



UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Extensão da Ferramenta Sikuli para Apoiar Testes Automatizados para Aplicações Windows Phone Sensíveis ao Contexto

Elizângela Santos da Costa

*Propto em que o TCE está inserido.
↳ Poderia estar escrito no introdução
de monografia.*

Manaus – AM
Fevereiro 2015

Elizângela Santos da Costa

Extensão da Ferramenta Sikuli para Apoiar Testes Automatizados para Aplicações Windows Phone Sensíveis ao Contexto

Monografia de Graduação apresentada ao Instituto de
Computação da Universidade Federal do Amazonas
como requisito parcial para obtenção do grau de
bacharel em Ciência da Computação.

Orientador

D.Sc. Arilo Claudio Dias Neto

Co-orientador

Rodrigo dos Anjos Cruz Reis

Universidade Federal do Amazonas – UFAM

Instituto de Computação – IComp

Manaus-AM

Fevereiro 2015

Extensão da Ferramenta Sikuli para Apoiar Testes Automatizados para Aplicações Windows Phone Sensíveis ao Contexto

Autor: Elizângela Santos da Costa

Orientador: D. Sc. Arilo Claudio Dias Neto

Co-orientador: Rodrigo dos Anjos Cruz Reis

RESUMO

A ampla difusão no uso dos dispositivos móveis trouxe a necessidade de aprimorar o processo de verificação e validação nos aplicativos móveis. A forma mais comum de interação existente entre esses aplicativos e os usuários é através da interface do dispositivo. Para aplicações sensíveis ao contexto, o número de interações que um usuário pode realizar é muito maior se comparado a aplicações comuns. Logo, a execução de teste manual se torna uma atividade exaustiva e passível de erros. Para amenizar esse problema e cobrir o máximo de falhas possíveis a serem encontradas pelos usuários, os testes automatizados têm sido cada vez mais praticados por reduzirem custos e esforços manuais contribuindo assim com a qualidade do produto final. Como principal contribuição, sugere-se então no presente trabalho a criação de novas funções pela ferramenta Sikuli que automatizem testes para aplicações móveis sensíveis ao contexto desenvolvidas na plataforma Windows Phone com a finalidade de desenvolver aplicativos mais confiáveis por meio de estratégia de teste eficaz.

Palavras-chave: Sensibilidade ao contexto, Testes automatizados, Sikuli, Teste Móvel.

Extension of Sikuli Tool to Support Automated Tests to Windows Phone Context-Aware Applications

Author: Elizângela Santos da Costa

Advisor: D. Sc. Arilo Claudio Dias Neto

Co-Advisor: Rodrigo dos Anjos Cruz Reis

ABSTRACT

The wide diffusion in the use of mobile devices has brought the need to improve the process of verification and validation on mobile applications. The usual way of interaction between this apps and users is by device interface. For mobile context-aware applications, the number of interactions that a user can perform it is much larger compared with common applications. Then, execution of manual testing becomes an exhaustive and error-prone activity. In order to deal this problem and to cover the maximum possible failure to be found by users, automated tests have been more and more practiced by reducing costs and manual efforts, thus, contributing to the quality of the final product. As main contribution, this work proposes the creation of new functions by Sikuli tool to automate tests for mobile context-aware application developed to Windows Phone platform in order to develop more reliable applications through effective test strategy.

Keywords: Context-awareness, Automated testing, Sikuli, Mobile Testing.

Lista de Figuras

Figura 1 – Funcionamento do Sikuli	17
Figura 2 - Interface Sikuli IDE.....	18
Figura 3 – Configurações de padrão de uma imagem	19
Figura 4 - Propriedade <i>Matching Preview</i>	19
Figura 5 – Propriedade Target Offset.....	20
Figura 6 – Exemplo de script que automatiza a ação de abrir a tela de configurações	21
Figura 7- Mensagem ao finalizar execução do script.....	21
Figura 8 - Estrutura de uma função	22
Figura 9 - Importando scripts	23
Figura 10 - Importando scripts desenvolvidos no trabalho	27
Figura 11 - Permissão de projeção da tela do celular	27
Figura 12 – Tela do celular sendo projetada pelo <i>Project My Screen App</i>	28
Figura 13 – Simulação de mudança de rotação.....	29
Figura 14 - Sensibilidade do aplicativo Google a mudanças de contexto	30
Figura 15 – Fluxo de automatização para o teste de orientação e conexão no aplicativo Google	31
Figura 16 - Mensagem de saída do Sikuli para o teste do aplicativo Google	31
Figura 17 - Sensibilidade do aplicativo Waze a mudanças de contexto.....	32
Figura 18 - Mensagem de saída do Sikuli para o teste do aplicativo Waze	33
Figura 19 – Sensibilidade do aplicativo Nível de Bateria a mudanças de contextos	34
Figura 20 - Entrada do usuário para o teste de bateria	34
Figura 21 - Mensagem de saída do Sikuli para o teste do aplicativo Nível de Bateria	35
Figura 22. Função <code>OpenApp(letra,app)</code>	41
Figura 23. Função <code>GoToSettings()</code>	41
Figura 24. Função <code>Home()</code>	42

Figura 25. Função OpenNotification()	42
Figura 26. Função DisconnectWifi()	43
Figura 27. Função Disconnect()	43
Figura 28. Função Connect()	43
Figura 29. Função Disconnect3G()	44
Figura 30. Função VerifyRotation()	44
Figura 31. Função Connect3G()	45
Figura 32. Função Connect3G() landscape	46
Figura 33. Função ChangeToPortrait()	45
Figura 34. Função ChangeToLandscape()	47
Figura 35. Função LocationOn()	47
Figura 36. Função LocationOn() landscape	48
Figura 37. Função LocationOff()	49
Figura 38. Função LocationOff() landscape	49
Figura 39. Função VerifyBattery(min,max)	50
Figura 40. Função VerifyBattery(min,max) landscape	50
Figura 41. Função AssertEqual(img)	48
Figura 42. Função flick_up()	51
Figura 43. Função flick_left()	51
Figura 44. Função flick_right()	52
Figura 45. Função GoogleTest()	53
Figura 46. Função GoogleTest() continuação	54
Figura 47. Função WazeTest()	54
Figura 48. Função WazeTest() continuação	55
Figura 49. Função NivelBateriaApp() - cores	55
Figura 50. Função NivelBateriaApp()	56
Figura 51. Função NivelBateriaApp() - continuação	56

Lista de tabelas

Tabela 1. Descrição das funções contidas no script <i>UtilWp</i>	25
Tabela 2 - Descrição das funções contidas no script scroll	26
Tabela 3 - Descrição das funções contidas no script PC	26

Índice

1.	Introdução	7
1.1	Motivação e Problema.....	8
1.2	Objetivo.....	9
1.3	Organização do Documento	9
2	Referencial Teórico	10
2.1	Computação Ubíqua e Sensibilidade ao Contexto	10
2.2	Teste para Aplicações Móveis.....	12
2.3	Teste GUI (Teste Baseado em Interface Gráfica)	14
2.3.1	Definições de Teste GUI	15
2.3.2	Técnicas de Teste GUI	15
2.4	Sikuli	16
2.4.1	Como funciona	17
2.4.2	Interface Sikuli IDE	18
2.4.3	Configurações de padrão	18
2.4.4	Exemplo de Script	20
2.4.5	Mensagens do Sikuli	21
2.4.6	Criando Novas Funções e Bibliotecas em Sikuli	22
3	Estendendo a Ferramenta Sikuli para Testes em Aplicações Móveis Sensíveis ao Contexto	24
3.1	Novas Funções para Testes em Aplicações Móveis Sensíveis ao Contexto	24
3.2	Prova de Conceito	29
3.2.1	Avaliação da Função de Orientação da Tela e Conexão com Internet.....	29
3.2.2	Avaliação da Função de Localização	32
3.2.3	Avaliação da Função de Bateria.....	33
3.3	Considerações sobre o uso das Funções Implementadas	35
4	Considerações Finais.....	37
5	Referências	38
	APÊNDICE A - Script UtilWP	41
	APÊNDICE B – Script scroll	51
	APÊNDICE C – Script PC	53

1. Introdução

Devido ao grande sucesso de dispositivos móveis, a engenharia de software busca aprimorar a qualidade nas aplicações desenvolvidas para esta plataforma (AMALFITANO et al., 2013). Na plataforma móvel, a principal forma de interação entre usuários e aplicações é através de sua interface gráfica (em inglês, GUI – *Graphical User Interface*). Assim, como seu desenvolvimento está sendo bastante difundido, surgem GUI cada vez mais complexas e que requerem uma maior preocupação com a sua qualidade.

