRÁFICAS

Aplicação apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, Maio de 2022.

Silva, Vitor Felix Oliveira da
Aprendizagem por Reforço na execução de Testes Exploratórios em Interfaces Gráficas/ Vitor Felix Oliveira da Silva. - Recife - PE, 2022.
xiii, 86 f. : il. ; 29 cm.

Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) Universidade de Pernambuco, Escola Politécnica de Pernambuco local, ano

Orientador (a): Prof. MSc. Joabe Bezerra de Jesus Júnior.

Inclui referências.

1. Teste de Software. 2. Teste Exploratório. 3. Automação de Testes. 4. Aprendizagem de Máquina. 5. Aprendizagem por Reforço. 6. Interface Gráfica de Usuário. I. Aprendizagem por Reforço na execução de Testes Exploratórios em Interfaces Gráficas. II. Júnior, Joabe Bezerra Jesus. III. Universidade de Pernambuco.

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 27/5/2022, às 16h00min, reuniu-se para deliberar sobre a defesa da monografia de conclusão de curso do(a) discente **VITOR FELIX OLIVEIRA DA SILVA**, orientado(a) pelo(a) professor(a) **JOABE BEZERRA DE JESUS JÚNIOR**, sob título APRENDIZAGEM POR REFORÇO NA EXECUÇÃO DE TESTES EXPLORATÓRIOS EM INTERFACES GRÁFICAS, a banca composta pelos professores:

TARCIANA DIAS DA SILVA (PRESIDENTE)
JOABE BEZERRA DE JESUS JÚNIOR (ORIENTADOR)

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,5 (nove e meio)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O(A) discente terá 7 dias para entrega da versão final da monografia a contar da data deste documento.

Tarciana Dias da Silva

AVALIADOR 1: Prof (a) **TARCIANA DIAS DA SILVA**

Joabe Bezerra de Jesus Júnior

AVALIADOR 2: Prof (a) **JOABE BEZERRA DE JESUS JÚNIOR**

AVALIADOR 3: Prof (a)

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Dedico este trabalho a todos os professores que fizeram parte da minha trajetória, em especial, meu orientador, Professor Joabe Jesus, pelos ensinamentos e conselhos que me trouxeram até aqui, assim como aos meus companheiros de empresa que tanto me apoiaram e me ensinaram durante o desenvolvimento do trabalho.

Agradecimentos

Agradeço, primeiramente a Deus, por toda a saúde que me foi concedida até o presente momento e por todas as bênçãos e oportunidades que me surgiram durante essa trajetória acadêmica e profissional.

Agradeço aos meus pais e meu irmão, minha maior riqueza e os maiores apoiadores e incentivadores nessa vida, a quem devo tudo o que tenho e o que sou. Serei eternamente grato por todo o esforço e dedicação que desempenharam para que eu tivesse acesso à educação de qualidade e pudesse ser capaz de abrir novas portas e trilhar novos caminhos.

Agradeço também aos meus amigos, que considero irmãos de outras famílias, e fizeram parte integral da minha caminhada ao longo da graduação, além de tornarem o caminho infinitamente mais fácil e prazeroso. Ao lado de cada um vivenciei momentos únicos que levarei para sempre na memória.

Agradeço, por fim, à instituição Universidade de Pernambuco, que me proporcionou acesso ao conhecimento através do seu corpo docente que sempre se fez presente. E a todas as pessoas com quem tive contato e convivi nos últimos anos, desde os companheiros de estágio nas empresas por onde passei enquanto adquiri conhecimento prático, até os que, mesmo indiretamente, tornaram mais leve essa trajetória.

Autorização de publicação de PFC

Eu, **Vitor Felix Oliveira Da Silva** autor(a) do projeto de final de curso intitulado: **APRENDIZAGEM POR REFORÇO NA EXECUÇÃO DE TESTES EXPLORATÓRIOS EM INTERFACES GRÁFICAS;** autorizo a publicação de seu conteúdo na internet nos portais da Escola Politécnica de Pernambuco e Universidade de Pernambuco.

O conteúdo do projeto de final de curso é de responsabilidade do autor.

Vitor Felix Oliveira da Silva

Vitor Felix Oliveira Da Silva

Joabe Bezerra de Jesus Júnior

Orientador(a): **Joabe Bezerra de Jesus Júnior**

Coorientador(a):

Prof, de TCC: **Daniel Augusto Ribeiro Chaves**

Data: 27/5/2022

Resumo

Com o passar dos anos e evolução acelerada da computação e suas áreas, o campo de Aprendizagem de Máquina segue se desenvolvendo rapidamente com o objetivo de colocar a tecnologia à serviço da melhoria da qualidade de vida das pessoas. A tendência é de que as aplicações da área alcancem maior popularidade e sejam cada vez mais acessíveis. De forma paralela aos objetivos e conquistas alcançados pelo campo no desenvolvimento de sistemas cada vez mais complexos, um dos requisitos é que essas aplicações sejam construídas de forma mais confiável e com menor propensão a erros. Portanto, existe uma preocupação constante com a qualidade dos softwares que são entregues aos usuários finais. O presente trabalho tem como objetivo geral o desenvolvimento de uma solução, baseada em técnicas existentes de Aprendizado de Máquina, que execute, de forma automatizada, Testes Exploratórios com a menor interferência possível do testador responsável pela tarefa, auxiliando todos os envolvidos na atividade de exploração. O trabalho se propõe a preencher uma lacuna existente, tendo em vista as limitações atuais encontradas na busca por alternativas no processo de testes que não seguem *scripts* e demandam criatividade. Assim sendo, sugere-se uma nova forma automatizada e independente das ações de um testador, para explorar sistemas com interface gráfica de usuário. Possibilitando, dessa forma, a identificação de bugs e o desenvolvimento de um relatório com análise e detalhes da execução dos testes.

Palavras-chave: Teste de Software; Teste Exploratório; Automação de Testes; Aprendizagem de Máquina; Aprendizagem por Reforço; Interface Gráfica de Usuário.

Abstract

Over the years and the accelerated evolution of computing and its areas, the field of Machine Learning continues to develop rapidly with the aim of putting technology at the service of improving people's quality of life. The trend is for applications in the area to reach greater popularity and become increasingly accessible. In parallel with the objectives and achievements reached by the field in the development of increasingly complex systems, one of the requirements is that these built applications must be more reliable and less error-prone. Therefore, there is a constant concern with the quality of the software that is delivered to the end users. The present work has as general objective the development of a solution, based on existing Machine Learning techniques, that executes, in an automated way, Exploratory Tests with the least possible interference from the tester responsible for the task, helping all those involved in the exploration activity. The work proposes to fill an existing gap, given the current limitations found in the search for alternatives in the testing process that do not follow scripts and demand creativity. Hence, a new automated way, independent of the actions of a tester, is suggested to explore systems with a graphical user interface. This makes it possible to identify bugs and develop a report with analysis and test execution details.

Keywords: Software Testing; Exploratory Testing; Test Automation; Machine Learning; Reinforcement Learning; Graphical User Interface.

Lista de Figuras

Lista de Siglas

AI	Artificial Intelligence
CASE	Computer Aided Software Engineering
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
DDT	Data-Driven Testing
DP	Dynamic Programming
ET	Exploratory Testing
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MIT	Massachusetts Institute of Technology
ML	Machine Learning
NASA	National Aeronautics and Space Administration
POC	Proof of Concept
QA	Quality Assurance
RL	Reinforcement Learning
RTDP	Real-Time Dynamic Programming
SARSA	State-Action-Reward-State-Action
SBTM	Session Based Test Management
SCM	Source Code Management
SUT	System Under Test
TICS	Tecnologias de Informação e Comunicação
UCB	Upper Confidence Bound
UNICODE	Universal Coded

UTF-8	Unicode Transformation Format 8-bit
WYSIWYG	What You See Is What You Get

Sumário

1 Introdução

1.1 Apresentação

Um dos requisitos no desenvolvimento de sistemas cada vez mais complexos é que esses sistemas sejam construídos de forma mais confiável e com menor propensão a erros (ZHANG; TSAI, 2006). Portanto, existe uma preocupação constante com a qualidade das aplicações que são entregues aos usuários finais. Avaliar e garantir a qualidade de sistemas e reduzir o seu risco de falhas em operação é o papel desempenhado pelos Testes de Software (OLSEN; POSTHUMA; ULRICH, 2019).

Além disso, sabe-se que as atividades de teste são imprescindíveis durante o processo de desenvolvimento de qualquer aplicação computacional. Sessões de teste mais bem distribuídas e aplicadas teriam economizado US$125 milhões para a agência de Administração Nacional da Aeronáutica e Espaço, do inglês *National Aeronautics and Space Administration* (NASA) e o governo norte-americano, que em 1998 lançaram a sonda *Mars Climate Orbiter* com o objetivo de colocá-la na órbita do Planeta Vermelho. Os engenheiros e responsáveis pelo projeto viram, menos de 10 meses após o lançamento, todo o investimento ser queimado e reduzido a pedaços na atmosfera de Marte por conta de um simples erro de conversão de medidas (DAVIS; BARNETT, 2016).

As inspeções e os testes de sistemas exigem especialistas, que podem fazer avaliações subjetivas e custosas. Normalmente, devido a restrições orçamentárias dos projetos, apenas partes de um programa podem ser testadas ou inspecionadas. Ainda assim, essas partes não são necessariamente as mais propensas a erros (BIRT, 2006).

Na Engenharia de Software, o auxílio das ferramentas de *Computer Aided Software Engineering* (CASE) é fundamental. Em particular, na subárea de Testes de Software, esse auxílio está presente nas ferramentas que auxiliam desde o planejamento e elaboração dos testes até sua execução (DIAS, 2017). Entretanto,

essas são tarefas que ajudam a aumentar o conhecimento sobre o programa e, logicamente, pode-se aprender com o sucesso ou fracasso dos testes.

Dedicada a melhorar o desempenho de alguma tarefa através de experiências passadas, a Aprendizagem de Máquina, do inglês *Machine Learning* (ML) baseia-se em ideias de diferentes disciplinas, como probabilidade e estatística, teoria da informação, psicologia, teoria do controle e filosofia (MITCHELL, 1997). O termo, do inglês *Machine Learning*, surgiu em 1959 através de Arthur Samuel, um engenheiro do Instituto Tecnológico de Massachusetts, do inglês *Massachusetts Institute of Technology* (MIT) que o definiu como: "Um campo de estudo que dá aos computadores a habilidade de aprender sem terem sido programados para tal" (SAMUEL, 1959). A ML pode ser considerada uma ramificação da Inteligência Artificial, do inglês *Artificial Intelligence* (AI) e avanços das técnicas de ML apresentam oportunidades de automatizar procedimentos através da AI (PASCHEK; LUMINOSU; DRAGHICI, 2017). Um exemplo de avanço é a Aprendizagem por Reforço, do inglês *Reinforcement Learning* (RL), que consiste em mapear situações para ações de modo a maximizar um sinal numérico de recompensa (SUTTON; BARTO, 2018).

Com o passar dos anos e evolução acelerada da computação e suas áreas, o campo de ML segue se desenvolvendo rapidamente com o objetivo de colocar a tecnologia à serviço da melhoria da qualidade de vida das pessoas (ANDERSON; RAINIE, 2018). Ainda estamos no início de uma grande revolução, onde a demanda por serviços proporciona avanços sem precedentes, respaldados pela necessidade de máquinas eficientes, altamente produtivas e inteligentes, que também está crescendo (MEDEIROS, 2021). Apesar de atualmente o investimento ainda estar mais focado em indústrias e grandes empresas, as aplicações de ML estão se tornando cada vez mais sofisticadas, abordando diferentes propósitos e a tendência é de que alcancem maior popularidade e sejam cada vez mais acessíveis.

Diversas etapas do processo e técnicas de teste já são beneficiadas pelo avanço dessas tecnologias de ML (DURELLI et al., 2019). Porém, as atividades de execução de Testes Exploratórios, do inglês *Exploratory Testing* (ET), que consistem em aprendizagem, projeto e execução de testes realizados de maneira simultânea

(BACH, 2003) e são consideradas puramente humanas e interpretativas, devem também aproveitar-se da disposição de tais ferramentas. Assim, tornar a máquina capaz de explorar, em um ambiente controlado, Interfaces Gráficas de Usuário, do inglês *Graphical User Interface* (GUI) em sistemas ou aplicações web, e interpretar os resultados obtidos é um cenário que seria beneficiado pelas técnicas de RL.

1.2 Objetivos do sistema

1.2.1 Objetivo geral

O presente trabalho tem como objetivo geral o desenvolvimento de uma solução, baseada em técnicas existentes de ML, que execute ET de forma automatizada, com a menor interferência possível do testador responsável pela tarefa, auxiliando todos os envolvidos na atividade de exploração. O trabalho se propõe a preencher uma lacuna existente, tendo em vista as limitações atuais encontradas na busca por alternativas no processo de testes que não seguem *scripts* e demandam criatividade. Assim sendo, sugere-se uma nova forma automatizada e independente das ações de um testador, para explorar sistemas com GUI. Possibilitando, dessa forma, a identificação de bugs e o desenvolvimento de um relatório com análise e detalhes da execução dos testes.

1.2.2 Objetivos específicos

Tendo em vista a alta complexidade e relevância do objetivo geral proposto, alguns objetivos específicos foram definidos visando entender melhor o atual momento das tecnologias abordadas. Os objetivos foram escolhidos a partir de uma análise de qual seria a melhor estrutura para a sequência do desenvolvimento da pesquisa, a fim de maximizar os resultados alcançados através da mesma. Os seguintes objetivos específicos foram definidos:

- Revisar a Bibliografia dos campos de ML, ET e GUI;
- Propor a arquitetura de uma solução para a execução automática de ET que

seja pouco dependente de participação humana.

- Desenvolver uma aplicação em Python na arquitetura proposta que auxilie os profissionais da área de Garantia de Qualidade, do inglês *Quality Assurance* (QA) durante a execução de ET.
- Executar a aplicação desenvolvida em formato de Prova de Conceito, do inglês *Proof of Concept* (POC), analisar e comparar os resultados obtidos;
- Utilização das técnicas mais comuns de ML.

1.3 Metodologia de desenvolvimento

O principal artefato do trabalho será desenvolvido em formato de aplicação e deverá ser implementado utilizando a linguagem de programação *Python* (PYTHON, c2022). O código principal do programa deverá receber, manipular e executar, através do framework *Pytest* (PYTEST, c2015), *scripts* de teste também escritos em *Python* que interajam com uma aplicação alvo, utilizando a biblioteca do *Selenium WebDriver (WEBDRIVER, c2022)*, que permite realizar a interação com os elementos disponíveis na página web. No presente estudo de caso, o *JIRA Software Cloud* (JIRA, c2022) será utilizado como o sistema *web* com GUI a ser explorado pelos *scripts* de teste, de forma que cada arquivo de teste fornecido passe por algumas etapas durante a execução da aplicação, incluindo a recuperação de alguns dados do repositório de Strings criado no *Google Cloud Datastore* (DATASTORE, 2013).

O repositório a ser utilizado foi criado especificamente para ser utilizado na etapa de manipulação do *script* com o objetivo de selecionar as *strings* mais propensas a revelar defeitos na aplicação alvo, e foi construído a partir da união de outros repositórios públicos de *strings* disponíveis na internet, como o *Big List of Naughty Strings* (WOLF, 2016), o *SecLists* (MIESSLER, 2018) e o *Fuzz String Lists* (MCINERNEY, 2016).

Os *scripts* de teste a serem inseridos para a utilização da aplicação devem ser implementados pelo usuário do sistema e eles podem ser gerados através de

ferramentas de gravação e reprodução, como o UI Vision (VISION, c2022) e o Selenium IDE (SELENIUM, c2022) que registram as ações tomadas pelo usuário em uma interface alvo e possibilitam a exportação do arquivo gerado em formatos como *scripts* de teste Python e Java.

Todo o acionamento e configuração dos *scripts* de teste e das rodadas de execução dos mesmos na aplicação alvo, deverão ser facilmente escolhidos através da interface do *Jenkins* (JENKINS, 2011). Esse é o servidor de automação escolhido para ser utilizado durante o presente trabalho e provê integração contínua com o repositório da aplicação desenvolvida, localizado no *Github* (GITHUB, 2008). Além de fornecer inúmeras ferramentas para facilitar a automação, o *backup* e o gerenciamento remoto das execuções dos testes com a menor participação possível do usuário do sistema.

1.4 Descrição dos usuários

A partir do estudo das soluções existentes e dos trabalhos em desenvolvimento, espera-se implementar uma prova de conceito que mostre ser possível auxiliar testadores e trabalhadores envolvidos com qualidade de software através da automação da execução de ET. O trabalho deve ser desenvolvido utilizando as técnicas mais comuns de ML, e tem que ser funcional em sistemas web com GUI.

Otimizar o processo de exploração de interfaces de um sistema resulta em recompensas muito positivas não apenas para o testador envolvido com a atividade, mas também para todo o setor de QA. A prática de automatizar tarefas manuais possibilita um maior desempenho nas atividades de testes e amplia o tempo útil do testador, além de reduzir o risco de falha humana durante a operação do *software* no processo de execução dos testes.

2 Fundamentação Teórica

O capítulo atual trata da base teórica e dos fundamentos estudados durante o desenvolvimento do presente trabalho. Busca-se fornecer, através do conteúdo deste capítulo, conceitos e informações detalhadas sobre os sistemas com GUI, assim como os ET e o campo da ML.

Sobre as aplicações *web* com GUI, são discutidos os conceitos fundamentais e a importância dessas interfaces no contexto dos sistemas *web*, abordando as primeiras interfaces de Interação Humano-Computador, do inglês *Human-Computer Interaction* (HCI) desenvolvidas, o surgimento da GUI ainda na década de 1980, sua estrutura em *websites* e como elas estão intrinsecamente interligadas com o sucesso dos testes de *software*, sejam eles automatizados ou manuais.

Posteriormente, são analisados o conceito e as práticas comuns aplicadas nos ET, uma das abordagens manuais de testes de software. Além disso, pretende-se tratar também sobre a estrutura do processo de exploração, a importância do gerenciamento de tempo durante a execução de tais testes, e a relevância da experiência profissional e características pessoais do testador responsável pela execução da atividade no resultado dos testes realizados.

Por fim, o campo de ML é abordado, seu conceito, as técnicas de Aprendizado Supervisionado, Aprendizado Não Supervisionado e RL, suas aplicações, as características que as diferenciam e finalmente, são discutidas as dificuldades e limitações encontradas atualmente. Dentre as técnicas abordadas, a de RL recebe destaque durante o texto por ser a técnica escolhida para o desenvolvimento deste trabalho.

2.1 Sistemas com Interface Gráfica de Usuário

Entendida como um dispositivo visual acessível, a GUI facilita nossa interação entre o mundo abstrato de zeros e uns dos computadores e a centrada e simbólica interpretação humana desta informação, que pode ser, geralmente, compreendida e

adaptada. A GUI é literalmente uma interface entre as mais variadas linguagens de código de computador existentes e uma linguagem utilizável por humanos. (GWILT, 2005.)