Uma das formas de avaliação da qualidade de aplicações por meio de sua GUI é por meio da realização de uma técnica de teste chamada Teste GUI (RUIZ; PRICE, 2008). Tal tipo de teste deve simular a sequência de eventos efetuada pelos usuários. O grande número de possibilidades de entrada para essa sequência torna o teste GUI uma atividade complexa e requer um grande esforço manual para o processo de testes. Neste contexto, o planejamento do teste é um passo fundamental, o que inclui levar em consideração o número de casos de testes, o tamanho de cada caso de teste, definição de objetivos, definição de agenda de criação e execução dos testes, a criação dos testes, dentre outros aspectos a serem considerados (XIE; MEMON, 2006).

Outro fator importante que deve ser levado em consideração durante o processo de avaliação de aplicações móveis atualmente é a característica de **sensibilidade ao contexto** (AMALFITANO et al., 2013). Esta característica visa descrever os diferentes contextos em que as aplicações estão sujeitas, o que faz com que elas possam reagir de forma diferenciada a mudanças em seus contextos. Nesta categoria de aplicações, espera-se que as aplicações móveis recebam entradas que representem diferentes contextos que o dispositivo móvel pode assumir, o que pode levar a imprevisibilidade e alta variabilidade das entradas que a aplicação potencialmente pode receber. Diferentes contextos em diferentes aplicações possuem uma grande possibilidade de gerar falhas, e o critério de cobertura para atingir todas as possibilidades deve ser proposto. Uma alternativa para reduzir o esforço nesses testes tem sido a adoção de testes automatizados.

Segundo Abdulrahman (2014), mais de 70% dos custos dos testes podem ser reduzidos por meio da utilização de um processo de automação de testes. A automação de testes está certamente entre os meios mais importantes para alcançar a redução de esforço e custo nas atividades de teste em um projeto de software. A percepção comum

nas aplicações móveis é que elas devem ser baratas e custar menos do que as aplicações tradicionais, porém elas devem ser corretas, confiáveis e ter bom desempenho (ABDULRAHMAN, 2014). Assim, automação de testes se apresenta como uma ótima alternativa para atingir este objetivo.

Dado esse cenário, o presente trabalho pretende fornecer a criação de novas funções, por meio da geração de scripts elaborados na ferramenta Sikuli que ampliem a ação de automatização de testes em aplicações móveis sensíveis ao contexto, especificamente da plataforma Windows Phone, incluindo a automatização de diferentes contextos que o dispositivo pode assumir os quais causam influência no comportamento da aplicação.

1.1 Motivação e Problema

Os sistemas de software baseados em GUI representam mais de 60% dos softwares desenvolvidos hoje (ISSA et al., 2012). O teste GUI visual, um dos tipos de teste GUI, é mais flexível interagindo com o componente GUI através do reconhecimento de imagem e robusto a mudanças e comportamentos inesperados durante o sistema sob teste (BORJESSON; FELDT, 2012). As aplicações baseadas no teste GUI tipicamente requerem o uso de uma ferramenta (BANERJEE et al., 2013). As ferramentas de teste GUI populares (ex: Sikuli) dependem da memória do testador para lembrar que eventos foram executados pelos casos de testes gerados e acabam perdendo a visão do global do processo de teste, necessitando de uma quantidade significativa de esforço manual (XIE; MEMON, 2006).

Para cobrir a necessidade de um método de automatização de testes em aplicações móveis que ofereça a elaboração de scripts robustos, de fácil aprendizado e que seja flexível, o presente trabalho pretende utilizar uma ferramenta que facilite a escrita de scripts automatizados. Assim, foi escolhida a ferramenta Sikuli, que não possui dentre suas funções operações pré-programadas de apoio à mudança em contextos em aplicações móveis. Assim, esta terá suas funções estendidas para simular mudanças de contextos dos dispositivos móveis e avaliar o comportamento dos aplicativos de acordo com essas mudanças, oferecendo testes que possam ser executados mais rapidamente e frequentemente sem demandar grande esforço manual.

1.2 Objetivo

Prover um apoio automatizado à realização de testes baseados em interface gráfica para aplicações móveis sensíveis ao contexto usando os recursos de desenvolvimento de scripts da ferramenta Sikuli para plataforma Windows Phone.

1.3 Organização do Documento

O trabalho está organizado na seguinte estrutura: O capítulo 1 introduz o tema do trabalho, relatando motivação e problema, juntamente com o objetivo. No capítulo 2, o referencial teórico é descrito trazendo os conceitos de Computação Ubíqua, Sensibilidade ao Contexto, Teste para aplicações móveis e Teste GUI, com suas definições e técnicas. Ainda é apresentada a ferramenta de automação de teste baseada no reconhecimento de imagens Sikuli, utilizada neste trabalho, e como ela funciona. No capítulo 3, encontra-se a solução desenvolvida neste trabalho, com as novas funções implementadas na ferramenta Sikuli visando testes automatizados de aplicações móveis sensíveis ao contexto. Ainda no capítulo 3, é apresentada uma prova de conceito com a aplicação das funções desenvolvidas em três aplicações móveis desenvolvidas para a plataforma Windows Phone, apontando as virtudes e limitações das funções propostas. Finalmente, no capítulo 4 são apresentadas as conclusões finais e perspectivas de trabalhos futuros dando continuidade a este projeto.

→ Sentido de metodologia.

- Sugiro criar um tópico falando das ferramentas de automação de testes GUI, e dizer porque escolhem a

2 Referencial Teórico

sua.

O objetivo deste capítulo é definir os principais conceitos relacionados ao presente trabalho: Extensão da Ferramenta Sikuli para Apoiar Testes Automatizados para Aplicações Windows Phone Sensíveis ao Contexto. Na seção 2.1 é abordado o conceito de computação ubíqua e sua ligação com a computação pervasiva, passando para a definição dos conceitos de sensibilidade ao contexto. Na seção 2.2, uma visão geral sobre Teste para Aplicações Móveis é fornecida. Na seção 2.3 o Teste GUI é discutido, com a apresentação de definições de GUI e testes baseados em interface, além das principais técnicas de Teste GUI disponíveis na literatura técnica. Na seção 2.4 é apresentada uma visão geral sobre Sikuli.

2.1 Computação Ubíqua e Sensibilidade ao Contexto

De acordo com Ashraf e Khan (2013), os sistemas de computação são classificados em três fases que são:

1. Computação Mainframe: Interação de uma máquina com muitas pessoas.
2. Computação Desktop: Interação de uma pessoa com uma máquina.
3. Computação Ubíqua: Interação de uma pessoa com muitas máquinas.

Weiser (1991) definiu Computação Ubíqua como: “área de pesquisa que estuda a integração da tecnologia às atividades humanas de forma transparente, quando e onde for necessário”. Ele idealizou ambientes físicos com dispositivos computacionais integrados que auxiliariam indivíduos na realização de suas tarefas cotidianas ao fornecer-lhes informações e serviços de forma contínua e transparente.

A computação pervasiva, definida como um paradigma que permite ao usuário acesso a seu ambiente computacional independente da localização, tempo e equipamento é um conceito interligado à computação ubíqua (LOPES, 2008).

A proposta de Weiser (1991) tem sido concretizada gradativamente através de avanços tecnológicos e pela disponibilização dos dispositivos móveis. A computação ubíqua idealiza que os computadores estarão presentes em todos os lugares e não serão percebidos pelas pessoas, devido à capacidade de processamento.

Computação ubíqua é uma ampla área de estudo que contém muitos subdomínios, tais como sensibilidade ao contexto, escalabilidade, adaptabilidade, privacidade,

segurança, dentre outros (ASHRAF; KHAN, 2013). A sensibilidade ao contexto será o foco deste trabalho, e será descrita com mais detalhes em sua continuidade.

Contexto é definido como: qualquer informação que possa ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que está incluindo os usuários e suas aplicações (AMALFITANO et al., 2013).

Os dispositivos, serviços e componentes de software devem ser conscientes de seus contextos e automaticamente adaptar-se às suas mudanças, caracterizando a sensibilidade ao contexto (LOPES, 2008). Para Lopes (2008), a sensibilidade ao contexto tem duas grandes frentes:

1. Aquisição e tratamento de dados que expressam informações relevantes ao contexto.
2. Adaptação às alterações de contexto.

O mecanismo para sensibilidade ao contexto deve prover suporte para diferentes tarefas, dentre as quais:

- Obtenção do contexto de diversas fontes (sensores físicos e lógicos);
- Interpretação dos dados sensorados, gerando dados contextualizados;
- Disseminação das informações para as partes interessadas, de modo distribuído e personalizado.