Ao longo dos anos, a pesquisa em HCI tem sido extremamente bem-sucedida e mudou fundamentalmente a computação. Um exemplo é a interface gráfica onipresente usada pelo *Microsoft Windows* 95, que se baseou no *Macintosh*, respaldado no trabalho da *Xerox* PARC, que por sua vez foi fundamentado em pesquisas iniciais no Laboratório de Pesquisa de Stanford, agora parte do Instituto de Pesquisa de Stanford, e no Instituto de Tecnologia de Massachusetts (MIT). (MYERS, 1998)

2.1.1 Interfaces de Interação Humano-Computador

Os estudos em HCI sempre tiveram como uma de suas maiores preocupações o objetivo de projetar novas tecnologias e interações a partir de uma perspectiva centrada nas pessoas. No entanto, o significado disso mudou radicalmente ao longo dos anos. Cada nova onda de tecnologia da computação traz consigo além dos inúmeros benefícios, diferentes desafios para interpretar quais aspectos humanos, computacionais e interativos são importantes. (FITZPATRICK, 2018)

As Tecnologias de Informação e Comunicação (TICs) estão se desenvolvendo em um ritmo acelerado, e cada vez mais fazem parte das nossas vidas pessoais e profissionais. A evolução e disseminação dessas tecnologias chegaram em um nível em que é difícil encontrar pessoas que ainda não tiveram contato com elas, independente de características como classe social, nível de escolaridade ou o local onde moram. Existem diversos estudos sobre a arquitetura de sistemas computacionais e interfaces buscando construir sistemas que favoreçam a experiência do usuário, onde os esforços têm resultado no desenvolvimento de diversas tecnologias e dispositivos para permitir e facilitar a interação com as pessoas. (BARBOSA, 2010)

O processo de desenvolvimento de um sistema interativo influencia a qualidade final do produto. Por isso, o conhecimento em abordagens de design de HCI, métodos, técnicas e ferramentas para construção e avaliação da interface acessível pelo usuário é extremamente relevante. Também é indispensável conhecer e analisar casos de HCI de sucesso e de insucesso em interfaces, de forma a buscar identificar os motivos que levaram a tal resultado. (BARBOSA, 2010)

2.1.2 Surgimento da GUI

Na história da computação, 1995 será lembrado como o ano em que o *Windows* 95 foi lançado. Disponibilizado em 24 de agosto, o sistema operacional foi uma versão nova e aprimorada do popular software de GUI *Windows*, da Microsoft. A GUI surge como um método que permite que os usuários de computador vejam e manipulem diretamente representações de objetos na tela do computador, em vez de abordar os objetos por meio de um código de linguagem de máquina. Por exemplo, o *Windows* permite que os usuários apontem e cliquem com o mouse em representações visuais de documentos para abri-los diretamente sem a necessidade de digitar um comando no teclado. (BARNES, 2010).

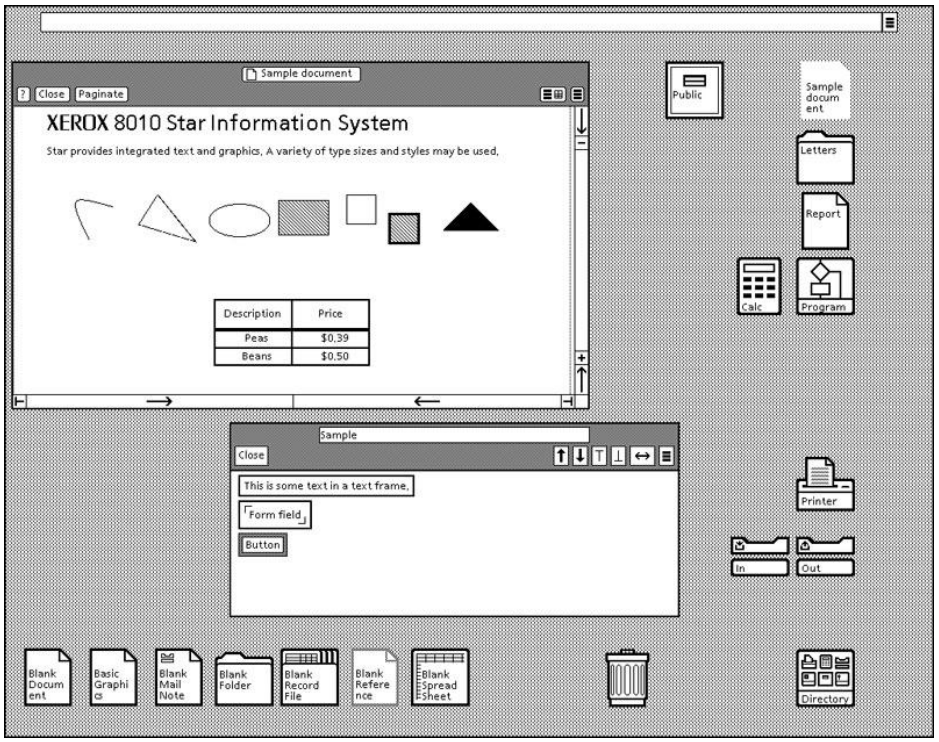
No passado, os sistemas de computador usavam texto básico para executar os desejos do usuário. No entanto, era difícil até mesmo executar comandos simples, e as ações mais complexas exigiam um nível elevado de experiência na linguagem de programação da máquina por parte do usuário. Com a invenção da GUI, os computadores tornaram-se mais fáceis de usar, ela se torna um dos principais meios de permitir a interação do usuário com os mais diversos dispositivos eletrônicos. (ABDO; ALALI, 2016)

O sistema de informação *Xerox Star 8010* foi o primeiro computador comercial com GUI a ser projetado por pesquisadores do centro de pesquisa Xerox Palo Alto em 1977 e, posteriormente, lançado em abril de 1981. A GUI do *Star* utilizava um conceito hoje conhecido como “O Que Você Vê É O Que Você Obtém”, do inglês *What You See Is What You Get* (WYSIWYG), em que as ações feitas pelo usuário

através do mouse, à época recém inventado, eram transferidas diretamente para a navegação na tela, onde os ícones das pastas e documentos representavam as áreas que poderiam ser atuadas sobre. O objetivo do WYSIWYG era manter a interface o mais invisível possível para o usuário de forma que o sistema fosse mais intuitivo e fácil de ser apreendido por pessoas inexperientes. (KERAMIDAS et al., 2015)

Os usuários do *Star* foram incentivados a pensar na área de trabalho da tela, que pode ser vista na figura 1, em termos físicos como ícones ou itens que podem ser movidos, organizados e armazenados da mesma maneira que nas suas áreas de trabalho reais. (KERAMIDAS et al., 2015)

Figura 1 – Interface gráfica do Xerox Star 8010

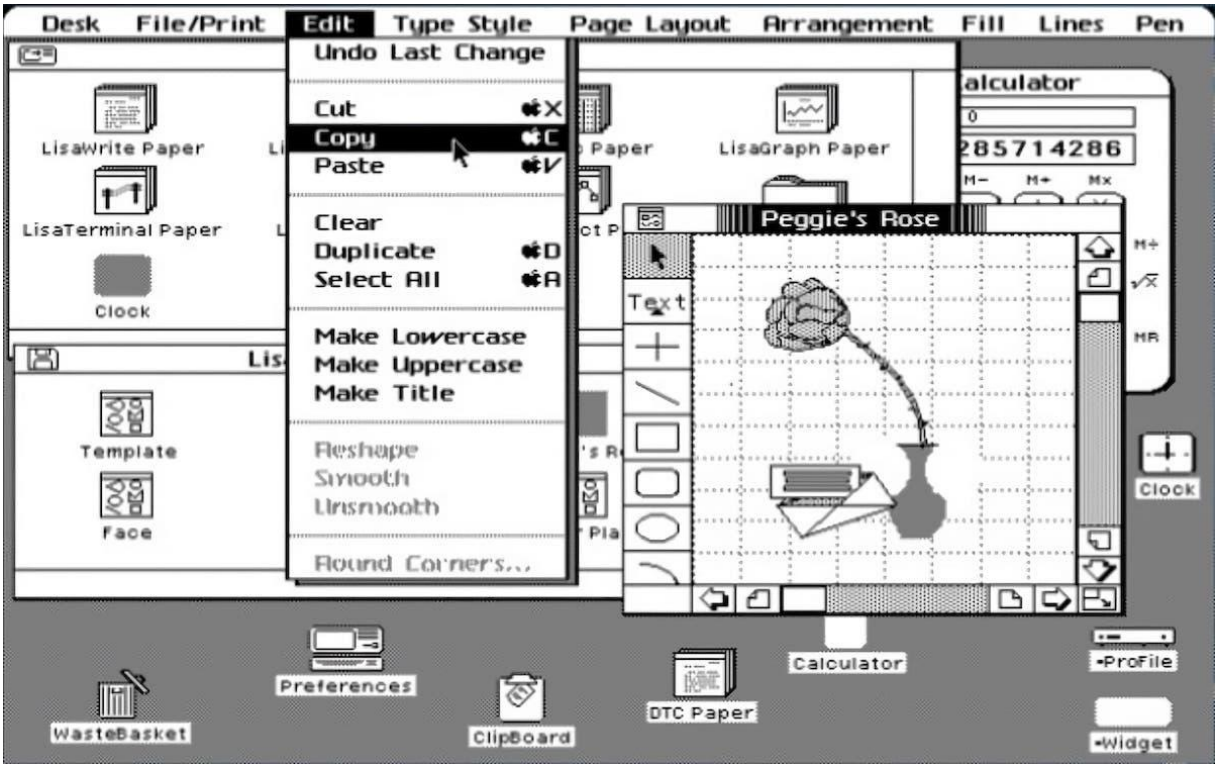


Fonte: Extraído de (KERAMIDAS et al., 2015).

Em 1979, Steve Jobs, um dos fundadores da *Apple*, fez um tour pelo *Xerox Palo Alto Research Center* (PARC) e viu uma aplicação comercial imediata para a tecnologia de interface desenvolvida por Alan Kay, que estava terminando seu *PhD* em ciência da computação na Universidade de Utah. Ele percebeu que colocar uma

tela gráfica nos computadores pessoais da *Apple* os tornaria mais fáceis de operar. Alguns anos mais tarde, a *Apple* alugou alguns recursos do *Xerox Star* para projetar seu primeiro sistema com uma GUI, que ficaria conhecido como *Apple Lisa*, mostrado na imagem 2, sucedido no ano seguinte pelo *Macintosh*. (BARNES, 2010)

Figura 2 – Interface gráfica do Apple Lisa



Fonte: Extraído de (KERAMIDAS et al., 2015).

2.1.3 Estrutura das GUI em sistemas web

Dada a complexidade da arquitetura das GUI e a relação direta entre um *website* bem estruturado e a execução de testes automatizados bem-sucedida, deve-se levar em consideração algumas características essenciais para a construção de um site de sucesso. O processo de desenvolvimento *web* vai além de conhecer linguagens como o *Hypertext Markup Language* (HTML) e o *Cascading Style Sheets* (CSS). A GUI deve envolver vários aspectos significativos, como a consistência e o tempo de carregamento. O sucesso ou fracasso de qualquer site depende muito de como esses atributos são abordados, de forma que a análise

dessas características ajuda a tornar a interface fácil de usar, atraente e eficaz. Os recursos de conteúdo da GUI são usados para manter contato com os visitantes, esses elementos são geralmente desenvolvidos visando o desejo e necessidades do cliente e do usuário final da aplicação. (ABDO; ALALI, 2016)

Os elementos presentes no código HTML do site, na forma de *tags*, são de extrema relevância para a interação dos scripts de automação com a interface alvo. É através do mapeamento desses elementos que se torna possível, por meio do navegador, clicar em botões presentes na tela, inserir dados em campos de texto, entre outras ações. Portanto, é uma característica importante da estrutura dos sites, da perspectiva dos testes automáticos, que seus elementos possuam identificadores únicos, o que auxilia a tarefa de mapeamento dos elementos presentes e reduz a chance de *scripts* de teste ficarem obsoletos após a realização de manutenções periódicas no site, que podem resultar em mudanças na posição de elementos já mapeados. A utilização dos identificadores únicos no mapeamento dos elementos, reduz drasticamente a manutenção das implementações dos *scripts* de teste e aumenta a consistência e confiabilidade dos resultados das execuções dos mesmos.

A performance do site também é extremamente importante na tomada de decisão do usuário quanto a permanecer ou sair da página, e algumas práticas podem ser adotadas para que o tempo de carregamento de um site seja reduzido. Dentre elas, vale destacar algumas como usar menos imagens, usar um servidor à parte para hospedagem de vídeos, como o *YouTube*, e colocar arquivos CSS e *javascript* separadamente dos arquivos HTML. (ABDO; ALALI, 2016) Um site com má performance afeta negativamente não apenas a experiência dos usuários, mas também os testes automatizados realizados na plataforma. Tendo em vista que os testes automáticos seguem passos definidos em *scripts* comumente inflexíveis e que normalmente apresentam dificuldades em contornar longos tempos de espera por respostas do servidor da aplicação, é importante que a performance seja priorizada durante a construção e manutenção de qualquer site.

A consistência no desenvolvimento de sites é um aspecto muito significativo que resulta na ampliação da satisfação do usuário e o conhecimento do mesmo sobre o site. (ABDO; ALALI, 2016) Quando uma GUI usa cores, fontes e *layout* consistentes, e quando títulos e auxílios de navegação estão sempre no mesmo lugar, os usuários sabem onde estão e onde irão encontrar as coisas. Eles não precisam descobrir um novo *layout* toda vez que alternam o contexto de uma página para outra. (TIDWELL, 2010) Além disso, a consistência da GUI significa que elementos presentes são organizados para apoiar os objetivos do site e a atenção do usuário, e a interface deve ser exibida de maneira adequada em todos os tipos de navegadores, como o Internet Explorer, o Google Chrome e o Firefox. (ABDO; ALALI, 2016)

O teste de GUI é uma etapa muito importante para o controle de qualidade de aplicativos de software. A GUI é o nó central no aplicativo de teste, de onde todas as funções são acessadas. Assim, é difícil testar completamente os programas por meio de sua GUI, especialmente porque elas são projetadas para trabalhar com humanos, não com máquinas de automação. Além disso, são interfaces inerentemente não estáticas, propensas a mudanças constantes causadas por atualizações de funcionalidade, usabilidade aprimorada, requisitos em mudança ou contextos alterados. Isso complica o desenvolvimento e a manutenção de casos de teste sem recorrer a testes manuais demorados e caros. (IVANOVA et al., 2020)

2.2 Testes Exploratórios

O presente capítulo discutirá o conceito e as práticas comumente aplicadas nos ET. Serão abordadas a definição dos ET, o seu surgimento e como ele se difere das práticas de testes mais tradicionais que seguem *scripts*. Serão elencados os principais pilares do processo como um todo. Também será apresentado quais técnicas podem ser aplicadas durante a execução dos testes para aumentar o desempenho dos mesmos. Além disso, pretende-se tratar também sobre a importância da experiência profissional e características pessoais do testador

responsável pela execução da atividade no resultado dos testes realizados. Por fim, será discutida a importância do controle do tempo alocado e utilizado durante a fase de exploração, para que a atividade não se torne exaustiva, longa e frustrante.

2.2.1 Surgimento dos Testes Exploratórios

Os ET são atividades de teste de *software* que compreendem uma abordagem especial dos testes manuais, e que diferem principalmente na forma como são executados. A ideia de executar testes com maior liberdade por parte do testador já existia há algum tempo, porém, era conhecida como testes *ad-hoc*, que consistem em um processo improvisado, onde não existe nenhuma diretriz ou missão principal e o papel do testador é basicamente interagir de forma arbitrária com o Sistema Sob Testes, do inglês *System Under Test* (SUT). O conceito de ET foi então introduzido por Cem Kaner no seu livro *Testing Computer Software* (KANER; FALK; NGUYEN, 1988), e anos depois foi, precisamente, definido por James Bach como aprendizagem, *design* de teste e execução de testes simultânea; ou seja, os testes não são definidos antecipadamente em um *script* de teste planejado e pré-estabelecido, mas são projetados dinamicamente, executados e modificados durante o processo de exploração do testador. (BACH, 2003)

Ainda segundo Bach, no início da década de 1990, um grupo de metodologistas começou a utilizar o termo “exploratório”, tentando com esta nova terminologia, enfatizar o processo intelectual envolvido nos testes que não seguiam roteiros com o objetivo de desenvolver tal prática em uma disciplina ensinável.

Foi demonstrado que o ET deve ser idealmente utilizado nas seguintes situações (KANER; BACH; PETTICHORD, 2001):

- Necessidade de fornecer feedback rápido sobre um novo produto ou serviço;
- Necessidade de aprender sobre um produto ou serviço rapidamente;
- Diversificar os testes e melhorá-los;

- Encontrar o erro mais importante no espaço de tempo mais curto possível;
- Investigar a situação de risco a fim de avaliar a necessidade de testes com *script* nessa área.

2.2.2 Estrutura dos Testes Exploratórios

Elisabeth Hendrickson, no livro *Explore It!*, utiliza uma adaptação da definição de ET por James Bach para explicar a prática da metodologia de testes em questão. Segundo a autora, o ET pode também ser definido como o projeto e execução de testes de forma simultânea para aprender sobre o sistema, de forma a usar as ideias obtidas no último experimento para informar o próximo. Cada parte dessa definição é importante: projetar testes, executá-los, aprender e orientar. (HENDRICKSON, 2013)

O projeto de teste envolve a identificação de coisas interessantes para variar e maneiras interessantes de variá-las. (HENDRICKSON, 2013) Uma estratégia básica do ET é ter um plano geral de ataque a ser seguido, mas o testador deve se permitir desviar dele por curtos períodos. Cem Kaner chama isso de princípio do “ônibus de turismo”. Mesmo as pessoas em um ônibus de turismo podem sair ocasionalmente e passear. A chave é não perder totalmente o passeio, nem adormecer dentro do ônibus. (BACH, 2003)

Em contraste com os testes de *script*, os testes são projetados conforme necessário, e executados no momento do design ou reutilizados posteriormente. Os testes são variados sempre que apropriado, e o ideal é projetar materiais de suporte com antecedência para que os mesmos possam ser usados durante os testes, como:

- Conjuntos de dados;
- Listas de modo de falha;
- Gráficos de combinação.

Testar sem *script* não significa estar despreparado. Trata-se de permitir a escolha e não a restringir. (KANER, 2008) Quanto mais familiarizado o testador estiver com o projeto predefinido de teste, melhor será capaz de projetar bons experimentos em tempo real. (HENDRICKSON, 2013)

Durante a exploração, o testador executa testes assim que pensa neles. O imediatismo da execução é um dos principais atributos que distinguem o ET de outras abordagens de teste com *script*. O fato de o testador não projetar os testes a serem realizados com antecedência antes de começar a executá-los é crucial. Até que o teste seja realizado, não se sabe quais perguntas de acompanhamento a ser investigadas irão ocorrer. A execução imediata permite que a investigação seja guiada de acordo com as informações mais interessantes. (HENDRICKSON, 2013) Durante esta etapa do ET, o testador interage com o SUT à procura de defeitos, a fim de verificar seu correto funcionamento e se as expectativas do cliente são atendidas, e reporta os resultados encontrados.