Quanto mais formal o modelo para descrição do contexto, maior é a capacidade de realização de pesquisa e inferência sobre ele. Os tipos de modelo de contexto são:

- Pares chave-valor: a informação é descrita em pares, onde um atributo descreve a propriedade do contexto em questão.
- Baseados em linguagens de marcadores: contextos são representados por uma estrutura hierárquica, compostas de pares chave-valor e descrita por linguagens de marcadores, como dialetos do XML.
- Gráficos: modelado por meio de linguagens gráficas, tais como extensões da *Unified Modeling Language* (UML).
- Baseados em lógica: Contexto representado por meio de fatos, expressões e regras que definem como uma informação de contexto pode ser inferida ou derivada a partir de outra.

- Baseados em orientação a objetos: as informações de contexto são representadas por meio de objetos.
- Baseados em ontologias: contextos e seus relacionamentos são descritos por meio de ontologias.

Um fator complexo em relação ao desenvolvimento e teste em aplicações móveis é dado pela sensibilidade a mudanças de contexto em que elas executam. Por exemplo, aplicações executando em um smartphone podem ser influenciadas pelas mudanças de localização, níveis de bateria, tipo de conexão, chamadas, movimentos do aparelho e muitos outros tipos de eventos de contexto (AMALFITANO, 2013).

Segundo Amalfitano (2013), os atuais aparelhos móveis são equipados com uma ampla variedade de sensores de hardware que são capazes de sentir o contexto em que os aparelhos se submetem. A heterogeneidade das plataformas móveis implica em desenvolvimento e testes mais caros. A variabilidade das condições de execução em aplicação móvel depende de usá-la em contextos variáveis.

Sensibilidade ao contexto de aplicações móveis rende vários novos desafios para testes destas aplicações, já que elas devem ser testadas em qualquer ambiente e sob qualquer entrada contextual.

2.2 Teste para Aplicações Móveis

O número de smartphones em uso mundial superou a marca de um bilhão de unidades pela primeira vez no terceiro trimestre de 2012. As aplicações móveis não estão livres de falhas e novas estratégias de engenharia de software são requeridas para testar essas aplicações (KIRUBAKARAN; KARTHIKEYANI, 2013).

Segundo Huy e Thanh (2012), inicialmente as aplicações móveis eram classificadas em aplicações móveis nativas, aplicações web móvel, aplicações HTML5 ou widgets móveis. Porém, neste trabalho entende-se Widgets móveis e aplicações HTML5 como Aplicações Móveis Híbridas. Logo, será adotada a classificação: aplicações móveis nativas, aplicações web móvel e aplicações móveis híbridas.

A explosiva demanda para dispositivos móveis e por ambas as aplicações nativas e web para tais dispositivos tem sido acompanhada por uma necessidade de mais e melhores ferramentas de testes para aplicações móveis (GAO et al., 2014).

Segundo Gao et al. (2014), o termo teste móvel se refere a diferentes tipos de teste, tais como para o teste de aplicação móvel nativa, teste de dispositivo móvel e teste app web móvel. Teste de aplicações móveis se refere a “testar atividades para aplicações nativas e web no dispositivo móvel usando métodos de teste de software bem definidos e ferramentas para assegurar qualidade nas funções, comportamento, desempenho e qualidade do serviço bem como funcionalidades, tais como mobilidade, usabilidade, conectividade, segurança e privacidade”.

Vários requisitos exclusivos distinguem os testes de aplicações móveis de testes de software convencionais. Em princípio, aplicações móveis devem funcionar corretamente em qualquer hora e lugar. Além disso, os serviços móveis são frequentemente desenvolvidos para um conjunto de dispositivos selecionados, aplicações devem funcionar apropriadamente entre plataformas que tem, por exemplo, diferentes sistemas operacionais, tamanhos de tela e vida de bateria (GAO et al., 2014).

Dados esses requisitos, de acordo com Gao (2014) os testes de aplicações móveis tendem a focar nas seguintes atividades e objetivos:

- Teste de funcionalidade e comportamento: atividade que validam funções, APIS Web móveis, comportamentos externos do sistema e UI (User Interface).
- Teste de QoS (*Quality of Service*): atividades que avaliam a carga do sistema, performance, confiança, escalabilidade e taxa de transferência.
- Teste de interoperabilidade: atividades que avaliam a interoperabilidade do sistema entre os diferentes dispositivos, plataformas, navegadores e rede wireless.
- Teste de usabilidade e internacionalização: atividades que avaliam o conteúdo UI e alertas, fluxos de operação do usuário e cenários, a riqueza da mídia, e suporte a interação de gestos.
- Teste de segurança e privacidade: atividades que avaliam a autenticação do usuário, segurança do dispositivo, segurança da sessão, segurança na comunicação móvel, segurança de transação, e privacidade do usuário.
- Teste de mobilidade: atividades que avaliam a segurança baseada na localização, perfis do usuário, dados do sistema, e dados do usuário.

- Teste de compatibilidade e conectividade: atividades que avaliam a o navegador móvel e a compatibilidade de plataforma e as diversas redes wireless.

As aplicações móveis nativas têm o foco de teste em funcionalidade e comportamento, requisitos QoS, usabilidade, segurança e privacidade. De acordo com Gao et al. (2014), as quatro abordagens de teste para aplicações móveis mais populares são:

- Teste baseado em emulação: Envolve um emulador de dispositivo móvel, o qual cria uma versão de máquina virtual do dispositivo móvel para estudo em um computador.
- Teste baseado no dispositivo: Requer a criação de um laboratório de teste e a compra de dispositivos móveis reais.
- Teste na nuvem: Cria uma nuvem de dispositivos móveis que pode suportar os serviços de teste em grande escala.
- *Crowd-Testing*: Usa engenheiros de teste contratados ou uma comunidade de usuários finais junto com uma infraestrutura de teste baseado em multidão e um servidor de gerenciamento de serviços para apoiar diversos usuários.

Testes de aplicações móveis automatizados levantam diversas questões: a falta de padronização na infraestrutura de teste móvel, na linguagem do script e nos protocolos de conectividade entre as ferramentas de teste móvel e a plataforma, a falta de uma infraestrutura de teste automatizada unificada e soluções que atravessem plataformas e navegadores na maioria dos dispositivos móveis. O que retorna ao desafio de teste, para lidar com a frequente atualização de dispositivos móveis e tecnologia, testadores precisam de um ambiente de teste reusável e com custo efetivo para teste de dispositivos moveis (GAO et al., 2014).

2.3 Teste GUI (Teste Baseado em Interface Gráfica)

O uso de software é bastante comum na vida das pessoas, e uma forma de interação importante e reconhecida entre o software e o usuário é feita pela interface gráfica (GUI – *Graphical User Interface*) que promove uma interação mais fácil e natural (ISSA et al., 2012). Embora a interface gráfica faça o software de fácil uso na perspectiva do

usuário, ela complica o processo de desenvolvimento de software. Assim, o Teste GUI é frequentemente visto como uma atividade difícil de ser realizada (RUIZ; PRICE, 2008).

2.3.1 Definições de Teste GUI

GUI consiste em componentes existentes em uma aplicação (ex: botões, menu, campos, etc). O usuário GUI interage com esses componentes, que por sua vez geram eventos. Além destes componentes visíveis na tela, o usuário também gera eventos usando dispositivos como mouse ou teclado. Testes GUI devem ser capazes de simular a entrada do usuário, sendo um processo de testar uma aplicação de software que possui *front-end* gráfico para garantir que ela atenda sua especificação. Cada sequência do comando GUI pode resultar num diferente estado, e um comando GUI deve ser avaliado em todos esses estados, o que aumenta a complexidade dos testes a serem realizados (MEMON et al., 2001).

Testar interações do humano com o software é mais difícil que testar programas de máquina, porque as sequencias de interações dos humanos não são conhecidas e o espaço de possibilidade é enorme (ABDULRAHMAN, 2014). Todas as técnicas de teste GUI estão em algum sentido simplificando o espaço de entrada manualmente ou automaticamente. Na mesma linha, técnicas que desenvolvem um oráculo de teste GUI – mecanismo que determina se um GUI foi executado corretamente para uma entrada de teste – estão baseados em simplificar o espaço de saída (BANERJEE et al., 2013).

2.3.2 Técnicas de Teste GUI

Uma das técnicas mais comum de teste GUI é a abordagem *Record and Replay* (R&R). R&R é baseado num processo de dois passos onde as entradas de mouse e teclado do usuário são primeiramente gravadas e automaticamente armazenadas em um script em que a ferramenta pode então reproduzir no segundo passo (BORJESSON; FELDT, 2012). Durante a fase de captura, o testador interage manualmente com o GUI a ser testado e executa eventos. A ferramenta grava a interação. O testador também afirma manualmente que certo atributo de um específico componente seja armazenado. O caso de teste gravado pode ser reproduzido automaticamente no software usando a parte de reprodução da ferramenta. As assertivas podem ser usadas para verificar a correta execução do GUI. Devido a isso, essas ferramentas requerem uma quantidade significativa de esforço manual (XIE; MEMON, 2006).

Teste GUI visual (VGT, em inglês *Visual GUI Testing*) é muito similar à abordagem R&R, mas com a importante diferença que as ferramentas de R&R não usam reconhecimento de imagem e são assim mais codificadas para o exato posicionamento dos elementos GUI. Nas ferramentas atuais de teste GUI visual, a abordagem comum é os cenários serem escritos manualmente nos scripts que incluem imagens para o sistema a ser testado (SUT, em inglês *System Under Test*). O script de entrada é dado para o SUT através de comandos do mouse e teclado ao componente bitmap GUI identificado através do reconhecimento de imagem. A saída é observada novamente pelo reconhecimento de imagem e comparada ao resultado esperado depois que a próxima sequência da entrada é dada para o SUT (BORJESSON; FELDT, 2012).

Outra abordagem para o teste GUI é baseada em modelos. Nesta técnica, casos de testes são gerados a partir da interação de um modelo que represente a GUI do sistema e que leva em consideração as possíveis sequências de eventos tratados entre a interface e o usuário. Depende muito da informação do tempo de execução para a geração de suítes de teste GUI. O modelo representa a descrição do comportamento do software. O exemplo comum é máquina de estados finitos (MEF, em inglês, FSM – *Finite State Machine*). O modelo representa o comportamento em termos do diagrama de transição de estados. Uma MEF é uma representação de uma máquina compostas por estados e eventos que correspondem a transições entre os estados. Uma transição é caracterizada por dois eventos: um de entrada e um de saída. A máquina pode estar em apenas um estado por vez. Ao ocorrer um evento de entrada, a máquina pode responder com um evento de saída e uma transição para outro estado (podendo ser o mesmo estado) (PINHEIRO, 2012).

Alégroth et al. (2013) afirma que a técnica de teste GUI baseada no reconhecimento de imagem com os cenários baseados em scripts, VGT, pode emular um usuário humano e portanto também podem testar todas as aplicações e apesar de ser sensível a alterações gráficas de GUI, como forma, tamanho e cor, essa técnica não sofre a limitação de ser sensível a mudanças de layout, API, ou mesmo alterações no código, tal técnica foi escolhida para o desenvolvimento do trabalho.

2.4 Sikuli

Testar o comportamento visual de um GUI requer testadores humanos para interagir com o GUI e observar se o resultado esperado da interação é apresentado (CHANG et

al., 2010). Nesse cenário, uma nova abordagem para teste GUI usando visão computacional para testadores automatizarem suas tarefas foi proposta.

Sikuli é uma ferramenta que utiliza a abordagem visual para busca e automatização de interfaces gráficas dos usuários utilizando *screenshots*. Sikuli é um projeto de pesquisa open-source originalmente iniciado pelo grupo de interface gráfica do usuário do MIT em 2009. É possível automatizar qualquer componente gráfico visível na tela. É muito útil quando não existe acesso ao interior de um GUI ou ao código fonte. Através dessa ferramenta, testadores podem escrever scripts visuais que usam imagens para especificar com qual componente GUI interagir e qual o retorno visual a ser esperado, possui a vantagem de ser independente de qualquer plataforma (CHANG et al., 2010).

Sem sentido, arrumar

Na linguagem dos Índios Huichol (grupo étnico indígena do México), Sikuli significa olhos de Deus, a habilidade de ver e entender.

2.4.1 Como funciona

Sikuli é composto por duas partes: Sikuli Script e Sikuli IDE, conforme a Figura 1.

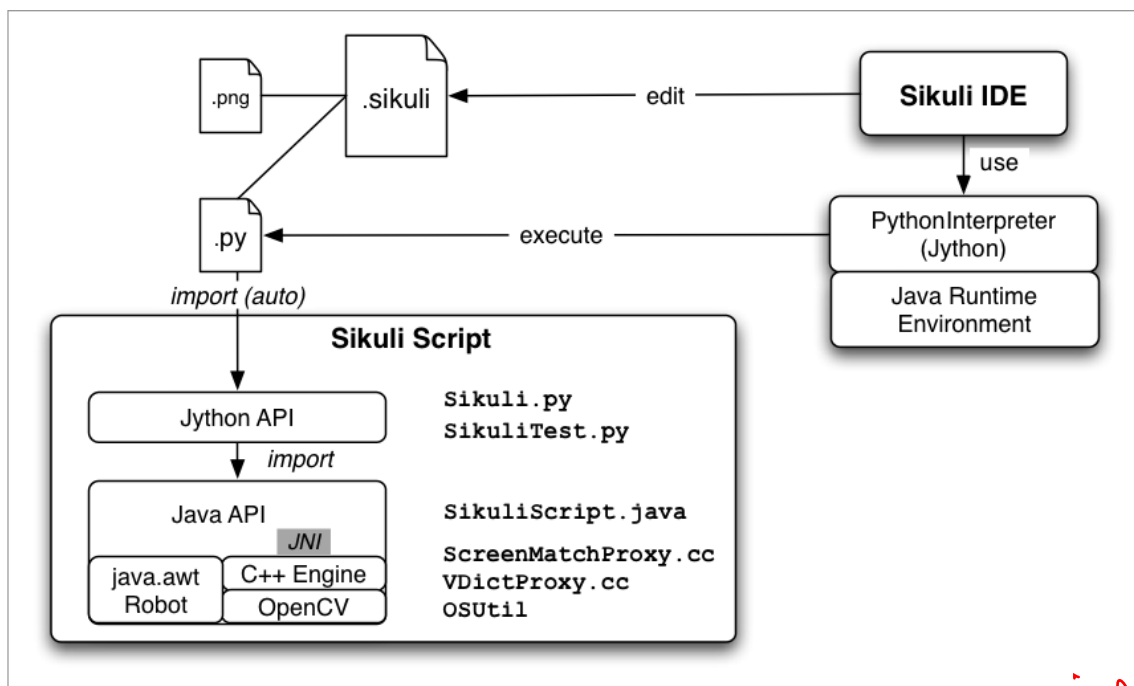


Figura 1 – Funcionamento do Sikuli

Fonte: HOCHE – How Sikuli Works

3-? - referencia melhor

Sikuli Script é uma biblioteca Java e Jython que automatiza as interações GUI usando padrões de imagens a eventos controlados pelos eventos de teclado/mouse. O núcleo do Sikuli script é uma biblioteca Java que consiste de duas partes:

java.awt.Robot, que entrega os eventos de teclado e mouse para as posições apropriadas, e uma engine C++ baseada no OpenCV, que procura o padrão de dada imagem na tela. A engine C++ é conectada ao Java via JNI (*Java Native Interface*) e precisa ser compilada para cada plataforma. No topo da biblioteca Java, uma fina camada Jython é fornecida para usuários finais como um conjunto de comandos simples e limpos. Portanto, deve ser fácil adicionar mais camadas finas para outras linguagens rodando em JVM, como JRuby, Javascript, dentre outras.

Sikuli IDE é o ambiente de desenvolvimento integrado para escrever os scripts utilizando os *screenshots* (capturas de telas). Os scripts visuais criados são processados pela API do Sikuli Script (HOCKE, 2014).

2.4.2 Interface Sikuli IDE

Na Figura 2, ao lado esquerdo são exibidos os principais comandos que compõe o script: *Find*, *Mouse Actions*, *Keyboard Actions*. Os componentes gráficos podem ser capturados pelas opções *Take screenshot* ou *Insert image* e utilizados diretamente no script. Para executar o script basta clicar no botão play.