Conforme o testador explora a aplicação, ele descobre como o software funciona e suas peculiaridades, e esse novo conhecimento adquirido caracteriza a fase de aprendizado dos testes exploratórios. A exploração deve ser realizada com atenção e a observação é crucial e diretamente proporcional ao nível de aprendizado adquirido. O testador responsável precisa olhar além do que espera ver para compreender o que realmente está acontecendo. (KANER, 2008)

Dentre alguns exemplos de atividades de aprendizagem, cabe citar:

- Estudar produtos competitivos (como funcionam, o que fazem, quais expectativas eles criam);
- Pesquisar o histórico destes produtos relacionados a design, falhas e suporte;
- Inspecionar o produto em teste e seus dados, criar listas de funções, gráficos de relacionamento de dados, estruturas de arquivos, atividades do usuário, listas de produtos, benefícios do produto, etc.

- Levantar questionamentos. É importante identificar informações ausentes, deve-se analisar fontes e perguntas potencialmente reveladoras;
- Revisar a documentação, como especificações, documentos de projeto, etc.

A cada experimento realizado no SUT, o testador ganha um pouco mais de conhecimento sobre como o software se comporta. Se percebe então, com quais tipos de condições o software não lida bem e se usa esse conhecimento para se esforçar ainda mais nas próximas experimentações. A curiosidade deve ser alimentada pelo que foi aprendido até então, para sugerir a próxima informação mais interessante a ser descoberta. Realizar os testes enquanto se concentra nas informações mais importantes a serem descobertas é uma das principais habilidades de um bom explorador. (HENDRICKSON, 2013)

2.2.3 Técnicas de teste comumente aplicadas na fase de exploração

Existe uma vasta literatura que abrange técnicas de projeto de teste relevantes durante a exploração, incluindo clássicos como *The Art of Software Testing*, de Glenford Myers, e *Software Testing Techniques*, de Boris Beizer. Esses livros abordam técnicas aplicáveis durante a etapa de exploração. (HENDRICKSON, 2013) As técnicas de teste são comumente classificadas como caixa-preta, caixa-branca ou baseada na experiência.

As técnicas de teste caixa-preta, ou baseadas no comportamento, se fundamentam em uma análise da base de teste apropriada, como documentos de requisitos formais, especificações, casos de uso, histórias de usuários ou processos de negócios. Essas técnicas se concentram basicamente nas entradas e saídas do objeto de teste, sem nenhuma referência a sua estrutura interna. (OLSEN; POSTHUMA; ULRICH, 2019)

Alguns exemplos de técnicas de teste caixa-preta são o Particionamento de equivalência, que divide os dados em partições, de tal forma que todos os membros de uma determinada partição deve ser processado da mesma maneira; A Análise de valor limite, que é uma extensão do particionamento de equivalência, mas só pode ser usada quando a partição é ordenada, consistindo em dados numéricos ou sequenciais; As Tabelas de decisão, que são uma boa maneira de registrar regras de negócios complexas que um sistema deve implementar e identificam as entradas e as saídas do sistema; O teste de transição de estados que mostra os possíveis estados do *software*, bem como a forma como ele entra, sai e transita entre os estados; e os Testes derivados de casos de uso, que são uma maneira específica de projetar interações com itens de *software*, incorporando requisitos para as funções de *software* que são representadas pelos casos de uso. (OLSEN; POSTHUMA; ULRICH, 2019)

Já as técnicas de teste caixa-branca, também chamadas de técnicas estruturais, são baseadas em uma análise da arquitetura, do detalhamento do projeto, da estrutura interna ou do código do objeto de teste. Ao contrário do que se vê nas técnicas de teste caixa-preta, as técnicas de teste caixa-branca se concentram na estrutura e no processamento dentro do objeto de teste. As técnicas de teste caixa-branca mais conhecidas são o teste e cobertura de instruções, que testa todas as instruções executáveis do código; e o teste e cobertura de decisão, que testa todas as decisões existentes no código e o código executado com base nos resultados de cada decisão. (OLSEN; POSTHUMA; ULRICH, 2019)

Por fim, temos as técnicas de teste baseadas na experiência, que aproveitam o conhecimento dos desenvolvedores, testadores e usuários para projetar, implementar e executar os testes. Estas técnicas são frequentemente combinadas com técnicas de teste caixa-preta e caixa-branca durante a execução de ET. (OLSEN; POSTHUMA; ULRICH, 2019)

2.2.4 Relevância da experiência do testador durante a execução dos testes

Os profissionais da área de QA reconhecem que os aspectos exploratórios são fundamentais para a maioria das atividades de testes manuais. Há um número crescente de relatórios e estudos sobre os benefícios dos ET. Nesses relatórios, o ET é comumente descrito no contexto de testes em nível de sistemas interativos por meio da GUI e do ponto de vista do usuário final. (ITKONEN; MANTYLA; LASSENIUS, 2013)

Em estudos de testes manuais e, em particular, no contexto de ET, a experiência e principalmente o conhecimento do domínio de aplicação dos testadores, têm sido reconhecidos como aspectos importantes que afetam o comportamento e resultados do testador. (ITKONEN; MANTYLA; LASSENIUS, 2013)

O conhecimento pode ser aplicado a diferentes tarefas e propósitos do ET. Primeiro, o conhecimento pode ser usado como informação para orientar o projeto de ET. Em segundo lugar, o conhecimento pode ser usado para provocar e reconhecer falhas, ou seja, como um oráculo para distinguir entre um resultado esperado correto e um resultado incorreto e defeituoso. Terceiro, o conhecimento, juntamente com o comportamento real observado do sistema testado, pode ser usado para criar melhores testes durante os ET. (ITKONEN; MANTYLA; LASSENIUS, 2013)

2.2.5 Gerenciamento de tempo durante a fase de exploração do SUT

Explorar pode ser um experimento completamente aberto. Sem algum mecanismo para estruturar e organizar o esforço realizado, o testador pode passar

horas ou dias vagando sem rumo pelo software e acabar sem informações interessantes ou úteis para compartilhar. (HENDRICKSON, 2013)

Em resposta, Jon Bach e James Bach criaram a prática de Gerenciamento de Teste Baseado em Sessão, do inglês *Session Based Test Management* (SBTM), que consiste em um método projetado especificamente para tornar os ET auditáveis e mensuráveis em uma escala mais ampla. (BACH, 2003) Nesse método, o testador estrutura seu teste em uma série de sessões de tempo limitado. Deve-se estabelecer um foco para as sessões com antecedência, e durante cada uma delas, o testador explora com fluidez, projetando e executando testes, passando de um experimento para outro, sem pausa. (HENDRICKSON, 2013)

Em cada sessão, faz-se anotações para saber o que se explorou e quais informações foram encontradas. No entanto, essas notas são apenas para uso do responsável pela execução dos testes, que se referirá a elas ao conversar com as partes interessadas, mas elas não são como casos de teste tradicionais ou relatórios de teste. O testador pode fazer anotações sobre ideias de teste, perguntas, riscos, áreas adicionais que deseja explorar e *bugs* encontrados. (HENDRICKSON, 2013)

No final da sessão, as informações que precisam ser transmitidas aos outros devem ser organizadas. Deve-se elencar por escrito as observações sobre as capacidades e limitações das áreas que foram exploradas. Se houver dúvidas sobre o software, deve-se procurar alguém que possa respondê-las. As sessões fornecem pontos de parada periódicos para o testador destilar suas descobertas e considerar a melhor área a ser explorada a seguir. Se bugs foram encontrados, o ideal é que também sejam reportados. (HENDRICKSON, 2013)

ET não é um teste rápido. Um teste rápido, ou um “ataque”, é um teste que requer pouca preparação, conhecimento ou tempo para ser realizado. O teste rápido é uma técnica que parte de uma teoria do erro e gera testes otimizados para encontrar formas de quebrar o programa. Esse teste pode ser mais parecido com o teste com *script* ou mais como o ET, depende da mentalidade do testador.

O gerenciamento do tempo é a diferença fundamental entre testes exploratórios e com *script*. O explorador é sempre responsável por gerenciar o valor de seu próprio tempo, podendo fazer qualquer combinação de aprendizado, projeto, execução e interpretação, a qualquer momento. (KANER, 2008) isso pode incluir:

- Reutilização de testes antigos;
- Criação e execução de novos testes;
- Criação de artefatos de suporte de teste, como listas de modo de falha e etc;
- Realização de pesquisa aprofundada que pode orientar o projeto de teste.

2.3 Aprendizagem de Máquina

O campo da ML, como foi definido por Tom M. Mitchell, estuda algoritmos computacionais que se preocupam com a questão de como construir programas de computador que melhoram automaticamente com a experiência (MITCHELL, 1997). Dentro da ML existem ramificações que exploram diferentes técnicas de aprendizado e atendem diferentes propósitos.

Atualmente existem três grandes áreas dentro do campo de ML, são elas:

- Aprendizado Supervisionado;
- Aprendizado Não Supervisionado;
- Aprendizado por Reforço.

2.3.1 Aprendizado Supervisionado

O Aprendizado Supervisionado é baseado nas técnicas de regressão básica e classificação, é fornecido um banco de dados e a máquina deve aprender a reconhecer características a partir destas informações. Como por exemplo, aprender a reconhecer uma bicicleta, apesar de poder ter diferentes marcas, modelos, cores e tamanhos, a máquina consegue aprender que existem elementos essenciais a

serem identificados para classificar este objeto. Dentre essas características temos o par de pedais, as duas rodas, o guidão a cela, etc. (FREITAS, 2019)

Esse aprendizado é usado quando para cada variável de entrada existe uma variável de saída correspondente conhecida. Nesse cenário, com o objetivo de prever as saídas para entradas futuras ou entender melhor a relação entre a entrada e a saída e realizar classificações, um algoritmo é usado para aprender a função de mapeamento da entrada para a saída, ensinando a máquina o que deve ser feito. Esses algoritmos encontram maneiras de produzir a saída desejada com base nos pares de entradas e saídas desejadas fornecidas pelo usuário. (DURELLI et al., 2019)

2.3.2 Aprendizado Não Supervisionado

O Aprendizado não Supervisionado, em contraste com a técnica discutida anteriormente, consiste em apresentar dados não rotulados para a máquina sem ensinar ao modelo qual o objetivo final da execução. (WALTRICK, 2020) Portanto, esse modelo de aprendizagem ocorre quando apenas os dados de entrada estão disponíveis, com o objetivo de entender a relação implícita entre tais entradas. (DURELLI et al., 2019)

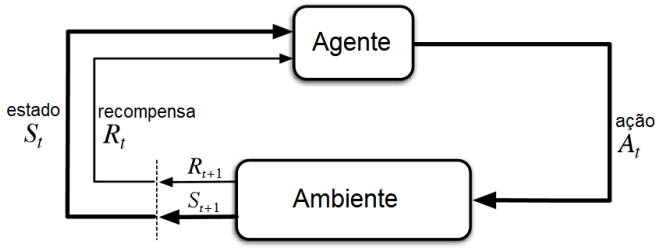
Esse tipo de aprendizado é baseado na busca e reconhecimento de padrões e relacionamentos em problemas de agrupamento, onde se busca similaridades entre os dados com o objetivo de determinar se as entradas se enquadram em grupos distintos, e de modo geral, é usado para se encontrar peculiaridades nos dados dispostos. Também é utilizado em problemas de associação, quando os dados apresentam sequências e se deseja encontrar padrões nas mesmas, e em problemas de redução de dimensão, onde se diminui a quantidade de dados observáveis eliminando as ocorrências aleatórias, objetivando ter um conjunto menor de variáveis principais. (WALTRICK, 2020)

2.3.3 Aprendizado por Reforço

O RL envolve ensinar um agente o que fazer mapeando situações para ações de modo a maximizar um sinal numérico de recompensa. Além disso, o agente não é informado sobre quais ações realizar, como nas outras formas de aprendizado de máquina, mas deve descobrir quais ações geram mais recompensas ao experimentá-las. (SUTTON; BARTO, 2018) Dada a estrutura do problema, o objetivo dos algoritmos de RL é encontrar políticas que maximizem a recompensa acumulada pelo agente. (FIGUEIREDO; REJAILI, 2018) Diferentemente dos tipos de aprendizado que foram discutidos anteriormente, o RL não está relacionado apenas a casos que disponham exclusivamente de bases de dados. Além disso, esse aprendizado é útil também em situações em que existe um ambiente para se lidar, como por exemplo um cenário em um jogo de videogame. (WALTRICK, 2020)

Analisando a figura 3, percebe-se que é possível aplicar o conceito de RL no ET. O testador assume o papel do agente que realiza ações em um ambiente, este representado pela interface gráfica da aplicação testada, fazendo com que os resultados dessas ações tomadas na tela da aplicação, equivalentes aos resultados das execuções dos testes, resultem em uma recompensa ou punição que vai auxiliar a construir um aprendizado e aprimorar as decisões de quais ações tomar futuramente.

Figura 3 – Framework básico de RL



Fonte: Extraído de (NEVES, 2020).

2.3.3.1 Estrutura básica

O problema do RL pode ser formalizado usando as ideias da teoria de sistemas dinâmicos, especificamente, como o controle ótimo de processos de decisão Markov. A ideia básica é apenas capturar os aspectos mais importantes do problema real com o qual o agente está lidando enquanto ele interage ao longo do tempo com seu ambiente para atingir um objetivo. Um agente deve ser capaz de reconhecer o estado de seu ambiente até certo ponto e deve ser capaz de realizar ações que afetam o estado. O agente também deve ter um ou mais objetivos relacionados ao estado do ambiente. Qualquer método que seja adequado para resolver tais problemas, é considerado um método de RL. (SUTTON; BARTO, 2018)

2.3.3.2 Algoritmos de AM por Reforço

De todas as formas de aprendizado de máquina, o RL é o mais próximo do tipo de aprendizado que humanos e outros animais fazem, e muitos dos principais algoritmos de RL foram originalmente inspirados em sistemas de aprendizado biológico. O componente mais importante de quase todos os algoritmos desse aprendizado é um método para estimar valores de forma eficiente. O papel central da estimativa de valor é sem dúvida a coisa mais importante que foi aprendida sobre a área nas últimas seis décadas. (SUTTON; BARTO, 2018)

Existe uma ampla variedade de algoritmos de RL, incluindo o *Upper Confidence Bound* (UCB), o *State-Action-Reward-State-Action* (SARSA), o *Q-Learning*, o *Monte Carlo Tree Search* (MCTS), entre outros. Existe ainda a Programação Dinâmica, do inglês *Dynamic Programming* (DP) que se refere a uma coleção de algoritmos, como o de Programação Dinâmica em Tempo Real, do inglês *Real-Time Dynamic Programming* (RTDP), que podem ser usados para calcular políticas ótimas dado um modelo perfeito do ambiente como um Processo de Decisão de Markov, do inglês *Markov Decision Process* (MDP). Os algoritmos clássicos de DP são de utilidade limitada no RL, tanto por conta de sua suposição de um modelo perfeito quanto por causa do alto custo computacional no processamento, mas ainda são importantes teoricamente. (SUTTON; BARTO, 2018)

Duas características fundamentais são encontradas em um sistema de RL, o aprendizado por interação e o retorno atrasado.

O Aprendizado por Interação consiste no aguardo pelo valor de reforço que o ambiente deve retornar em resposta à última ação tomada pelo agente no próprio ambiente. Já o Retorno atrasado considera que um valor de reforço alto recebido pelo agente ao executar uma determinada ação em um estado não deve significar que a ação escolhida pelo agente é a recomendada. Uma ação é apenas um produto de uma decisão local no ambiente, mas de forma geral, o intuito do agente é alcançar objetivos globais. A qualidade das ações tomadas deve ser mensurada a partir das soluções a longo prazo. (LACERDA, 2013)

2.3.4 Aplicação na Indústria

As técnicas de ML permitem a geração de inteligência acionável na indústria, de forma a processar os dados coletados para aumentar a eficiência da fabricação sem alterar de forma significativa os recursos necessários. Além disso, a capacidade das técnicas de ML de fornecer entendimento para realizar predições permitiu discernir padrões de fabricação complexos e oferece um caminho para um sistema de suporte à decisão inteligente em uma variedade de tarefas de fabricação. Dentre estas tarefas estão a inspeção inteligente e contínua, a manutenção preditiva, a melhoria de qualidade, a otimização de processos, o gerenciamento de cadeia de fornecimento e o agendamento de tarefas. (RAI et al., 2021)

O monitoramento das condições de atuação aliados à manutenção preditiva dos equipamentos elétricos utilizados pela indústria evita graves perdas econômicas, que costumeiramente resultam de falhas inesperadas, melhorando muito a confiabilidade do sistema. Os métodos automatizados oferecem uma solução viável para muitas empresas detectando e coletando informações confidenciais de equipamentos de forma mais eficaz do que humanos podem fazer. A manutenção preditiva com o auxílio de sensores integrados pode evitar a substituição desnecessária de equipamentos, reduzir o tempo de inatividade da máquina,

encontrar a causa raiz da falha e, dessa forma, economizar custos e melhorar a eficiência. (PAOLANTI et al., 2018)

Os algoritmos de RL também são usados em diversas aplicações no mercado. Apesar de serem comumente vinculados à utilização em jogos digitais, existem aplicações reais que valem ser destacadas. Como por exemplo, o *Microsoft Office*, que usa a técnica de RL para melhorar as sugestões que são realizadas para os usuários nos seus aplicativos. (LANGSTON, 2020)

2.3.5 Dificuldades e Limitações atuais

Existem limitações conceituais, processuais e estatísticas específicas de modelos de ML quando aplicados à sociedade. Os próprios modeladores de ML devem questionar constantemente para identificar possíveis pontos de falha e pensar em como resolvê-los, e os consumidores de modelos de ML podem saber o que questionar quando confrontados com a decisão sobre se, onde e como aplicar o ML. As limitações vão desde compromissos inerentes à própria quantificação, até mostrar como dependências não modeladas podem levar a validação cruzada a ser excessivamente otimista como forma de avaliar o desempenho do modelo. (MALIK, 2020)

Muitas das limitações identificadas estão associadas ao uso apenas de ferramentas de ML, o que pode, portanto, ser visto mais como um problema cultural entre as pessoas que fazem e usam ML do que limitações intrínsecas atribuíveis ao próprio campo. É importante entender que o ML não é uma solução completa, ao usar métodos mistos e incluir outras abordagens paralelamente para determinados problemas, muitas dessas limitações podem ser evitadas. (MALIK, 2020)

A falta de conhecimento profundo dos diferentes métodos de ML, seus requisitos específicos, bem como as limitações dos métodos individuais, muitas vezes podem ser um obstáculo para o uso correto dos métodos. (ECKART, L.; ECKART, S.; ENKE, 2021)

Levando em consideração as limitações mencionadas, é possível perceber que nem todos os problemas podem ser resolvidos com o uso de técnicas de ML. Assim como qualquer outra técnica dentro da ML, os algoritmos de RL são voltados para um determinado nicho de problemas e são extremamente eficientes quando aplicados a eles.

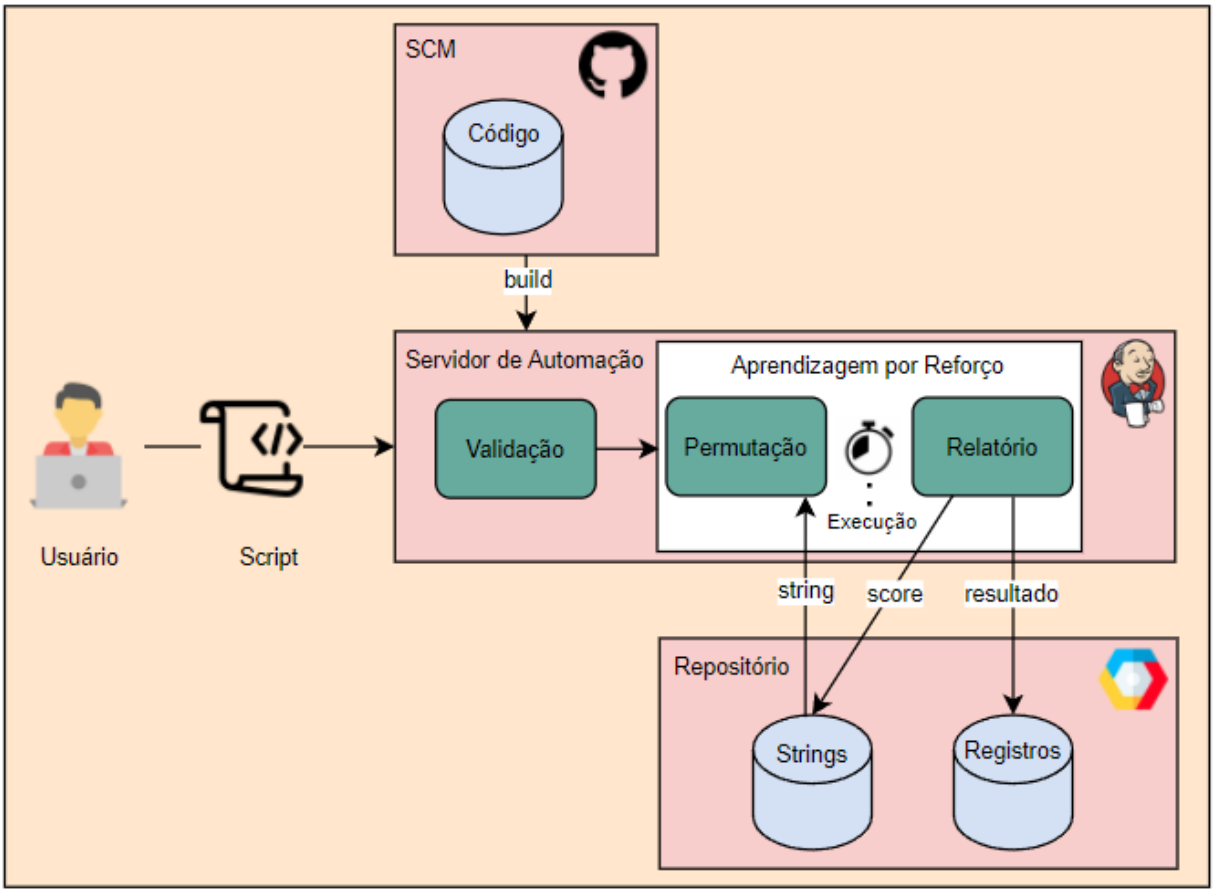
3 Automação de Testes Exploratórios em Sistemas com GUI

Neste capítulo será discutido, em detalhes, o funcionamento e as práticas adotadas no processo de desenvolvimento da aplicação que consiste no principal objeto deste trabalho. A integração entre o *script* principal, o *Google Cloud Datastore*, o *GitHub* e o *Jenkins* é explicada e fica clara a importância de cada um dos sistemas no resultado obtido. A lógica de reforço e como as recompensas e punições foram implementadas também serão discutidas, além das decisões tomadas buscando o melhor resultado dos testes realizados.

3.1 Descrição da aplicação

A aplicação desenvolvida no presente trabalho consiste na implementação de um código capaz de executar algumas das tarefas necessárias para a realização do ET de uma maneira mais independente da presença do testador. O código principal, escrito em *Python*, é responsável por interpretar, validar e modificar *scripts* de teste implementados também em *Python* seguindo os passos definidos na metodologia de Testes Guiados por Dados, do inglês *Data-Driven Testing* (DDT) (CHANDRAPRABHA; SAXENA, 2015; JHA; GULATI, 2018), além de reforçar a aprendizagem no processo de exploração, recompensando ou punindo as *Strings* utilizadas nas execuções dos *scripts* de teste automatizados fornecidos como entrada. A estrutura simplificada da aplicação pode ser vista na figura 4 a seguir.

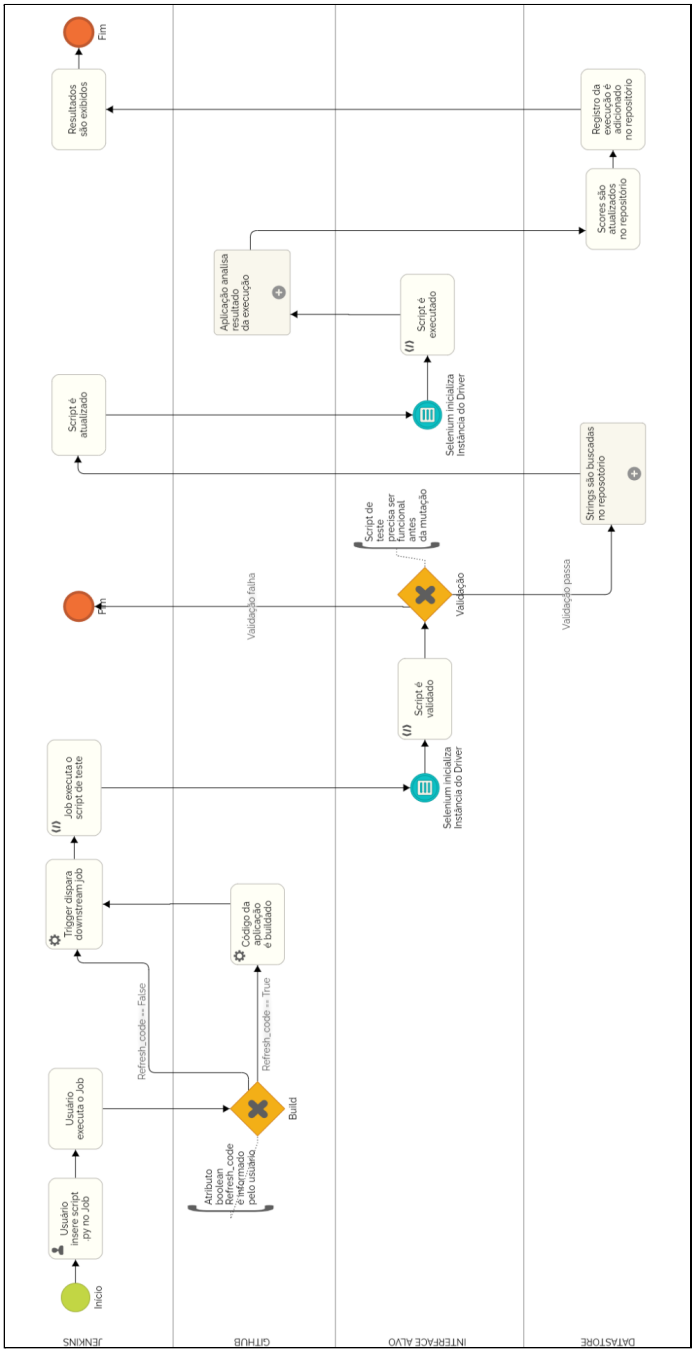
Figura 4 – Estrutura geral da aplicação



Fonte: O autor

O fluxo de execução do processo estabelecido pela aplicação desenvolvida foi modelado e pode ser analisado na figura 5 para um melhor entendimento da estrutura mostrada acima.

Figura 5 – Diagrama de modelagem da aplicação

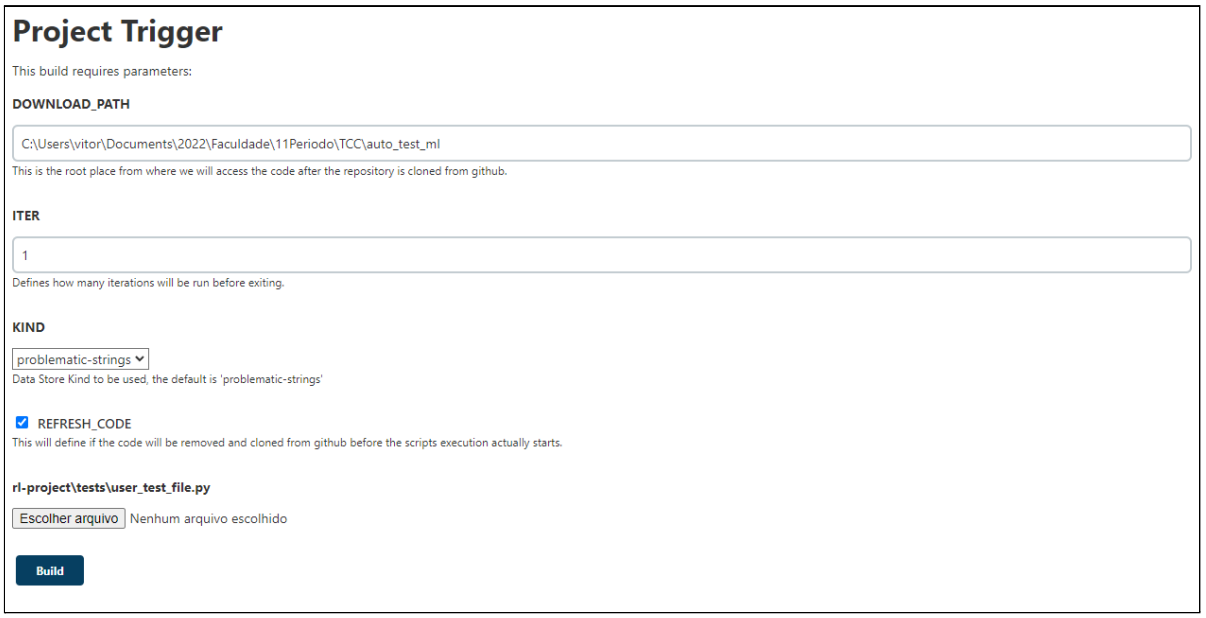


Fonte: O autor

3.2 Estrutura básica

O código principal fica armazenado em um repositório do *GitHub* que foi criado e utilizado para facilitar o versionamento durante o desenvolvimento da aplicação e possibilitar a realização de alterações de forma rápida, além de permitir o acesso remoto ao código implementado. A interface de apresentação e uso da aplicação para os usuários, como pode ser visto na figura 6, é a própria interface do *Jenkins*, que possui um *job* responsável por receber e enviar os parâmetros de entrada inseridos pelo usuário para a aplicação, acessando o repositório e clonando o seu conteúdo para a execução do código mais recente da aplicação. Dentre os parâmetros que a aplicação espera que sejam enviados através do *Jenkins*, estão a quantidade de iterações em que o mesmo deverá ser executado e qual o repositório de *Strings* do *Datastore*, também conhecido como *kind*, deverá ser utilizado.

Figura 6 – Interface de execução da aplicação

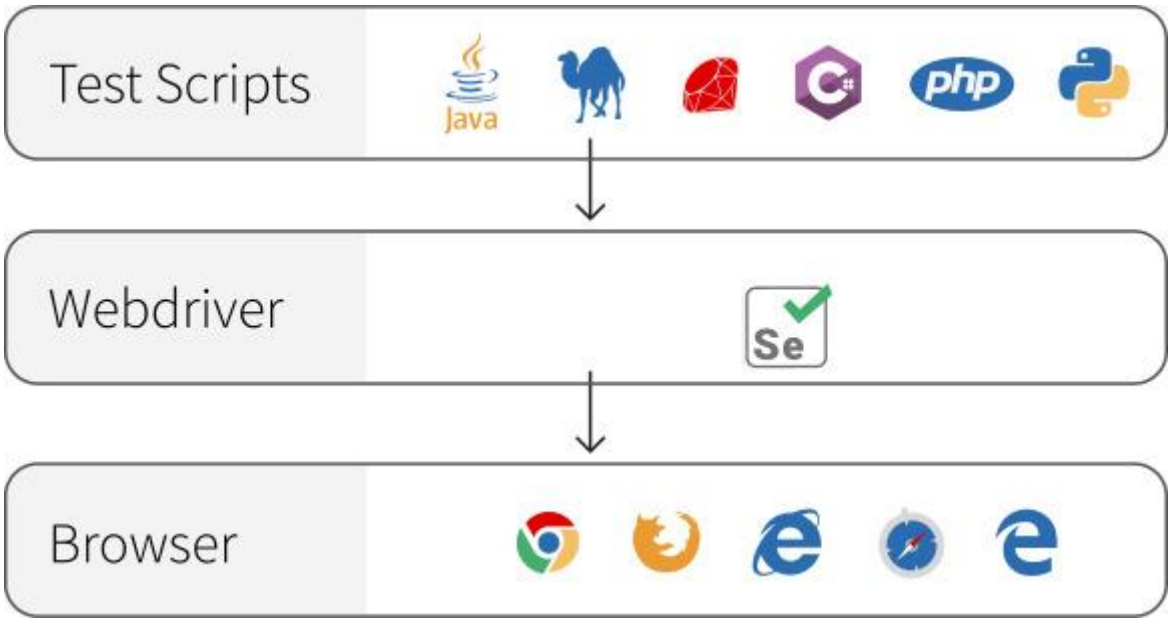


Fonte: O autor

3.3 Validação

A estrutura montada possui um código principal que fica na classe *main.py* localizado na pasta raiz do projeto. Assim que a aplicação é inicializada, passando-se um *script* de teste como parâmetro de entrada, o código é responsável por validar o *script* fornecido, realizando a sua execução através do *framework Pytest*. O *script* de teste inserido na aplicação deve contar com o auxílio dos métodos do *Selenium WebDriver* para realizar a comunicação com os elementos disponíveis na página do *website* alvo através dos navegadores de internet, essa biblioteca, como mostra a figura 7 abaixo, possibilita o mapeamento e interação com botões, *links*, campos de entradas de texto, que serão exploradas na aplicação desenvolvida, entre outros.

Figura 7 – Fluxo de funcionamento do *Selenium WebDriver*



Fonte: Extraído de (RANOREX, c2022).

O resultado da validação é fornecido pelo próprio *framework* através do *exit_code*, que corresponde à uma saída numérica obtida ao fim da execução. O sistema analisa então o código retornado, a validação é concluída com falha se o mesmo for diferente de 0, o que significa que houve erros durante a execução do

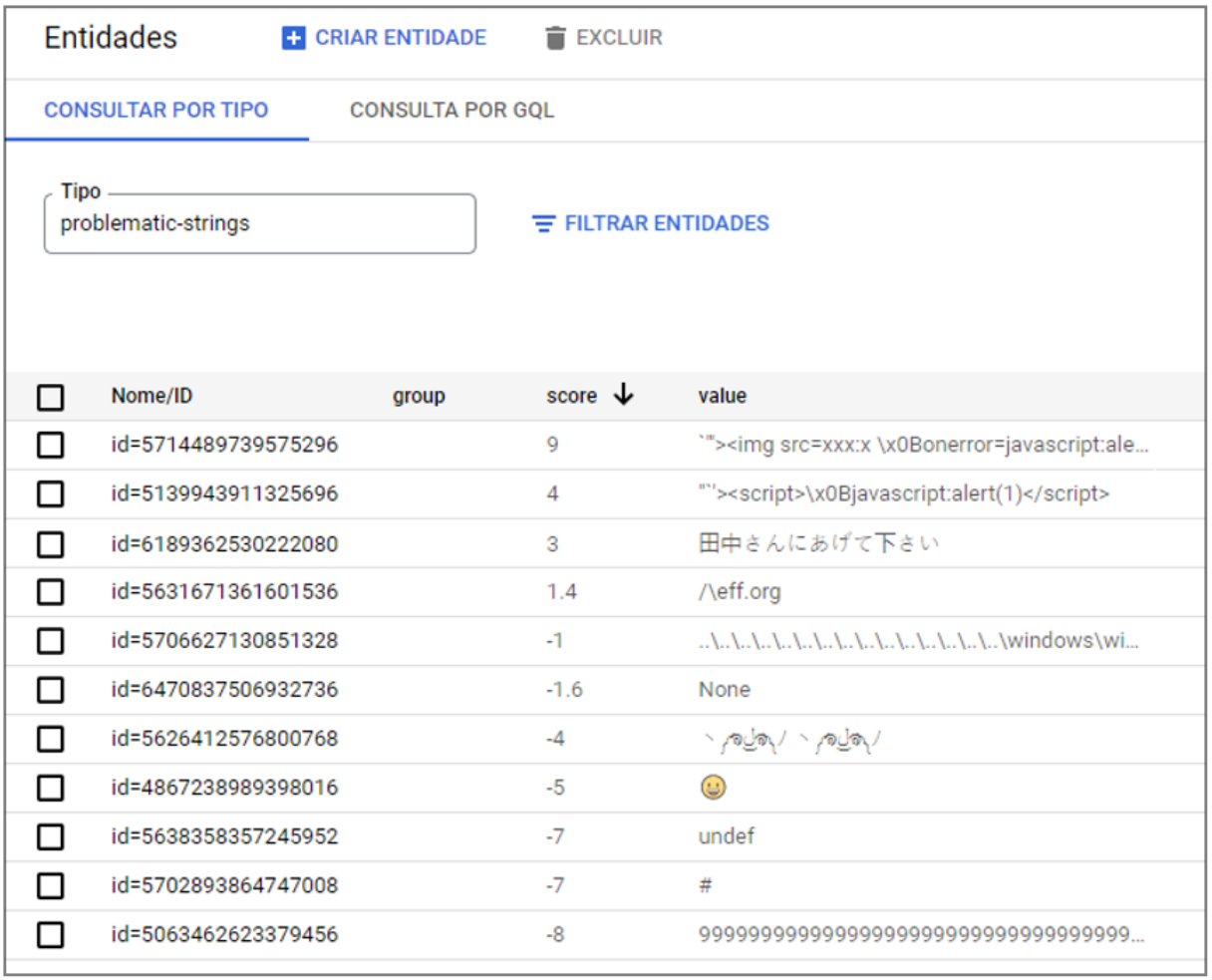
script fornecido pelo usuário da aplicação, encerrando desta forma, à execução da mesma. Caso o código retornado pelo *Pytest* seja igual à 0, significa que não houveram erros durante a execução do *script Python* fornecido, portanto ele é considerado como funcional, e o sistema está apto a prosseguir para as fases seguintes.