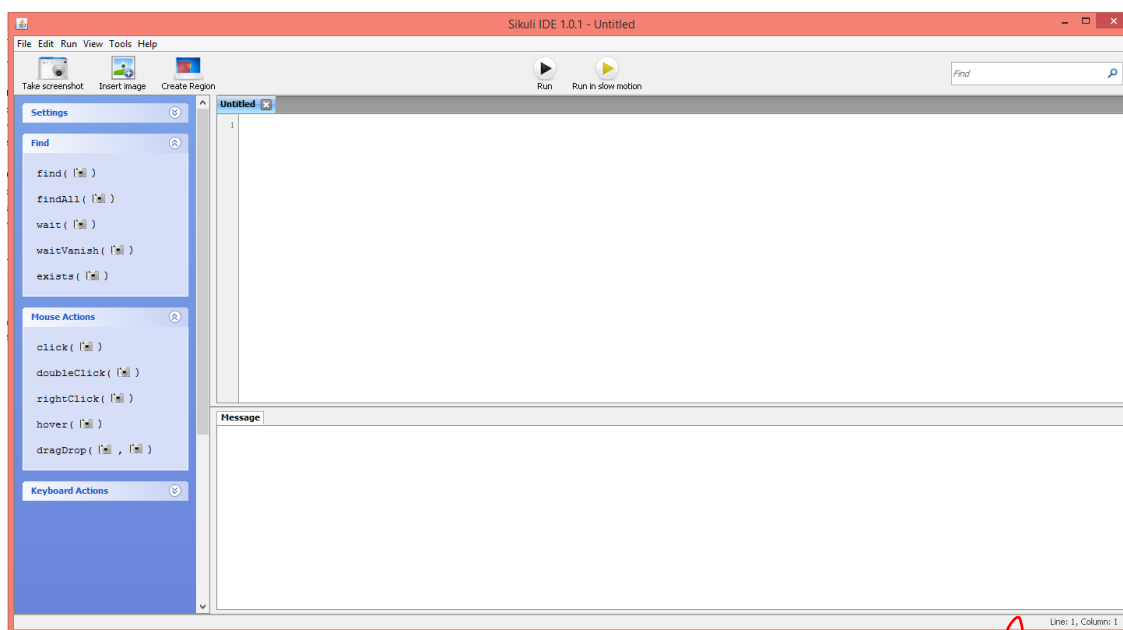


Figura 2 - Interface Sikuli IDE

Fonte: Elaborado pela autora

Isso é realmente necessário?

2.4.3 Configurações de padrão

Após as imagens serem inseridas no script, suas propriedades de configurações de padrão podem ser alteradas através das abas *Matching Preview* e *Target Offset*, conforme a Figura 3.

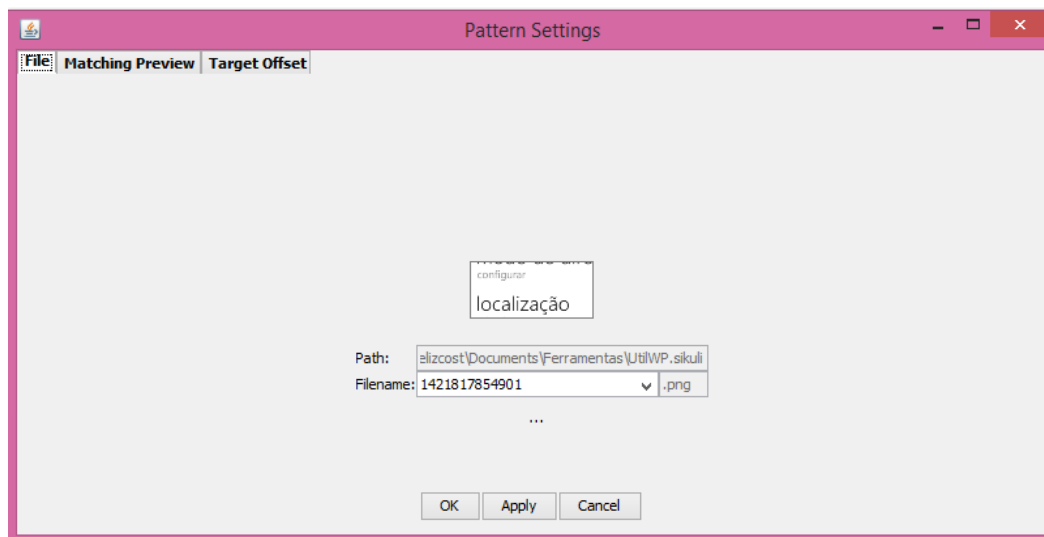


Figura 3 – Configurações de padrão de uma imagem

~~Fonte: Elaborado pela autora~~

Na Figura 4, a aba de *Matching Preview* possibilita definir o nível de similaridade da imagem inserida no script com a tela através do índice de similaridade que pode ser aumentado ou diminuído. Seu uso é recomendado quando na execução as imagens não estão sendo correspondidas e deseja-se melhorar a eficácia da identificação das imagens no script.

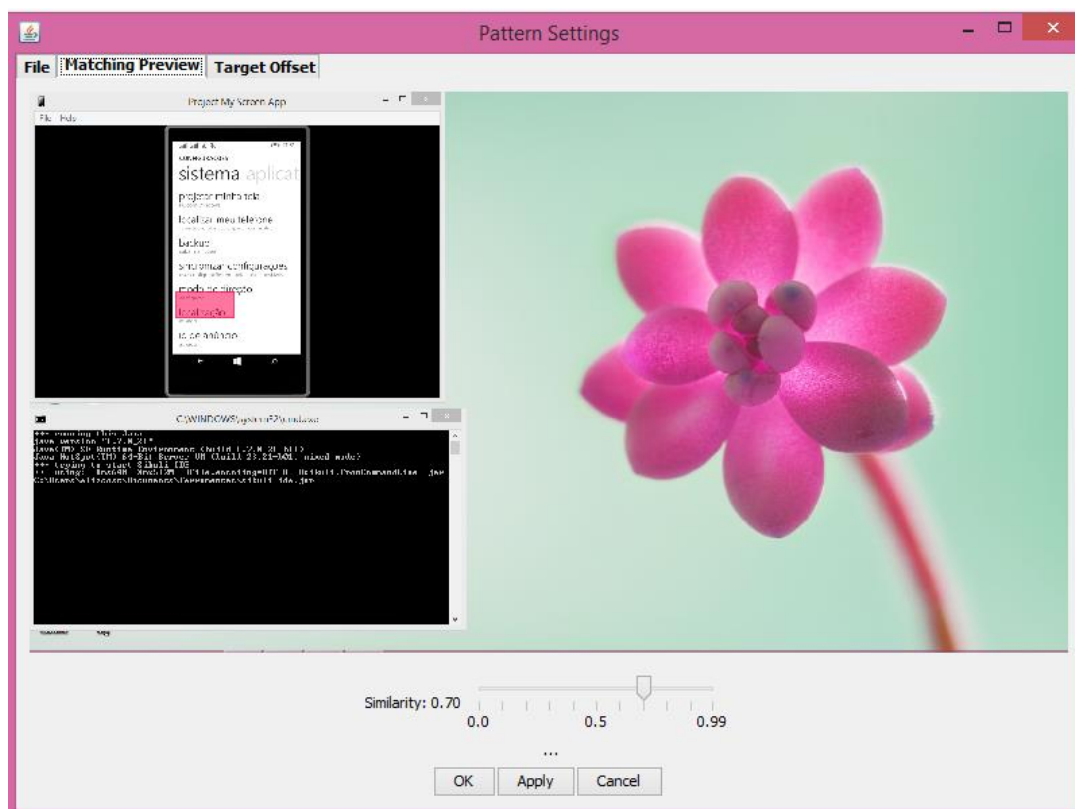


Figura 4 - Propriedade *Matching Preview*

~~Fonte: Elaborado pela autora~~

Na Figura 5, a aba *Target Offset* define o ponto alvo para o clique no componente. É útil quando um clique em uma posição específica da imagem implica na ação esperada. Utiliza-se as coordenadas X, Y do *grid view* do Sikuli.