3.4 Modificação do script

A etapa subsequente à validação corresponde à permutação de trechos do código original do *script* de teste utilizado. Através de uma varredura no *script* e uma expressão regular, os métodos comumente utilizados para realizar a inserção de caracteres textuais em campos de entrada de dados em websites, como o *send_keys()* e o *execute_script()*, são procurados ao longo do código. Caso existam ocorrências dos métodos descritos, os índices de cada linha onde se localizam os métodos encontrados são armazenados em uma lista de inteiros que servirá como um mapa durante a modificação do código do *script*. Ao menos uma ocorrência dos métodos mencionados precisa existir, caso não haja nenhum número identificador de linha para ser adicionado na lista, a execução da aplicação é encerrada.

Antes do início da modificação do *script* de teste, os objetos textuais a serem utilizados nesta etapa são buscados no repositório de *Strings* localizado no *Google Cloud Datastore*, que pode ser visto na figura 8. Cada *String* armazenada no repositório contém um conjunto de atributos que às dão características decisivas em suas utilizações. Todas as *Strings* possuem um *ID*, que corresponde a uma sequência numérica que as identifica única e exclusivamente; possuem também um *group*, que pode ser utilizado para agrupar *Strings* com características semelhantes; além disso, possuem um *score*, que é o atributo numérico responsável por definir o seu potencial de revelar falhas em interfaces de *websites*; e, por fim, possuem o *value*, que corresponde aos caracteres que de fato a compõem.

Figura 8 – Interface do repositório de *strings*



Nome/ID	group	score ↓	value
id=5714489739575296		9	`"><img src=xxx:x \x0Bonerror=javascript:ale...
id=5139943911325696		4	"'><script>\x0Bjavascript:alert(1)</script>
id=6189362530222080		3	田中さんにあげて下さい
id=5631671361601536		1.4	/\eff.org
id=5706627130851328		-1	..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\..\windows\wi...
id=6470837506932736		-1.6	None
id=5626412576800768		-4	ヽ(ಠ_ಠ)ノ ヽ(ಠ_ಠ)ノ
id=4867238989398016		-5	😀
id=5638358357245952		-7	undef
id=5702893864747008		-7	#
id=5063462623379456		-8	99999999999999999999999999999999...

Fonte: O autor

A integração com a ferramenta de armazenamento de dados é feita através de métodos auxiliares que estão localizados no arquivo *util.py*, situado dentro da pasta *util* na raiz do projeto. A comunicação é estabelecida através de credenciais do usuário ainda na fase de inicialização da aplicação, possibilitando a leitura e escrita de novos dados e a edição dos já existentes no repositório associado. Através da integração, é feita uma busca pelo conjunto das 5 *Strings* que possuem o maior valor numérico no atributo *score*, as mesmas são armazenadas em uma lista que posteriormente é utilizada para fornecê-las como entradas substitutas na aplicação principal.

A definição de qual *String* será inserida e utilizada na execução atual do *script* de teste é feita de forma simples e leva em consideração o valor numérico do parâmetro *Iter*, que é informado na inicialização da aplicação principal. O parâmetro

mencionado, fica conhecido como *round*, e representa a quantidade de iterações de execução do teste desejada pelo usuário; caso o mesmo possua mais tempo para a utilização do sistema, é possível informar um número mais alto de iterações que tornarão a aprendizagem mais precisa e aumentarão as chances de falhas serem reveladas no decorrer de cada *round* de execução. As execuções do teste que acontecem em *rounds* ímpares usam como entrada o conteúdo da lista com as *Strings* de maior *score*. Já as execuções dos *rounds* pares utilizam um método de busca aleatória por *Strings* no repositório, reduzindo a chance de vícios na utilização de *Strings* repetidas e dando uma chance igualitária para todas as *Strings* existentes, estimulando assim, a exploração de todo o conteúdo de texto armazenado disponível. As *Strings* selecionadas nesta etapa, deverão substituir o texto presente nos métodos contidos no *script* de teste original durante a permutação dos dados em cada *round* de execução.

Antes da efetivação da substituição das *Strings* no texto do código, é realizado um tratamento manual das *Strings* recuperadas no repositório, a fim de contornar caracteres especiais que estão presentes, como o ‘ “ ‘, a ‘ \ ‘, o ‘ \w ‘, o ‘ \. ’ ou o ‘ \e’, que se não forem tratados desencadeiam em erro durante sua escrita no código. Portanto, cada caractere especial mencionado recebe um caractere de escape, o ‘ \ ’ é usado para identificar um único caractere numa cadeia de caracteres que altera o significado de seu sucessor. No fim da etapa de tratamento das *Strings*, é realizada uma codificação padrão dos caracteres para *bytes* no formato *Unicode Transformation Format 8-bit* (UTF-8) e, logo após, uma decodificação especificando o esquema de codificação *unicode-escape*, para obter rapidamente uma representação do Código Universal, do inglês *Universal Coded* (UNICODE) literal sem escape nos caracteres.

Uma vez que o tratamento dos caracteres presentes na *String* a ser utilizada no *round* atual esteja finalizado, é o momento de substituí-la no *script* de teste e algumas condições precisam ser respeitadas nesta etapa. Se a lista com o mapa de linhas que identificam os métodos de inserção possuir 2 ou mais elementos, no estudo de caso desenvolvido no presente trabalho utilizando o *JIRA Software* como

website alvo dos testes, os 2 elementos estarão relacionados aos campos de entrada de texto do *login*, para a informação de usuário e senha, e, portanto, não serão incluídos nos campos onde a permutação é permitida. Em trabalhos futuros este comportamento pode ser abstraído para tornar a aplicação mais genérica e aumentar suas possibilidades de uso. Logo, caso o *script* de teste possua *x* campos de entrada de texto, onde $x > 2$, a permutação acontecerá nos $x-2$ campos restantes. A escolha do campo a ter seu valor alterado é feita de forma aleatória e individual, portanto, apenas um campo por *round* é modificado, e a cada rodada, o *script* alterado gerado na rodada anterior é utilizado como *script* base, fazendo com que ao passar dos *rounds* os campos possam ser explorados por combinações de *Strings* diferentes.

3.5 Execução

Ao fim da etapa de permutação, se não houver erros inesperados no tratamento da *String* que resultem na interrupção da mutação atual e pulo para o próximo *round*, um novo arquivo *.py* é gerado. Ele representa o *script* original com uma modificação no valor que deverá ser passado a um campo de entrada de texto no *website* alvo. O novo *script* deverá ser executado através do *pytest* e o código de saída de sua execução é armazenado. Outras informações relevantes da execução também são salvas para posteriormente serem inseridas em outro repositório do *Datastore*, responsável por armazenar dados como resultado, tempo de execução e a *url* de execução do *job* no *Jenkins*, que são indispensáveis para o rastreamento de cada *round* executado.

3.6 Análise de códigos HTTP no console logs

Durante a execução do *script* de teste alterado, a aplicação é responsável por analisar e salvar em um arquivo de texto os *logs* de *javascript* registrados no *console* do navegador onde o teste está sendo executado. Apenas logs com nível *SEVERE* são armazenados dada a sua relevância para a análise da execução. Após o

salvamento desses dados, uma varredura no arquivo de texto é feita através de um método no arquivo *util.py*, e uma flag é disparada na aplicação informando, se existir, o código *HyperText Transfer Protocol* (HTTP) mais relevante presente nos logs armazenados. A lista de 5 códigos HTTP selecionados para serem analisados foi criada baseando-se na posição de cada código no ranking de erros mais buscados no Google na última década. Os 5 mais buscados correspondem aos erros:

- 401 (*Unauthorized*)
- 400 (*Bad Request*)
- 404 (*Not Found*)
- 403 (*Forbidden*)
- 500 (*Internal Server Error*)

O erro 401 *Unauthorized* ocorre, por exemplo, quando um visitante de um *website* tenta acessar uma página restrita sem possuir a autorização necessária para acessá-la, geralmente acontece por conta de uma tentativa falha de login.

O segundo erro da lista, o 400 *Bad Request*, é basicamente uma mensagem de erro do servidor informando que a aplicação utilizada para o acessar, como o navegador, tentou realizar o acesso de forma incorreta ou que a solicitação foi de alguma forma corrompida no caminho.

Já o erro 404 *Not Found* é um dos mais comumente encontrado nos websites, ele acontece quando o usuário tenta acessar um recurso em um servidor, como uma página de um website, que não existe. Alguns motivos para isso acontecer podem ser, por exemplo, um *link* quebrado, uma *url* digitada incorretamente ou o gerente do servidor moveu a página solicitada para outro lugar, ou a excluiu. Para combater o efeito nocivo de *links* quebrados, alguns sites configuram páginas personalizadas para serem exibidas quando um erro 404 é detectado.

O erro 403 *Forbidden* é semelhante ao erro 401 *Unauthorized*, mas neste caso, o usuário não possui nem a chance de tentar uma interação de autenticação na página, no exemplo do login, a página para inserir as credenciais não estaria

disponível. Esse erro pode acontecer, por exemplo, se o usuário tentar acessar um diretório proibido em um site.

E o erro HTTP mais comum de todos é o 500 *Internal Server Error*, ou erro interno de servidor, e o próprio nome é autoexplicativo. É uma mensagem de erro de uso geral para quando um servidor da web encontra algum tipo de erro interno. Por exemplo, o servidor pode estar sobrecarregado com muitos acessos simultâneos e, portanto, incapaz de lidar com as solicitações corretamente. Situação bastante comum em *e-commerces* durante períodos promocionais, por exemplo.

Já a ordem de prioridade dos códigos HTTP foi desenvolvida levando em consideração a gravidade de cada erro para a execução do website alvo dos testes, tendo o erro "401" e o erro “500” como o menos prejudicial, e o mais grave dentre os mencionados, respectivamente.

3.7 Reforço

No código principal da aplicação, o reforço é aplicado na *String* participante do *round* de acordo com o código de saída de sua execução e a *flag* retornando um dos 5 códigos de erro HTTP mencionados anteriormente ou retornando 0, caso nenhum dos erros descritos seja encontrado no arquivo de texto temporariamente criado. A combinação de possíveis valores dessas duas variáveis define quanto será o valor da recompensa ou punição da *String* analisada, e é feita da seguinte forma:

- + 1.0 - Recompensa; se o código de saída do *pytest* != 0 (falha) e *flag* = 500 (*Internal Server Error*).
- + 0.5 - Recompensa; se o código de saída do *pytest* != 0 (falha) e *flag* = 403 (*Forbidden*).
- + 0.4 - Recompensa; se o código de saída do *pytest* != 0 (falha) e *flag* = 404 (*Not Found*).
- + 0.3 - Recompensa; se o código de saída do *pytest* != 0 (falha) e *flag* = 400 (*Bad Request*).

- + 0.2 - Recompensa; se o código de saída do *pytest* != 0 (falha) e *flag* = 401 (*Unauthorized*).
- - 0.6 - Punição; se o código de saída do *pytest* = 0 (sucesso) e *flag* = 500 (*Internal Server Error*).
- - 0.8 - Punição; se o código de saída do *pytest* = 0 (sucesso) e *flag* = 403 (*Forbidden*).
- - 1.0 - Punição; se o código de saída do *pytest* = 0 (sucesso) e *flag* = 404 (*Not Found*).
- - 1.2 - Punição; se o código de saída do *pytest* = 0 (sucesso) e *flag* = 400 (*Bad Request*).
- - 1.4 - Punição; se o código de saída do *pytest* = 0 (sucesso) e *flag* = 401 (*Unauthorized*).

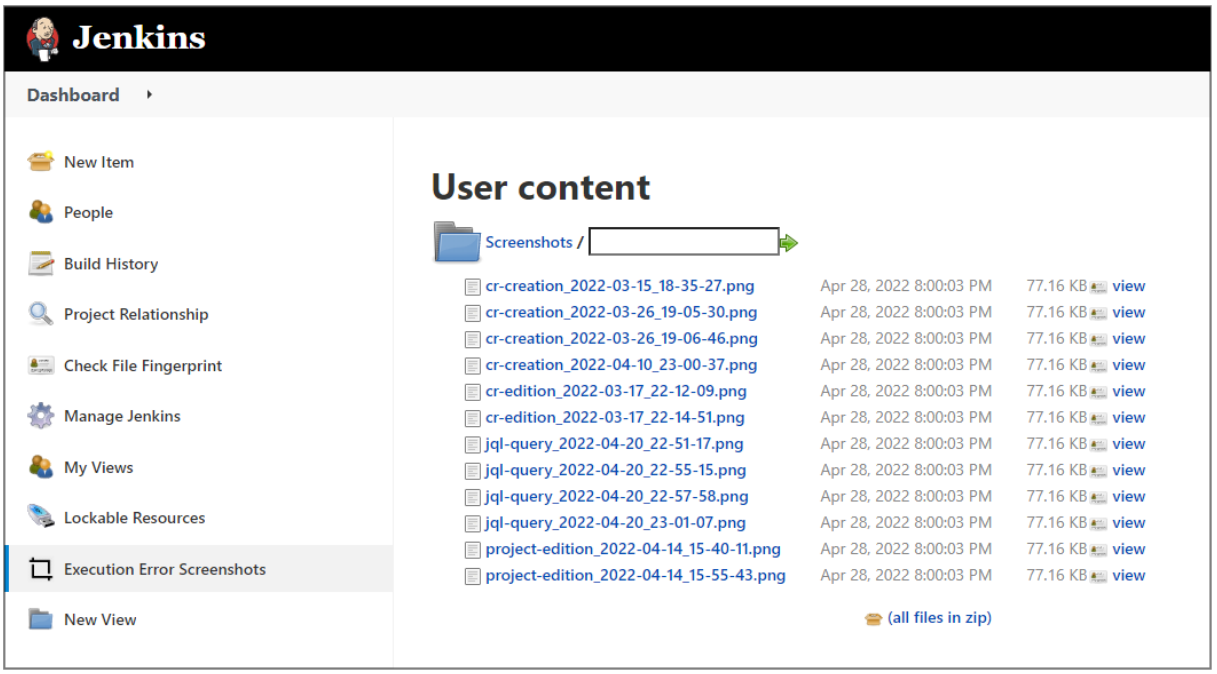
Após a definição do valor de reforço a ser aplicado, o novo valor do atributo *score* da *String* utilizada é persistido no repositório do *Datastore* através do método *update_data* localizado no arquivo *util.py*, além disso, as informações da execução do *round* atual são adicionadas em um dicionário de dados e o próximo *round* se inicia. Ao fim da execução de todos os *rounds* solicitados pelo usuário da aplicação, o dicionário contendo o registro de todas as execuções dos *scripts* mutados é inserido no repositório *execution-records* do *Datastore* e a execução do sistema termina.

3.8 Resultados

A visualização do resultado dos testes se dá através da interface do console do *Jenkins*, com informações detalhadas e os *logs* da execução de cada rodada, incluindo os possíveis erros que podem ocorrer durante a execução. As *screenshots* de cada erro ficam armazenadas no *Jenkins* e são disponibilizadas para o usuário através da pasta *Screenshots*, que pode ser acessada a partir da tela inicial, clicando na opção “Execution Error Screenshots”, que pode ser vista na figura 9. Cada *screenshot* pode ser visualizada na própria interface do *Jenkins* e baixada

individualmente, ou todas as imagens podem ser baixadas de uma vez e comprimidas em formato *.zip*.

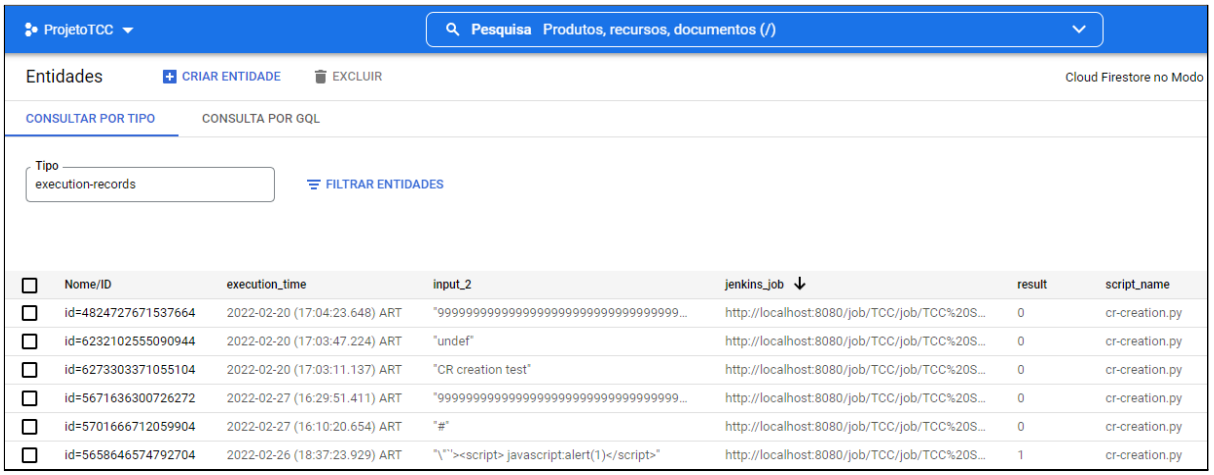
Figura 9 – Interface do repositório de *screenshots* no *Jenkins*



Fonte: O autor

E por fim, o usuário tem acesso aos detalhes e rastreamento das execuções falhas e de sucesso através do *kind execution-records* no *Datastore*, que disponibiliza informações importantes de cada *round* de execução, como o nome do *script* executado, o tempo de execução, cada uma das *Strings* utilizadas durante o *round*, a *url* do *job* no *Jenkins* e o resultado da execução. A organização dos resultados registrados pode ser vista na figura 10.

Figura 10 – Interface do repositório de registro das execuções



Fonte: O autor

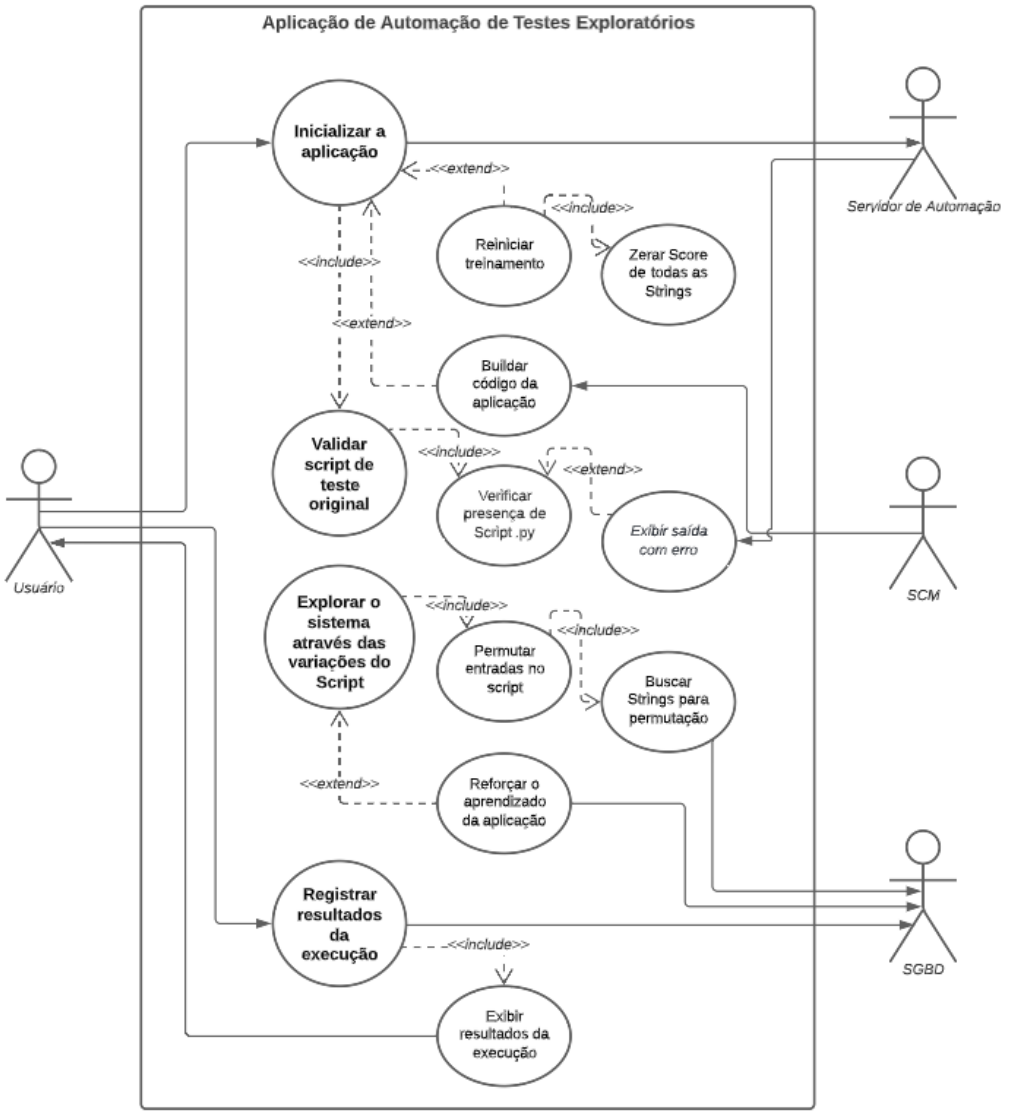
Caso o usuário deseje iniciar um novo treinamento com um novo *script* para um sistema alvo diferente do anteriormente utilizado, isso se faz possível com a limpeza do atributo score de todas as *Strings* disponíveis no repositório. O usuário consegue acionar o método presente no *script util.py*, que irá resetar todo o aprendizado acumulado até então através de uma opção na inicialização da aplicação no *Jenkins*. Também é possível realizar a limpeza completa do *kind execution-record*, que apesar de ser uma operação irreversível, auxiliará a não poluir visualmente os dados importantes em caso de um recomeço do treinamento.

4 Casos de Uso

Os casos de uso descrevem um comportamento que o software a ser desenvolvido apresentará quando estiver pronto (MEDEIROS, 2004). O presente capítulo detalha os casos de uso desenvolvidos para a aplicação construída.

4.1 Diagrama de casos de uso

Figura 11 – Diagrama de Casos de Uso



Fonte: O autor

4.2 Atores

4.2.1 Usuário (Primário)

Ator responsável por iniciar a execução da aplicação e prover os dados necessários para que a execução seja bem-sucedida. O Usuário deve informar dados como: O *script* a ser utilizado; a quantidade de rodadas a serem executadas e a instrução de retreino caso esse seja o objetivo.

4.2.2 Servidor de automação (Secundário)

Ator responsável por prover a interface gráfica capaz de gerenciar e fornecer os dados do usuário para a aplicação desenvolvida. O servidor de automação utilizado, *Jenkins*, possibilita que o usuário inicialize a execução da aplicação, além de exibir possíveis saídas de erro ao longo do processo, e exibir também o resultado da execução.

4.2.3 Gerenciamento de Código Fonte (Secundário)

Ator responsável por disponibilizar o código fonte atualizado da aplicação desenvolvida para ser executado no servidor de automação. A ferramenta de Gerenciamento de Código Fonte, do inglês *Source Code Management* (SCM) utilizada é um repositório no GitHub que controla todas as versões da aplicação desde o início da etapa de desenvolvimento.

4.2.4 Repositório (Secundário)

Ator responsável por armazenar todas as informações importantes da aplicação. O repositório no *Google Cloud Datastore* guarda todas as *Strings* utilizadas nas permutações durante a execução, assim como registra todos os dados gerados pela aplicação, de forma a manter a rastreabilidade das operações.

4.3 [UC001] - Inicializar a aplicação

4.3.1 Atores

Usuário (Principal) e *Jenkins* (Secundário)

4.3.2 Pré-Condições

O usuário está logado no *Jenkins*.

4.3.3 Fluxo Principal

1. O usuário acessa o Job Trigger no Jenkins que exibe a lista de jobs existentes;

2. O usuário clica no botão “*Build With Parameters*”;

3. O usuário insere um arquivo .py;

4. O usuário informa a quantidade de iterações de execução do *script*;

5. O usuário seleciona qual o repositório de *Strings* deverá ser utilizado na execução;

6. O usuário clica no botão “*Build*”;

7. O caso de uso é encerrado.

4.3.4 Fluxo Alternativo

1. (2-5) O usuário marca a opção “REFRESH_CODE” e o código da aplicação será clonado novamente do repositório no GitHub;

2. Continua no (6) do Cenário de Fluxo Principal.

2. (2-5) O usuário marca a opção “RESTART_TRAINING” e a aplicação irá resetar todos os scores das strings presentes no *kind* do datastore selecionado.

Continua no (6) do Cenário de Fluxo Principal.

4.3.5 Fluxo de Exceção

N/A

4.3.6 Pós-Condições

A aplicação é inicializada.

4.4 [UC002] - Validar script de teste original

4.4.1 Atores

Servidor de automação (Secundário)

4.4.2 Pré-Condições

Usuário está logado no sistema.

Usuário possui *script* de teste.

4.4.3 Fluxo Principal

1. O usuário inicializa a aplicação inserindo *script .py* válido;

2. A aplicação executa o *script .py* inserido pelo usuário;

3. A aplicação analisa o resultado da execução do *script .py* inserido pelo usuário;

4. O *Jenkins* exibe o resultado de sucesso da execução;

5. A aplicação inicializa a etapa de permutação do *script .py* inserido pelo usuário;

6. O caso de uso é encerrado.

4.4.4 Fluxo Alternativo

1. (1) O usuário inicializa a aplicação inserindo arquivo com formato inválido;

2. Continua no (2) do Cenário de Fluxo Principal.

3. (4) O *Jenkins* exibe o resultado de falha para a execução com arquivo inválido;

4. (5) A aplicação é encerrada com mensagem de erro na interface do *Jenkins*.

5. Continua no (6) do Cenário de Fluxo Principal.

4.4.5 Fluxo de Exceção

1. (4) O *Jenkins* exibe o resultado de falha para a execução com *script .py* válido;

2. (5) A aplicação é encerrada com mensagem de erro na interface do *Jenkins*.

3. Continua no (6) do Cenário de Fluxo Principal.

4.4.6 Pós-Condições

A aplicação inicia a etapa de permutação do *script* inserido.

4.7 [UC003] - Explorar o sistema através das variações do script

4.7.1 Atores

Servidor de automação (Secundário)

4.7.2 Pré-Condições

Usuário está logado no sistema.

Usuário insere *script* de teste válido.

Usuário inicializa a aplicação.

Sistema valida com sucesso o *script* inserido.

4.7.3 Fluxo Principal

1. A aplicação inicializa a etapa de permutação do *script .py* inserido pelo usuário;

2. A aplicação busca uma *String* no repositório do *Datastore*;

3. A aplicação realiza o tratamento da *String* buscada no repositório;

4. A aplicação insere a *String* buscada em algum campo de texto válido do *script .py*.

5. A aplicação executa o *script .py* alterado;

6. A aplicação analisa o resultado da execução e os *logs* do *website* alvo;

7. A aplicação determina o reforço a ser aplicado na *String* utilizada;

8. A aplicação atualiza o atributo *score* da *String* utilizada no repositório do *Datastore*;

9. A aplicação repete os passos acima de acordo com o atributo ITER escolhido pelo usuário;

10. O caso de uso é encerrado.

4.7.4 Fluxo Alternativo

N/A

4.7.5 Fluxo de Exceção

1. (3) A aplicação falha em realizar o tratamento da String buscada no repositório;

2. A aplicação registra a falha no tratamento e pula a execução da rodada atual;

3. Continua no (2) do Cenário de Fluxo Principal.

4.7.6 Pós-Condições

A aplicação finaliza as permutações e exibe o registro final da execução.

4.8 [UC004] - Registrar resultados da execução

4.8.1 Atores

Usuário (Principal) e Repositório (Secundário)

4.8.2 Pré-Condições

Usuário está logado no sistema.

Usuário insere *script* de teste válido.

Usuário inicializa a aplicação.

Sistema valida com sucesso o *script* inserido.

Sistema finaliza a execução de todas as permutações.

4.8.3 Fluxo Principal

1. A aplicação registra o resultado das rodadas no kind “execution-records” no repositório;

2. A aplicação encerra sua execução;

3. O *Jenkins* finaliza a execução do Job;

4. O Jenkins exibe o resultado da execução para o Usuário;

5. O caso de uso é encerrado.

4.8.4 Fluxo Alternativo

N/A

4.8.5 Fluxo de Exceção

1. (1) A aplicação falha em criar registros no repositório do *Datastore*;

2. A aplicação exibe a mensagem de erro no console do *Jenkins*;

3. Continua no (2) do Cenário de Fluxo Principal.

4.8.6 Pós-Condições

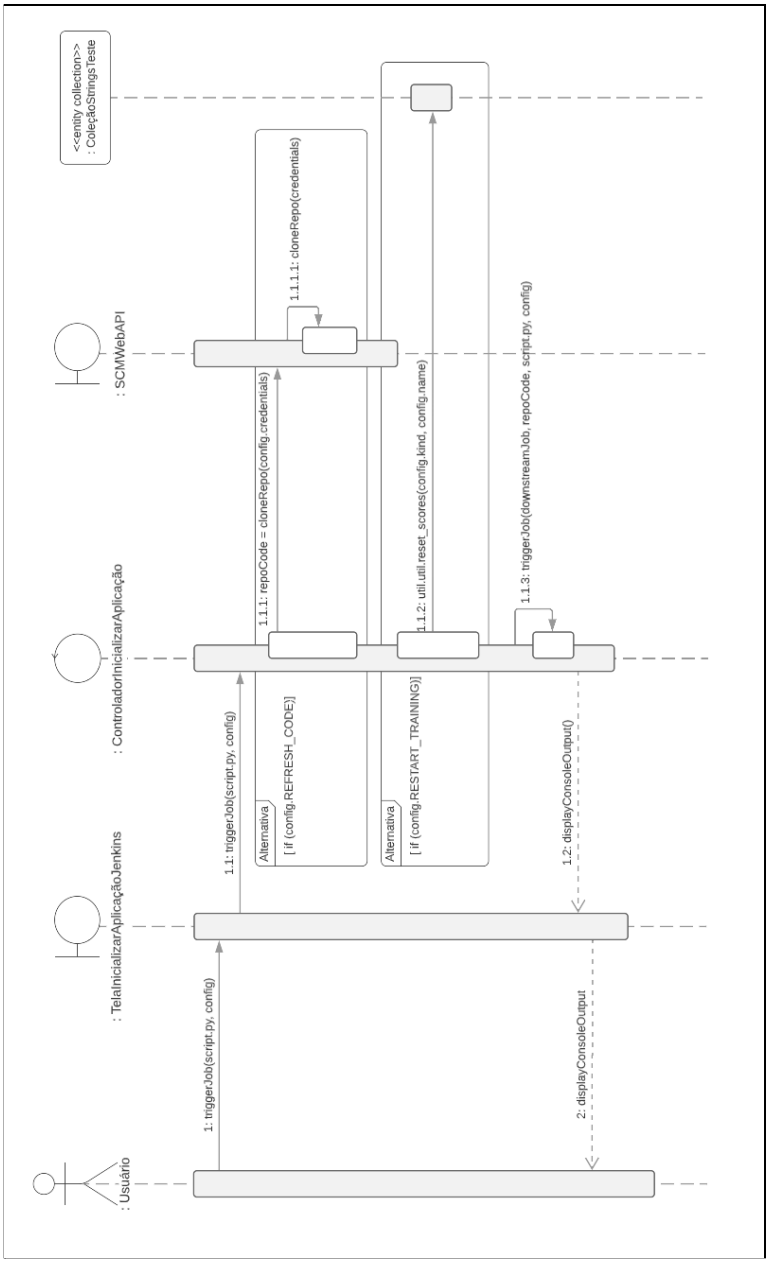
A execução da aplicação é encerrada.

5 Diagramas UML

5.1 Diagramas de Sequência

5.1.1 Inicializar a aplicação

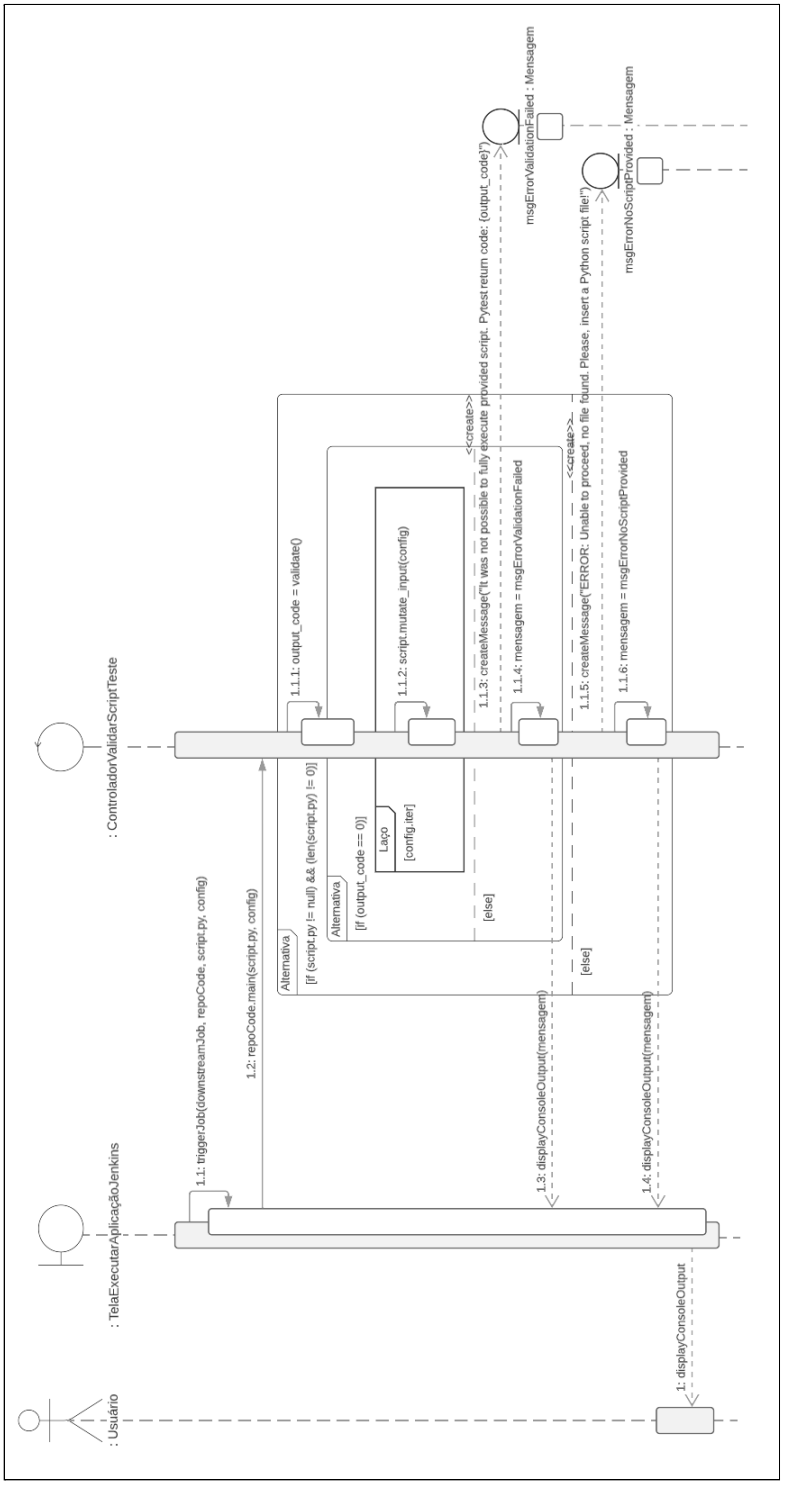
Figura 12 – Diagrama de sequência de análise do caso de uso [UC001] - Inicializar a aplicação