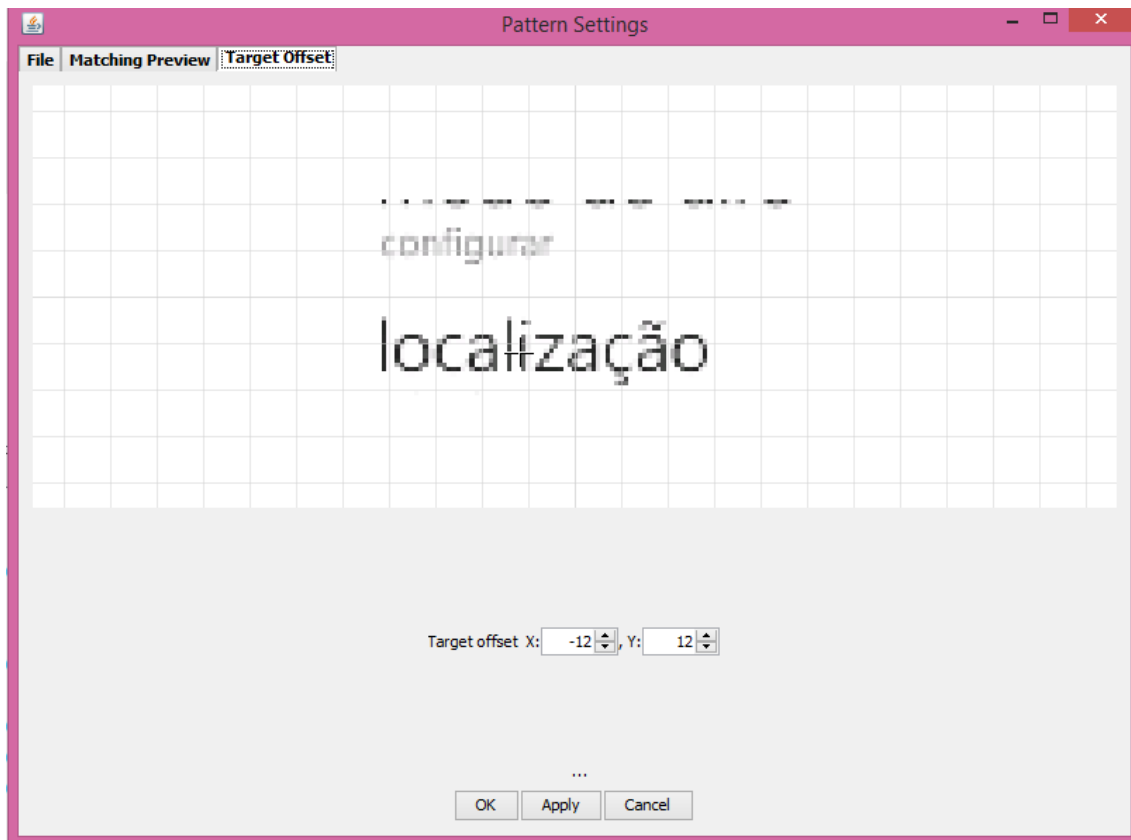


Figura 5 – Propriedade Target Offset



Fonte: Elaborado pela autora


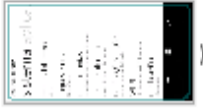
Além dessas configurações de propriedade, eliminar complexos papéis de parede do desktop serve como uma boa prática na elaboração dos scripts que evita confundir os algoritmos de reconhecimento de imagem do Sikuli, assim como eliminar os ícones do desktop que não serão necessários ao Sikuli.

2.4.4 Exemplo de Script

Na Figura 6, uma ação para abrir a tela de configurações do celular é automatizada. Primeiramente, verifica-se a orientação atual do celular – função *VerifyRotation()*. Conforme seja a orientação, o script abre a área de notificações – função *OpenNotification()*, clica na opção todas as configurações – comando *click()*, e espera que a tela apareça – comando *wait()*. Caso ocorra algum erro em um desses passos e não seja possível abrir a tela de configurações, uma mensagem é exibida na tela de saída do Sikuli.

```

def GoToSettings():
    rotation=VerifyRotation()
    if rotation=='portrait':
        try:
            OpenNotification()
            click()
            wait()
        except FindFailed:
            print 'Portrait - Settings not found'

    if rotation=='landscape':
        try:
            OpenNotification()
            click()
            wait()
        except FindFailed:
            print 'Landscape - Settings not found'

```

Figura 6 – Exemplo de script que automatiza a ação de abrir a tela de configurações

Fonte: Elaborado pela autora

2.4.5 Mensagens do Sikuli

Quando a execução dos scripts chega ao fim, mensagens do log das ações realizadas e o estado atual de contexto do dispositivo são exibidos, como mostra a Figura 7.

Message
Rotation: portrait
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
Battery level: 37%
Battery: medium
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
Wifi connected
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(367,164)@S(0)[0,0 1600x900]
[log] CLICK on L(260,278)@S(0)[0,0 1600x900]
Location: off

Figura 7- Mensagem ao finalizar execução do script

Fonte: Elaborado pela autora

Sikuli ainda possui algumas limitações, como: a necessidade do ambiente estar visível na tela – caso não esteja, os componentes descritos no script não serão encontrados e erros ocorrerão se não forem tratados; Resolução de telas diferentes – Sikuli não reconhece as imagens dos scripts se forem utilizadas por um monitor que tenha resolução diferente da utilizada na hora da criação do script; Similaridade – as imagens podem ser confundidas com componentes que não são os ideais e se um alto nível de similaridade for definido na propriedade da imagem não é garantido que a imagem será reconhecida nas próximas execuções.

2.4.6 Criando Novas Funções e Bibliotecas em Sikuli

A estrutura de uma função no Sikuli é a mesma de uma função criada na linguagem Python: Uma palavra reservada *def*, o nome da função, parênteses (contendo parâmetros ou não) e dois pontos. Também é possível receber entradas do usuário pelo comando *input()*. Na Figura 8, é demonstrado um exemplo de criação de função que recebe o nome do usuário como parâmetro.

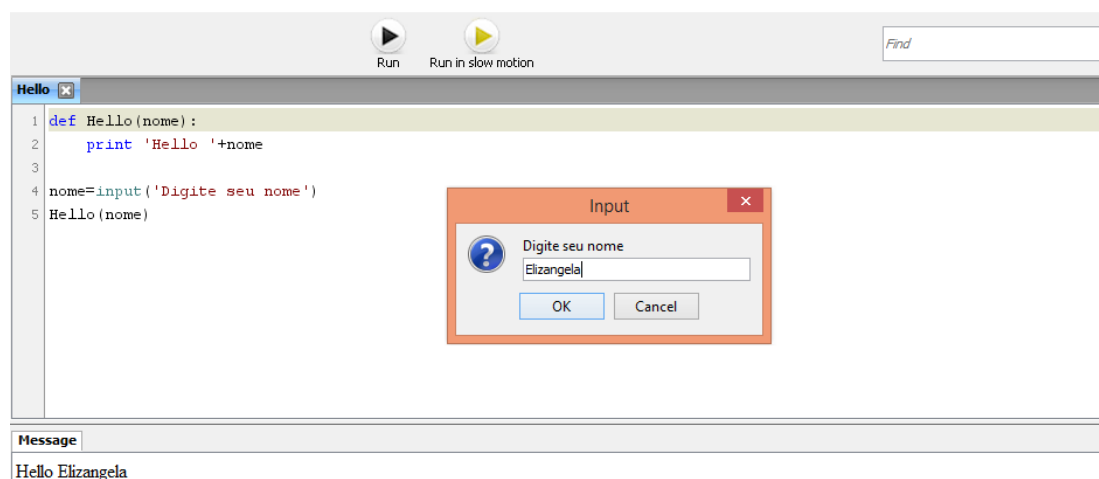


Figura 8 - Estrutura de uma função

Fonte: Elaborado pela autora

Bibliotecas podem ser criadas permitindo importar diversos scripts. Para isso, todos os arquivos de scripts devem estar em um mesmo diretório. Em cada script que será importado deve ser inserido na primeira linha o texto: *from sikuli import **, fazendo com que todos os elementos como imagens, variáveis, dentre outros, sejam encontrados no script principal. E no script principal, para importar scripts devem ser inseridas as linhas: *import exemploScript, reload(exemploScript), from exemploScript import **. Para um melhor entendimento, um script é importado na Figura 9. O script *Import* apenas chama a função criada no script *Hello*.

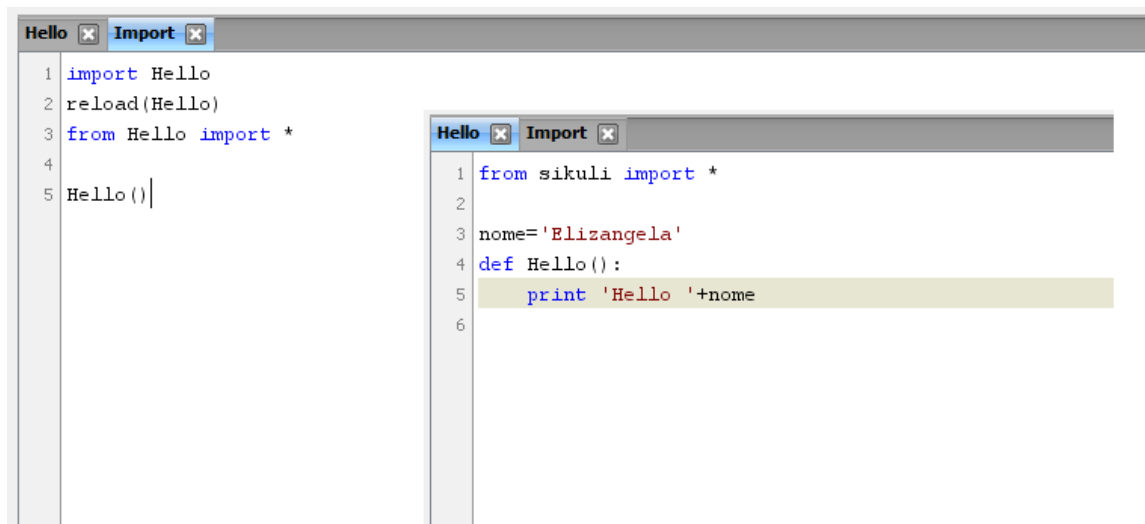


Figura 9 - Importando scripts

Fonte: Elaborado pela autora

3 Estendendo a Ferramenta Sikuli para Testes em Aplicações Móveis Sensíveis ao Contexto

O capítulo apresenta as seguintes seções: seção 3.1 - Novas Funções para Testes em Aplicações Móveis Sensíveis ao Contexto, descrevendo as funções implementadas no trabalho, seção 3.2 - Prova de Conceito, aplicando as funções na automatização dos testes de um aplicativo e seção 3.3 – Considerações sobre o uso das Funções Implementadas.