Fonte: O autor

5.1.2 Validar script de teste original

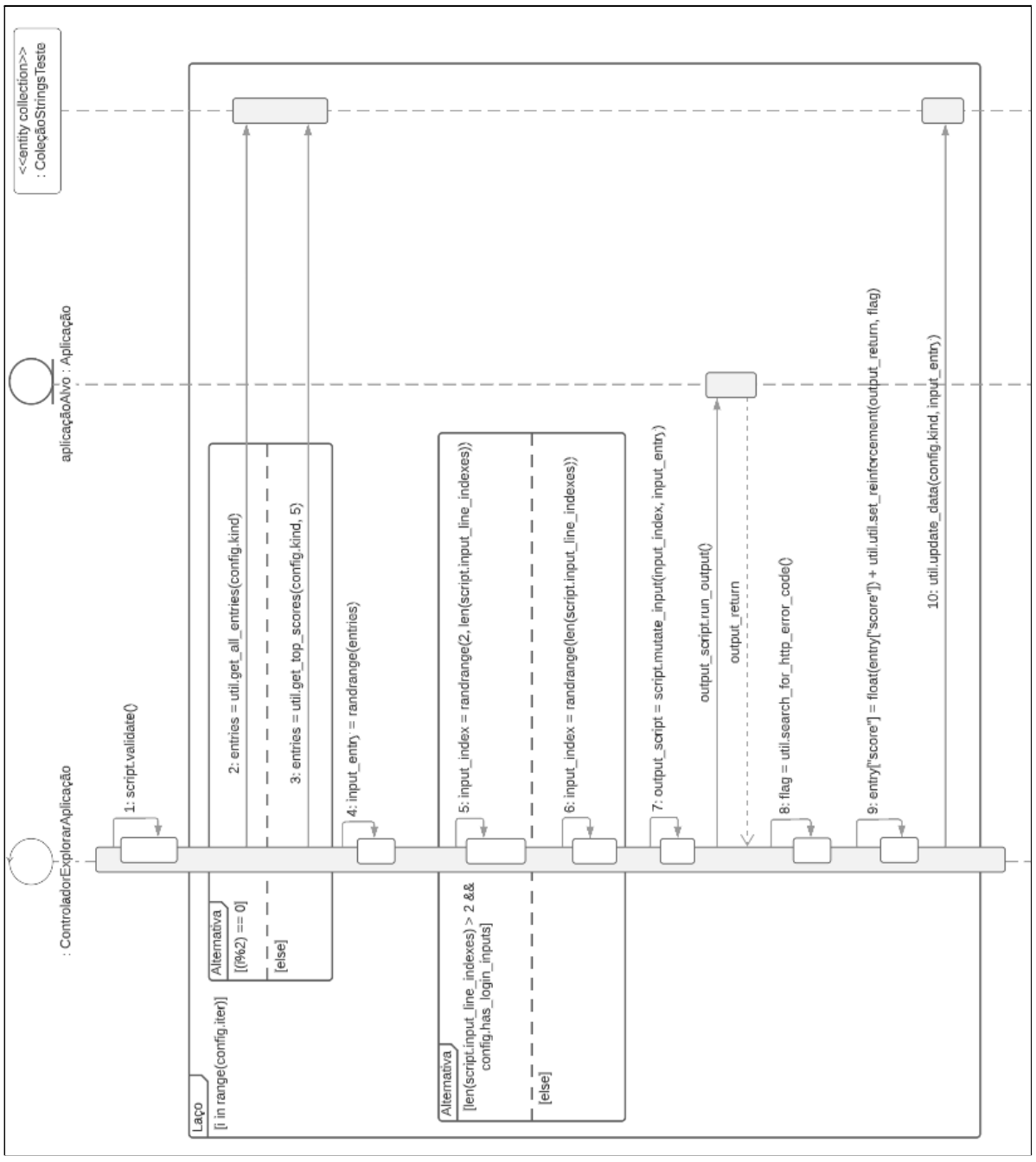
Figura 13 – Diagrama de sequência de análise do caso de uso [UC002] - Validar script de teste original



Fonte: O autor

5.1.3 Explorar o sistema através das variações do script

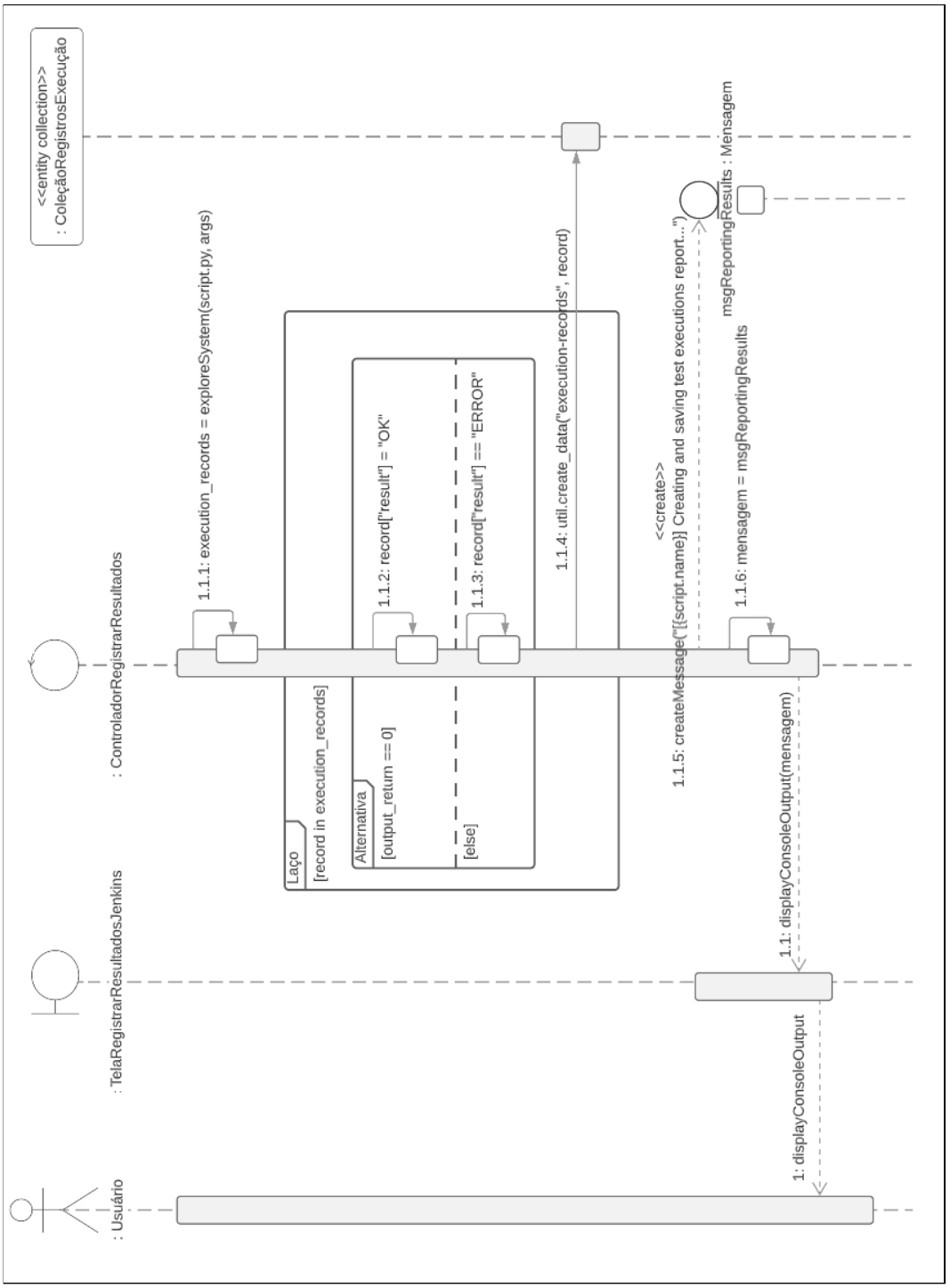
Figura 14 – Diagrama de sequência de análise do caso de uso [UC003] - Explorar o sistema através das variações do script



Fonte: O autor

5.1.4 Registrar resultados da execução

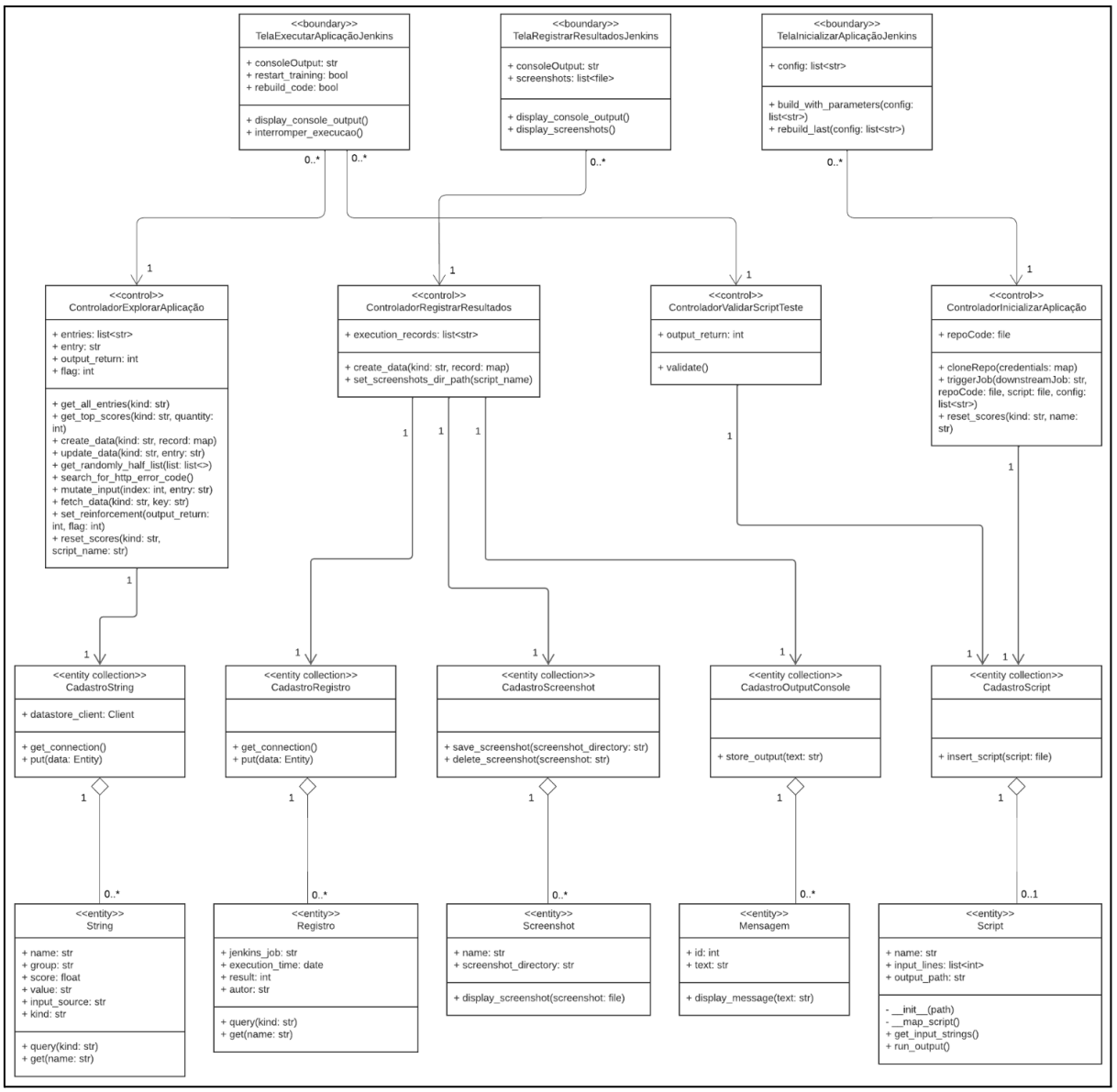
Figura 15 – Diagrama de sequência de análise do caso de uso [UC004] - Registrar resultados da execução



Fonte: O autor

5.2 Diagrama de Classes

Figura 16 – Diagrama de classes de análise



Fonte: O autor

6 Requisitos Funcionais

6.1 [RF001] Inicializar a aplicação

Com esta funcionalidade, é possível que o usuário forneça as informações necessárias e inicialize o job através da interface do *Jenkins*, que será responsável pela execução da aplicação.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

6.1.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

6.1.2 Fluxo de eventos principal:

1. Acessar o *Job Trigger* no *Jenkins*;
2. Clicar em “*Build With Parameters*”;
3. Inserir um arquivo .py;
4. Informar a quantidade de *rounds* (iterações) de execução do *script*;
5. Selecionar qual o repositório de *Strings* deverá ser utilizado na execução;
6. Clicar em “*Build*".

6.1.3 Saídas e pós condições:

O usuário deverá ser notificado de que o *job Trigger* está sendo executado. Ao fim de sua execução, o *job* dispara automaticamente o *downstream job* responsável por executar os testes com os *scripts* permutados e novamente notifica o usuário.

6.2 [RF002] Reiniciar treinamento

Com esta funcionalidade, é possível que o usuário opte por apagar todo o conhecimento previamente adquirido pela aplicação com relação aos *inputs* mais propensos a ocasionar falhas nos *scripts*. Resetando o atributo *score* de cada *String* presente no repositório, um novo treinamento será iniciado.

Prioridade: [ ] Essencial [ ] Importante [X] Desejável

6.2.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

6.2.2 Fluxo de eventos principal:

1. Acessar o *Job Trigger* no *Jenkins*;
2. Clicar em “*Build With Parameters*”;
3. Inserir um arquivo .py;
4. Informar a quantidade de *rounds* (iterações) de execução do *script*;
5. Selecionar qual o repositório de *Strings* deverá ser utilizado na execução;
6. Selecionar a opção “*Restart Training*”;
7. Clicar em “*Build*".

6.2.3 Saídas e pós condições:

O usuário deverá ser notificado de que o job Trigger está sendo executado. Ao fim de sua execução, o job dispara automaticamente o *downstream job* responsável por executar os testes com os *scripts* permutados, utilizando todo o escopo de *inputs* disponível no repositório do Datastore.

6.3 [RF003] Buildar código da aplicação

Com esta funcionalidade, é possível que o usuário utilize na execução o código mais recente da aplicação disponível no repositório do *GitHub*.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

6.3.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

6.3.2 Fluxo de eventos principal:

1. Acessar o *Job Trigger* no *Jenkins*;
2. Clicar em “*Build With Parameters*”;
3. Inserir um arquivo .py;
4. Informar a quantidade de *rounds* (iterações) de execução do *script*;
5. Selecionar qual o repositório de *Strings* deverá ser utilizado na execução;
6. Selecionar a opção “*Refresh Code*”;
7. Clicar em “*Build*".

6.3.3 Saídas e pós condições:

O usuário deverá ser notificado de que o código antigo armazenado na máquina foi removido e que o repositório foi clonado para que a aplicação seja executada utilizando o código mais recente disponível no repositório do *GitHub*.

6.4 [RF004] Validar script de teste original

Com esta funcionalidade, é possível que a aplicação valide o *script* de teste inserido pelo usuário para garantir que o código é funcional antes de realizar qualquer modificação no *script*. A validação consiste na análise do resultado da execução do *script* original.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

6.4.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

6.4.2 Fluxo de eventos principal:

1. Acessar o *Job Trigger* no *Jenkins*;
2. Clicar em “*Build With Parameters*”;
3. Inserir um arquivo .py;
4. Informar a quantidade de *rounds* (iterações) de execução do *script*;
5. Selecionar qual o repositório de *Strings* deverá ser utilizado na execução;
6. Clicar em “*Build*".

6.4.3 Saídas e pós condições:

O usuário deverá receber a informação de que o *script* inserido é válido e a etapa de permutação será iniciada, ou a aplicação informará que o *script* inserido é inválido e a execução da aplicação será encerrada.

6.5 [RF005] Explorar o sistema através das variações do script

Com esta funcionalidade, é possível que o usuário obtenha o principal objeto produzido pela aplicação desenvolvida neste trabalho, que consiste na exploração da interface do sistema alvo. Cada campo de entrada de dados na tela de uma aplicação é regido por regras específicas que devem ser validadas constantemente para garantir a consistência e integridade do software.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

6.5.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

2. O usuário deve inicializar a aplicação com todas as informações necessárias;

3. A aplicação deve validar o *script* original com sucesso;

6.5.2 Fluxo de eventos principal:

1. Aplicação executa todos os *rounds* definidos pelo usuário;

2. Aplicação realiza o reforço de cada *String* participante da exploração;

3. Aplicação registra toda a informação de cada execução no repositório;

4. Aplicação encerra sua execução.

6.5.3 Saídas e pós condições:

O usuário verá, ao final da exploração, o conjunto de informações relevantes que foram geradas durante cada rodada de execução, associado ao resultado obtido e registrado no repositório do *Datastore* acerca do treinamento da aplicação para futuras execuções.

6.6 [RF006] Permutar entradas no script

Com esta funcionalidade, é possível que a aplicação proporcione alto nível de manipulação do *script* original inserido pelo usuário, com o objetivo de interagir o máximo possível com o sistema alvo.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

6.6.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

2. O usuário deve inicializar a aplicação com todas as informações necessárias;

3. A aplicação deve validar o *script* original com sucesso;

6.6.2 Fluxo de eventos principal:

1. A aplicação deve validar o *script* original com sucesso;

2. A aplicação deve identificar as linhas do *script* que deverão ser modificadas;

3. A aplicação deve buscar *Strings* do repositório escolhido pelo usuário no *Google Cloud Datastore*;

4. Aplicação modifica uma linha por *round*, inserindo as *Strings* buscadas.

6.6.3 Saídas e pós condições:

A aplicação deverá, obrigatoriamente, a cada *round* de execução, gerar um novo *script .py* com um novo valor de *input* em algum dos métodos utilizados para inserir texto no sistema alvo, respeitando as regras de negócio previamente definidas. A cada *round*, o último *script* gerado é utilizado como base para a nova etapa de alteração.

6.7 [RF007] Reforçar o aprendizado da aplicação

Com esta funcionalidade, é possível que a aplicação analise o resultado da execução e aplique o reforço, positivo ou negativo, no atributo *score* de cada *String* participante da última rodada executada, de acordo com sua influência na identificação de erros no sistema alvo.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

6.7.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

2. O usuário deve inicializar a aplicação com todas as informações necessárias;

3. A aplicação deve validar o *script* original com sucesso;

4. A aplicação deve finalizar a rodada de execução atual para uma determinada *String*.

6.7.2 Fluxo de eventos principal:

1. A aplicação deve analisar o resultado da execução do *script* disponibilizado pelo *Pytest*;

2. A aplicação deve analisar os *logs* do *console* do navegador obtidos durante a execução do *script*;

3. A aplicação deve definir a pontuação numérica a ser acrescida ou decrescida do valor atual do atributo *score* da *String* utilizada na execução;

4. A aplicação deve realizar a alteração do valor no *Datastore*.

6.7.3 Saídas e pós condições:

O usuário verá informações nos *logs* sobre a atualização do valor do *score* da *String* utilizada no *Jenkins*, assim como poderá encontrar o novo valor atualizado no repositório do *Datastore*.

6.8 [RF008] Registrar resultados da execução

Com esta funcionalidade, é possível que o usuário tenha, de forma fácil e acessível, uma alta rastreabilidade dos testes executados pela aplicação e das informações mais relevantes coletadas durante cada execução.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

6.8.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

2. O usuário deve inicializar a aplicação com todas as informações necessárias;

3. A aplicação deve validar o *script* original com sucesso;

4. A aplicação deve finalizar a execução de todas as rodadas definidas inicialmente pelo usuário.

6.8.2 Fluxo de eventos principal:

1. A aplicação deve persistir toda a informação relevante no *kind execution-records* do *Datastore*.

6.8.3 Saídas e pós condições:

O usuário poderá acessar e visualizar um resumo das informações importantes que foram obtidas durante a execução das rodadas no repositório do *Datastore*.

6.9 [RF009] Exibir resultados da execução

Com esta funcionalidade, é possível que o usuário tenha, em tempo real no console do *Jenkins*, o rastreamento de informações de *logs* dos testes executados pela aplicação durante cada rodada de execução.

Prioridade: [ ] Essencial [ ] Importante [X] Desejável

6.9.1 Entradas e pré-condições:

1. O usuário deverá estar devidamente logado no *Jenkins*;

2. O usuário deve inicializar a aplicação com todas as informações necessárias;

3. A aplicação deve validar o *script* original com sucesso;

4. A aplicação deve finalizar a execução de todas as rodadas definidas inicialmente pelo usuário.

6.9.2 Fluxo de eventos principal:

1. A aplicação deve disponibilizar um resumo das informações que foram coletadas durante a execução;

6.9.3 Saídas e pós condições:

O usuário verá um resumo das informações importantes que foram obtidas durante a execução das rodadas no *console* do *Jenkins*.