3.1 Novas Funções para Testes em Aplicações Móveis Sensíveis ao Contexto

O trabalho foi desenvolvido ~~nos seguintes passos:~~

através das seguintes etapas:

1. Escolha dos contextos que serão verificados na automatização dos testes;
2. Definição das variações dos elementos de contexto;
3. Desenvolvimento dos scripts;
4. Definição de aplicativos que sejam sensíveis ao contexto dos elementos selecionados;
5. Projeção do aplicativo da plataforma Windows Phone através do emulador *Project My Screen App*;
6. Execução dos scripts;
7. Comparação dos resultados após execução dos scripts.

No primeiro passo, os elementos de contextos do dispositivo definidos para serem automatizados foram: rotação do celular, bateria do celular, conexão com internet e localização.

Realizando o segundo passo, todos os elementos de contexto escolhidos possuem variações. A orientação do celular pode estar posicionada na vertical ou na horizontal, a bateria possui diversos níveis representados por porcentagem, a conexão pode ser ativada ou desativada tanto para Wi-Fi como para 3G, a localização pode estar ativada ou desativada.

No terceiro passo, foram desenvolvidos os scripts de automatização no Sikuli. No presente trabalho foram criados os seguintes scripts:

- **UtilWP:** Contém as principais funções para a automatização dos testes;
- **PC:** Script para a prova de conceito, onde as funções criadas no script *UtilWP* são chamadas automatizando testes nos aplicativos sensíveis ao contexto;
- **NivelBateria:** Script que armazena uma base de dados para as imagens dos níveis de bateria existentes;

- **NivelBateriaH:** Script que armazena uma base de dados para as imagens dos níveis de bateria existentes no modo horizontal;
- **Scroll:** Contém funções que realizam o movimento de deslize na tela na direção horizontal e vertical;
- **Alfabeto:** Script que armazena uma base de dados para as letras do alfabeto auxiliando na função de abertura de aplicativo.

No apêndice A estão contidas as funções do script *UtilWP*, citadas na Tabela 1.

Tabela 1. Descrição das funções contidas no script *UtilWp*.

Função	Descrição
<i>OpenApp(letra,app)</i>	Função para abrir o aplicativo, a primeira letra do aplicativo e a imagem do aplicativo são passadas como parâmetro.
<i>Home()</i>	Função que clica no botão home para ir à tela inicial do dispositivo móvel.
<i>OpenNotification()</i>	Função para abrir a tela de notificações onde se encontram atalhos, nível da bateria, opção de configurações, entre outros.
<i>GoToSettings()</i>	Função que abre a tela de configurações, com o intuito de alterar a propriedade de determinada opção dessa tela, como, por exemplo, desativar a localização.
<i>DisconnectWifi()</i>	Desconecta a conexão Wi-Fi pela tela de notificações.
<i>Disconnect3G()</i>	Desconecta a conexão 3G através da opção “rede celular + SIM” encontrada na tela de configurações.
<i>Disconnect()</i>	Desconecta conexão Wi-Fi e 3G chamando essas duas funções <i>DisconnectWifi</i> e <i>Disconnect3G</i> .
<i>ConnectWifi()</i>	Conecta a conexão Wi-Fi pela tela de notificações.
<i>Connect3G()</i>	Conecta a conexão 3G através da opção “rede celular + SIM” encontrada na tela de configurações.
<i>Connect()</i>	Conecta conexão Wi-Fi e 3G chamando essas duas funções <i>ConnectWifi</i> e <i>Connect3G</i> .
<i>VerifyRotation()</i>	Verifica a atual orientação do celular.
<i>ChangeToPortrait()</i>	Altera orientação do celular para a vertical.
<i>ChangeToLandscape()</i>	Altera orientação do celular para a horizontal.
<i>LocationOn()</i>	Ativa localização através da opção localização na tela de configurações.
<i>LocationOff()</i>	Desativa localização através da opção localização na tela de configurações.
<i>VerifyBattery(min,max)</i>	Verifica o nível atual de bateria, recebe como parâmetro um intervalo para comparação de níveis. O Script captura um <i>screenshot</i> da tela, compara a base de imagens dentro do intervalo definido com o <i>screenshot</i> da tela e retorna verdadeiro se o nível atual de bateria está contido no intervalo ou falso caso contrário.
<i>AssertEqual(img)</i>	Verifica se a imagem passada como parâmetro é igual à imagem exibida no dispositivo móvel.

O apêndice B contém as funções do script *Scroll*, citadas na Tabela 2, que realizam os movimentos para determinada direção na tela.

Tabela 2 - Descrição das funções contidas no script scroll

Função	Descrição
<i>Flick_up()</i>	Realiza movimentos do mouse para cima, esta função foi criada para visualizar as opções de configurações no modo vertical.
<i>Flick_left()</i>	Realiza movimentos do mouse para a esquerda, esta função foi criada para visualizar as opções de configurações no modo horizontal.
<i>Flick_right()</i>	Realiza movimentos do mouse para a direita, esta função foi criada para visualizar as aplicações abertas que estão executando em <i>background</i> .

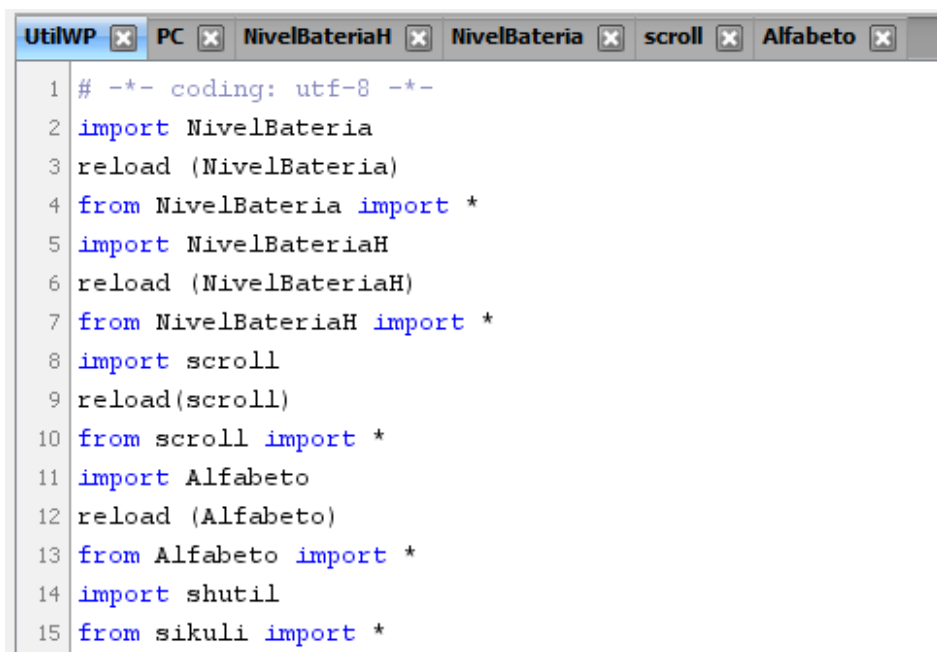
O apêndice C contém as funções principais do trabalho, que executam as mudanças de contexto e validam o comportamento das telas. Cada uma dessas funções está listada na Tabela 3 e será detalhada no próximo capítulo.

Tabela 3 - Descrição das funções contidas no script PC

Função	Descrição
GoogleTest()	Executa as funções de modificação na orientação e conexão, abre o aplicativo Google e verifica se o comportamento esperado foi alcançado.
WazeTest()	Executa as funções de modificação na localização, abre o aplicativo Waze e verifica se o comportamento esperado foi alcançado.
NivelBateriaApp()	Executa as funções de verificação do nível de bateria atual, abre o aplicativo Nível de Bateria e verifica se o comportamento esperado foi alcançado.

O script *UtilWP* importa o script *NivelBateria*, *NivelBateriaH*, *Scroll* e *Alfabeto* e bibliotecas necessárias, conforme a Figura 10.