7 Requisitos Não Funcionais

7.1 Usabilidade

7.1.1 [RNF001] Acessibilidade

Com esta funcionalidade, é possível que o usuário navegue e alcance seus objetivos através das interfaces utilizadas na aplicação de forma simples e intuitiva.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

7.1.2 [RNF002] Responsividade

Com esta funcionalidade, é possível que o usuário utilize as funcionalidades da aplicação em diferentes aparelhos com diferentes configurações de *software* e *hardware*.

Prioridade: [ ] Essencial [ ] Importante [X] Desejável

7.2 Desempenho

7.2.1 [RNF003] Eficiência

Com esta funcionalidade, é possível que o usuário possa utilizar a aplicação sem enfrentar travamentos ou falhas constantes. Além disso, as funcionalidades devem ser executadas no menor tempo possível.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

7.2.2 [RNF004] Uso de recurso

Com esta funcionalidade, é possível que o usuário utilize os recursos de seu *hardware* para outras tarefas simultaneamente à execução da aplicação, levando em

consideração o baixo uso de recursos dada a otimização de funcionamento do sistema desenvolvido.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

7.3 Confiabilidade

7.3.1 [RNF005] Disponibilidade

Com esta funcionalidade, é possível que o usuário tenha suas necessidades atendidas pela aplicação todos os dias do ano, 24 horas por dia.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

7.4 Segurança

7.4.1 [RNF006] Integridade

Com esta funcionalidade, é possível que o usuário possua precisão e consistência nas informações disponibilizadas pela aplicação, de forma que não haja interferência externa sobre os dados a fim de corrompê-los de alguma forma.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

7.4.2 [RNF007] Confidencialidade

Com esta funcionalidade, apenas é possível que o usuário utilize a interface do *Jenkins* para realizar a operação de execução dos testes através de autenticação com senha. Além disso, informações de *login* e senhas utilizadas no *script* de teste não são armazenadas ou disponibilizadas no repositório do *Datastore*.

Prioridade: [X] Essencial [ ] Importante [ ] Desejável

7.5 Manutenibilidade

7.5.1 [RNF008] Reparo

Com esta funcionalidade, em caso de problemas com os serviços de integração, é possível que o usuário utilize a recuperação de desastre e restaure o sistema em uma versão estável através do *backup* automático semanal no *Jenkins*, gerenciado pelo plugin *ThinBackup*.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

7.5.2 [RNF009] Evolução

Com esta funcionalidade, é possível que a comunidade interessada nos benefícios promovidos pela aplicação adicione e incremente suas funcionalidades.

Prioridade: [ ] Essencial [ ] Importante [X] Desejável

7.6 Tecnologias envolvidas

7.6.1 [RNF0010] Integrações

Com esta funcionalidade, é possível que o usuário seja beneficiado pela qualidade e estabilidade das aplicações integradas ao sistema desenvolvido, como o *Jenkins*, o *GitHub* e o *Datastore*.

Prioridade: [ ] Essencial [X] Importante [ ] Desejável

8 Conclusão e Trabalhos Futuros

8.1 Conclusão

Sabe-se que os testes de *software* devem ser encarados como um processo que exige um planejamento detalhado. E dentre as diversas abordagens disponíveis atualmente para sua realização, a utilização da técnica de testes automatizados possui um retorno mais alto se comparado às alternativas de testes, isso se dá devido a sua praticidade e alto desempenho. Portanto, os esforços para a implantação e desenvolvimento de novas técnicas que permitam saltos e avanços para a automação de testes que explorem cada vez mais o ambiente onde são executados se mostram válidos, pois pagam o investimento.

A aplicação desenvolvida, apesar de possuir diversas possíveis melhorias a serem implementadas, se mostrou funcional e capaz de executar testes que não estão contidos em *scripts* estáticos. A utilização de *scripts* que são dinamicamente alterados com o passar do tempo de forma a exercer o aprendizado sobre quais alterações realizar em seu favor e explorar o SUT demonstra o potencial que a solução automatizada possui. Deve-se levar em consideração que o trabalho desenvolvido utilizou a ferramenta JIRA como alvo das execuções de teste apenas para demonstrar o funcionamento da aplicação em situações em que formulários de entrada de dados estão rotineiramente presentes, não existindo o objetivo de obter uma alta taxa de aprendizado ou de erros encontrados na ferramenta.

De forma geral, o resultado obtido com a implementação da POC foi satisfatório e demonstrou ser capaz realizar a execução de ET de forma automatizada com a menor intervenção humana possível, com base em reforços obtidos em experiências passadas. A aplicação desenvolvida tem o potencial de auxiliar todos os profissionais de QA envolvidos no processo de testes, como os testadores que executam *scripts* de teste estáticos de regressão, os próprios testadores automatizadores que desenvolvem os *scripts* de automação, além de desenvolvedores que podem se beneficiar ao executar um ET em seu objeto de

trabalho recém desenvolvido, a fim de obter um *feedback* automático e confiável sobre o nível de qualidade e consistência do SUT.

8.2 Trabalhos Futuros

Apesar dos resultados satisfatórios obtidos durante a execução de testes utilizando a aplicação desenvolvida no presente trabalho, melhorias podem ser realizadas a fim de solidificar a utilização da ferramenta na área de testes, objetivando sempre melhorar o processo de testes e auxiliar o responsável pela atividade de execução dos testes de *software*.

Uma generalização no tratamento dos *scripts* de teste inseridos na ferramenta pode ser implementada para abranger testes que interagem com sistemas com diferentes interfaces de *login* ou telas de apresentação, resultando em um aumento de desempenho na mutação dos *scripts* e consequentemente uma melhoria no processo de exploração.

A aplicação deve ser disponibilizada para uso de forma *online*. Um estudo pode ser realizado para identificar os ajustes necessários na interface de uso da aplicação no *Jenkins* e no repositório da massa de dados no *Datastore* para tornar a aplicação acessível aos potenciais usuários. Uma outra forma de alcançar esse objetivo futuro seria a adaptação da ferramenta desenvolvida para um *plugin* do Jenkins. O servidor de automação é conhecido pela sua enorme flexibilidade e alta disponibilidade de *plugins* que complementam as funcionalidades nativas do sistema, inserir a aplicação como uma extensão do software pode incluir a execução de ET nas diversas fases de Integração Contínua, do inglês *Continuous Integration* (CI) e da Entrega Contínua, do inglês *Continuous Delivery* (CD).

O surgimento de novas ferramentas e atualizações e aprimoramentos das opções já existentes no mercado acontecem frequentemente. A versão 4 do *framework Selenium*, disponibilizada em outubro de 2021, por exemplo, inseriu a opção de *Relative Locators* nas suas funcionalidades, que auxilia na exploração da página *web* através da possibilidade de interação com elementos próximos aos já

mapeados. Um novo módulo pode ser desenvolvido e integrado ao já existente para utilizar o novo modelo de mapeamento a favor da execução dos ET, agregando a interação com os demais elementos da página *web*, além dos campos de entrada de dados.

Além da adição de módulos que incrementem a exploração da interface, a técnica de *workflow based testing* pode ser utilizada para que a navegação no *website* alvo siga instruções e sequências lógicas durante a execução, como foi experimentado e documentado no estudo do uso de RL em interfaces *web* usando a exploração guiada por *workflow*. (LIU et al., 2018).

Existe também espaço para a aplicação de algoritmos de reforço mais bem elaborados, tendo em vista que a recompensa e punição implementadas no presente trabalho é uma simples atribuição estática de valores baseadas no resultado da execução do *script* e nos erros HTTP presentes no *console log* do navegador, utilizada apenas para provar o conceito do reforço no aprendizado durante a exploração. Um estudo associado a experimentações práticas pode revelar técnicas que alcancem um melhor desempenho e maior generalização de utilização da ferramenta desenvolvida.

Outra funcionalidade que pode ser evoluída na solução desenvolvida neste trabalho é tornar o permutador de *scripts* compatível com *scripts* em diferentes linguagens de programação, tendo em vista que atualmente apenas o *Python* é utilizado, e automaticamente gerados por ferramentas de gravação e captura de navegação. Tais ferramentas estão presentes em extensões como o Selenium IDE e o UI Vision e são compatíveis com diversos navegadores de internet, como o Google Chrome e o Firefox. Os *scripts* podem ser gerados em diferentes linguagens de programação, e a aplicação pode ser adaptada para utilizá-los sem que o usuário precise realizar nenhuma intervenção no formato ou linguagem do *script* gerado.

Referências

ABDO, Ahmed; ALALI, Sardar. Graphical User Interface Features In Building Attractive And Successful Websites. **European Journal of Scientific Research**. maio 2016.

ANDERSON, Janna; RAINIE, Lee. Artificial Intelligence and the future of Humans. **Pew Research Center**, p. 86-87, 10 dez. 2018.

BACH, James. **Exploratory Testing Explained.** v.1.3. c2003.

BARBOSA, Simone. Interação Humano-Computador. **Elsevier**. ed. 1. ago. 2010.

BARNES, Susan. User Friendly: A Short History of the Graphical User Interface. **Sacred Heart University Review**. v. 16. a. 4. fev. 2010.

BIRT, James. Software Reliability Enhancement Through Error-Prone Path Identification using Genetic Algorithms. **School of Information and Communication Technology**. 8 maio 2006.

CHANDRAPRABHA; KUMAR, Ajeet; SAXENA, Sajal. Data Driven Testing Framework using Selenium WebDriver. **International Journal of Computer Applications (0975 – 8887)**. v. 118. n. 18. maio 2015.

DAVIS, Phillips; BARNETT, Amanda. Mars Climate Orbiter. **NASA Science Solar System Exploration**. maio. 2016. Disponível em: <https://solarsystem.nasa.gov/missions/mars-climate-orbiter/in-depth/> Acesso em: 02 de março de 2022.

DIAS, Kapila. Evolvement of Computer Aided Software Engineering (CASE) Tools: A User Experience. International Journal of Computer Science and Software Engineering (IJCSSE). v. 6, p.55-60. mar. 2017.

DURELLI, Vinicius; DURELLI, Rafael; BORGES, Simone; ENDO, Andre; ELER, Marcelo; DIAS, Diego; GUIMARAES, Marcelo. Machine Learning Applied to Software Testing: A Systematic Mapping Study. **IEEE Transactions on Reliability**. v.68. 11 fev. 2019.

ECKART, Li; ECKART, Sven; ENKE, Margit. A brief comparative study of the potentialities and limitations of machine-learning algorithms and statistical techniques. **E3S Web of Conferences**. v. 266. jun. 2021.

FIGUEIREDO, Jonathas; REJAILI, Rodrigo. Aplicação de algoritmos de aprendizagem por reforço para controle de navios em águas restritas. **EPUSP**. 2018.

FITZPATRICK, Geraldine. A Short History of Human Computer Interaction: A People-Centred Perspective. **SIGUCCS'18**. out. 2018.

FREITAS, Taina. **Os três tipos de aprendizado no machine learning, um ramo da inteligência artificial**. out. 2019. Disponível em: <https://www.startse.com/noticia/nova-economia/machine-learning-inteligencia-artificial-aprendizado> Acesso em: 01 de abr. de 2022.

GitHub. Página inicial. abr. 2008. Disponível em: <https://github.com/> Acesso em: 02 de março de 2022.

Google Cloud Datastore. Página inicial. maio. 2013. Disponível em: <https://cloud.google.com/datastore> Acesso em: 05 de fevereiro de 2022.

GWILT, Ian. DIGITAL ART: A brief history of the Graphical User Interface in contemporary art practice, 1994 - 2004. **Proceedings of the Ninth International Conference on Information Visualisation (IV'05)**. jul. 2005.

HENDRICKSON, Elizabeth. Explore It! Reduce Risk and Increase Confidence. **Pragmatic Bookshelf**. mar. 2013.

ITKONEN, Juha; MANTYLA, Mika; LASSENIUS, Casper. The Role of the Tester's Knowledge in Exploratory Software Testing. **IEEE Transactions on Software Engineering**. v. 39. p. 707-724. maio. 2013.

IVANOVA, Kateryna; KONDRATENKO, Galyna; SIDENKO, Ievgem; KONDRATENKO, Yuriy. Artificial Intelligence in Automated System for Web-Interfaces Visual Testing. **CEUR workshop proceedings**. v. 2604. p. 1019-1031. 2020.

Jenkins. Página inicial. fev. 2011. Disponível em: <https://www.jenkins.io/> Acesso em: 02 de março de 2022.

JHA, Nisha; GULATI, Rishu. Keyword Driven Testing Framework using Selenium Webdriver. **International Journal of Advanced Research in Engineering and Technology (IJARET)**. v. 9. p. 180–186. ago. 2018.

JIRA Software Cloud. c2022. Página inicial. Disponível em: <https://www.atlassian.com/software/jira> Acesso em: 02 fevereiro de 2022.

KANER, Cem. A Tutorial in Exploratory Testing. **QAI QUEST Conference**. abr. 2008.

KANER, Cem; BACH, James; PETTICHORD, Bret. Lessons Learned in Software Testing. **John Wiley & Sons, Inc**. 1 ed. dez. 2001.

KANER, Cem; FALK, Jack; NGUYEN, Hung Q. Testing Computer Software. **Tab Books**. 1 ed. jan. 1988.

KERAMIDAS, Kimon; BANAS, Emily; DAMATO, Martina; DICHTER, Caitlin; GARDNER, Andrew; JIWA, Alana; KILLMAR, Jane; KOK, CynthiA. **The Interface Experience: 40 Years of Personal Computing**. 2015. Disponível em:

<https://interface-experience.org/objects/xerox-star-8010-information-system/> Acesso em: 03 de abril de 2022.

LACERDA, Denis. Aprendizado por reforço em lote: um estudo de caso para o problema de tomada de decisão em processos de venda. **Instituto de Matemática e Estatística, USP**. ago. 2013.

LANGSTON, Jennifer. **Com o aprendizado por reforço, a Microsoft traz uma nova classe de soluções de IA para os clientes**. dez. 2020. Disponível em: <https://news.microsoft.com/pt-br/com-o-aprendizado-por-reforco-a-microsoft-traz-uma-nova-classe-de-solucoes-de-ia-para-os-clientes/> Acesso em: 05 de abril de 2022.

LIU, Evan; GUUZ, Kelvin; PASUPATY, Panupong; SHIY, Tianlin; LIANG, Percy. Reinforcement Learning on Web Interfaces Using Workflow-Guided Exploration. **International Conference on Learning Representations (ICLR)**. fev. 2018.

MALIK, Momin. A Hierarchy of Limitations in Machine Learning. **arXiv.org**. fev. 2020.

MCINERNEY, Dan. **Fuzz String Lists**. Disponível em: <https://github.com/DanMcInerney/FuzzStrings> Acesso em: 02 de março de 2022.

MEDEIROS, Amanda. A evolução do aprendizado de máquina e o impacto disso no mercado. **Consumidor Moderno**, 17 Mar. 2021. Disponível em: <https://www.consumidormoderno.com.br/2021/03/17/evolucao-aprendizado-de-maquina-impacto-mercado/>. Acesso em: 2 mar. 2022.

MEDEIROS, Ernani. Desenvolvendo software com UML 2.0 - Definitivo. **Pearson Makron Books**. 1 ed. jan. 2004.

MIESSLER, Daniel. **SecLists**. Disponível em: <https://github.com/danielmiessler/SecLists/tree/master/Fuzzing> Acesso em: 02 de março de 2022.

MITCHELL, Tom. Machine Learning. **McGraw-Hill Science/Engineering/Math**. 1. ed. 1 mar. 1997.

MYERS, Brad. A Brief History of Human Computer Interaction Technology. **interactions**. mar. 1998.

NEVES, Enzo. **Aprendizado por Reforço #1— Introdução**. fev. 2020. Disponível em: <https://medium.com/turing-talks/aprendizado-por-refor%C3%A7o-1-introdu%C3%A7%C3%A3o-7382ebb641ab> Acesso em: 01 de maio de 2022.

OLSEN, Klaus; POSTHUMA, Meile; ULRICH, Stephanie. Certified Tester Foundation Level Syllabus - CTFL 2018br. **ISTQB International Software Testing Qualifications Board**. v. 3.1. nov. 2019.

PAOLANTI, Marina; ROMEO, Luca; FELICETTI, Andrea; MANCINI, Adriano; FRONTONI, Emanuele; LONCARSKI, Jelena. Machine Learning approach for Predictive Maintenance in Industry 4.0. **2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)**. jul. 2018.

PASCHEK, Daniel; LUMINOSU, Caius; DRAGHICI, Anca. Automated business process management – in times of digital transformation using machine learning or artificial intelligence. **MATEC Web Conf**. v.121. 9 ago. 2017.

Pytest. c2015. Página inicial. Disponível em: <https://docs.pytest.org/en/7.0.x/> Acesso em: 05 de fevereiro de 2022.

Python. c2022. Página inicial. Disponível em: <https://www.python.org/> Acesso em: 03 de março de 2022.

RAI, Rahul; TIWARI, Manoj; IVANOV, Dmitry; DOLGUI, Alexandre. Machine learning in manufacturing and industry 4.0 applications. **International Journal of Production Research**. v. 59. p. 4773-4778. ago. 2021.

Ranorex GmbH. c2022. Selenium WebDriver. Disponível em: <https://www.ranorex.com/resources/testing-wiki/selenium-testing/> Acesso em: 31 de maio de 2022.

SAMUEL, Arthur L. Some Studies in Machine Learning Using the Game of Checkers. **IBM Journal of Research and Development**. v. 3, n. 3, p.210–229. jul. 1959.

Selenium IDE. c2022. Página inicial. Disponível em: <https://www.selenium.dev/selenium-ide/> Acesso em: 31 de maio de 2022.

Selenium WebDriver. c2022. Página inicial. Disponível em: <https://www.selenium.dev/documentation/webdriver/> Acesso em: 31 de maio de 2022.

SUTTON, Richard.; BARTO, Andrew. Reinforcement Learning, second edition: An Introduction. **Bradford Book**. 2. ed. p.1. 13 nov. 2018.

TIDWELL, Jenifer. Designing Interfaces Second Edition. **O'Reilly Media, Inc**. dez. 2010.

UI Vision. c2022. Página inicial. Disponível em: <https://ui.vision/> Acesso em: 31 de maio de 2022.

WALTRICK, Camila. **Machine Learning — O que é, tipos de aprendizagem de máquina, algoritmos e aplicações**. maio. 2020. Disponível em:

<https://medium.com/camilawaltrick/introducao-machine-learning-o-que-e-tipos-de-aprendizado-de-maquina-445dcfb708f0> Acesso em: 01 de abril de 2022.

WOLF, Max. **Big List of Naughty Strings**. Disponível em: <https://github.com/minimaxir/big-list-of-naughty-strings> Acesso em: 02 de março de 2022.

ZHANG, Du. TSAI, Jeffery. Advances in Machine Learning Applications in Software Engineering. **IGI Global**. 1. ed. out. 2006.