No quarto passo, para analisar as funções criadas, foram selecionadas três aplicações móveis. Para avaliar a mudança de contexto da rotação e conexão com Internet, foi selecionado o aplicativo *Google*. A mudança de contexto da localização foi verificada utilizando-se o aplicativo *Waze*. Por fim, a mudança de contexto da bateria foi verificada por meio do aplicativo *Nível de Bateria*. Essas provas de conceito serão descritas na próxima seção.

A screenshot of a code editor window with multiple tabs at the top: 'UtilWP', 'PC', 'NivelBateriaH', 'NivelBateria', 'scroll', and 'Alfabeto'. The 'UtilWP' tab is active, displaying a Python script with 15 lines of code. The code imports several modules and reloads them: 'NivelBateria', 'NivelBateriaH', 'scroll', 'Alfabeto', 'shutil', and 'sikuli'.

```
1 # -*- coding: utf-8 -*-
2 import NivelBateria
3 reload (NivelBateria)
4 from NivelBateria import *
5 import NivelBateriaH
6 reload (NivelBateriaH)
7 from NivelBateriaH import *
8 import scroll
9 reload(scroll)
10 from scroll import *
11 import Alfabeto
12 reload (Alfabeto)
13 from Alfabeto import *
14 import shutil
15 from sikuli import *
```

Figura 10 - Importando scripts desenvolvidos no trabalho

Fonte: Elaborado pela autora

Antes da execução dos scripts, é necessário que a tela do aplicativo seja projetada na tela para a viabilidade de execução dos scripts automatizados do Sikuli no dispositivo móvel. Para essa finalidade, foi utilizado o emulador para Windows Phone *Project My Screen App*. Para projetar a tela, o dispositivo é conectado ao computador por um cabo USB. Ao abrir o emulador, uma tela de permissão é exibida no dispositivo solicitando a confirmação de exibição do conteúdo na tela, conforme a Figura 11.

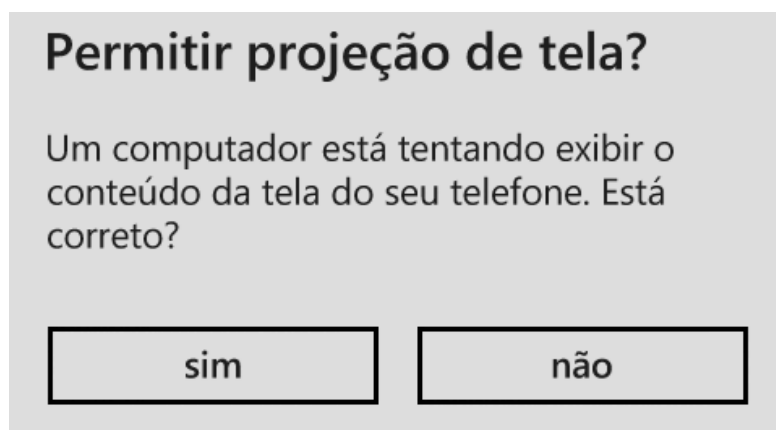


Figura 11 - Permissão de projeção da tela do celular

Fonte: Elaborado pela autora

Quando a permissão é confirmada, a tela é projetada pelo emulador – Figura 12. Após a tela ser exibida, é possível tanto controlar o celular por eventos do mouse como projetar em tempo real as ações realizadas através do celular. Esses são os procedimentos relacionados ao quinto passo.

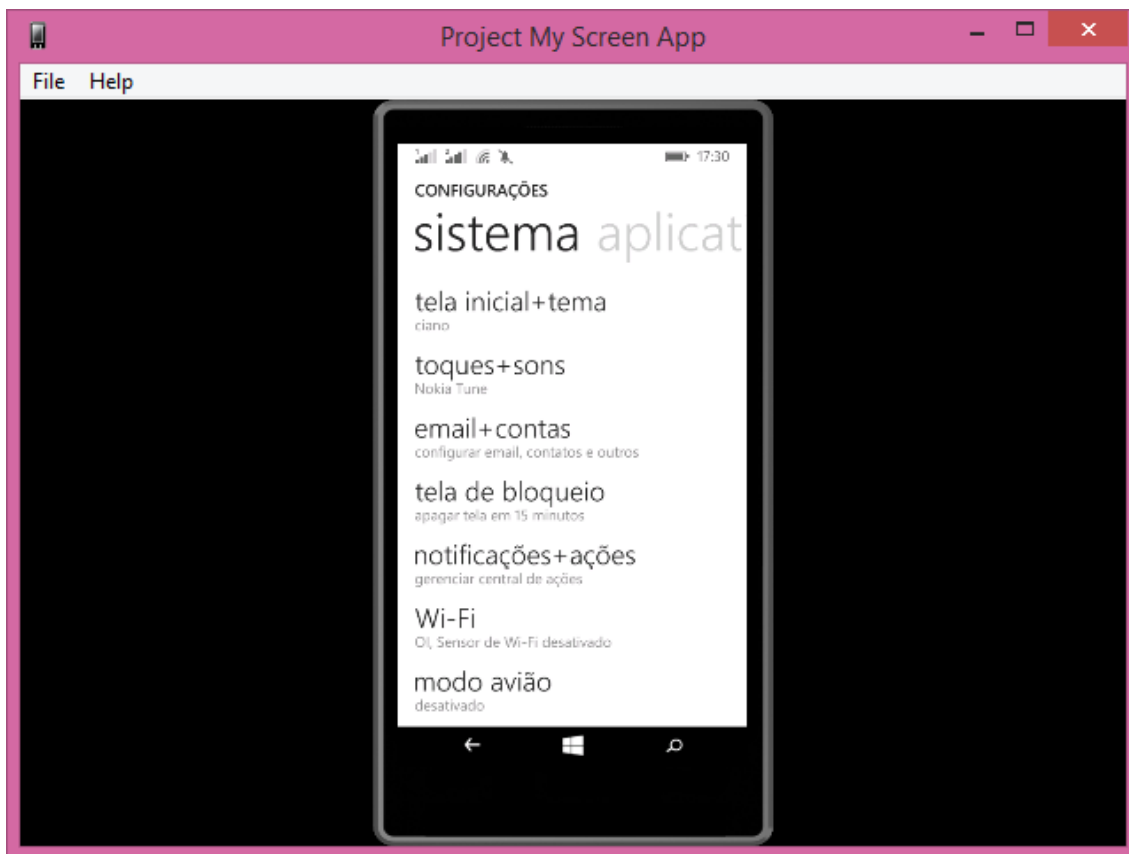


Figura 12 – Tela do celular sendo projetada pelo *Project My Screen App*

Fonte: Elaborado pela autora

No sexto passo, é realizada a execução dos scripts. Para se alcançar todas as funções criadas, os scripts foram executados a partir do script PC (sigla para prova de conceito que será abordado com mais detalhes no próximo capítulo).

Através desse script também é efetuada a comparação dos resultados após a execução dos scripts (sétimo passo). A comparação é feita entre a tela que deve ser o resultado esperado após as mudanças de contexto realizadas e a tela atual resultada após a execução. Caso a comparação entre as telas determine que elas sejam iguais, a automatização de teste relata que o teste passou; caso o comportamento da tela atual não seja igual ao comportamento da tela esperada, a automatização de teste relata que o teste falhou.

*Resultados?
Não sabe
resultados?*

Um exemplo pode ser extraído na Figura 13. Dada a orientação do celular como vertical, a chamada de função *ChangeToLandscape* deve mudar a orientação do celular para horizontal.



Figura 13 – Simulação de mudança de rotação
 Fonte: Elaborado pela autora

Caso esse comportamento ocorra, o caso de teste é aprovado; senão, se após a chamada de função *ChangeToLandscape* a orientação do celular não for mudada para a horizontal, o caso de teste falhou.

3.2 Prova de Conceito

Para validar a reação dos aplicativos móveis aos contextos, primeiramente foi analisado como eles deveriam se comportar em relação a determinadas mudanças de contextos do dispositivo.

3.2.1 Avaliação da Função de Orientação da Tela e Conexão com Internet

Alterações foram aplicadas nos contextos rotação do celular e conexão com internet do dispositivo para verificar como o aplicativo Google reagiria a essas alterações. Notou-se que de acordo com os estados dos contextos, o aplicativo se comporta da seguinte maneira: o aplicativo habilita o campo de busca e a opção de login somente se houver conexão com a Internet, demonstrado pela Figura 14.

Antes da execução do script é importante lembrar as boas práticas de desenvolvimento e execução do Sikuli discutidas na seção 2.4. A tela do emulador necessita estar em primeiro plano e também é recomendado que o celular esteja com a tela de bloqueio desativada, evitando problemas na hora da execução e comparação das imagens.

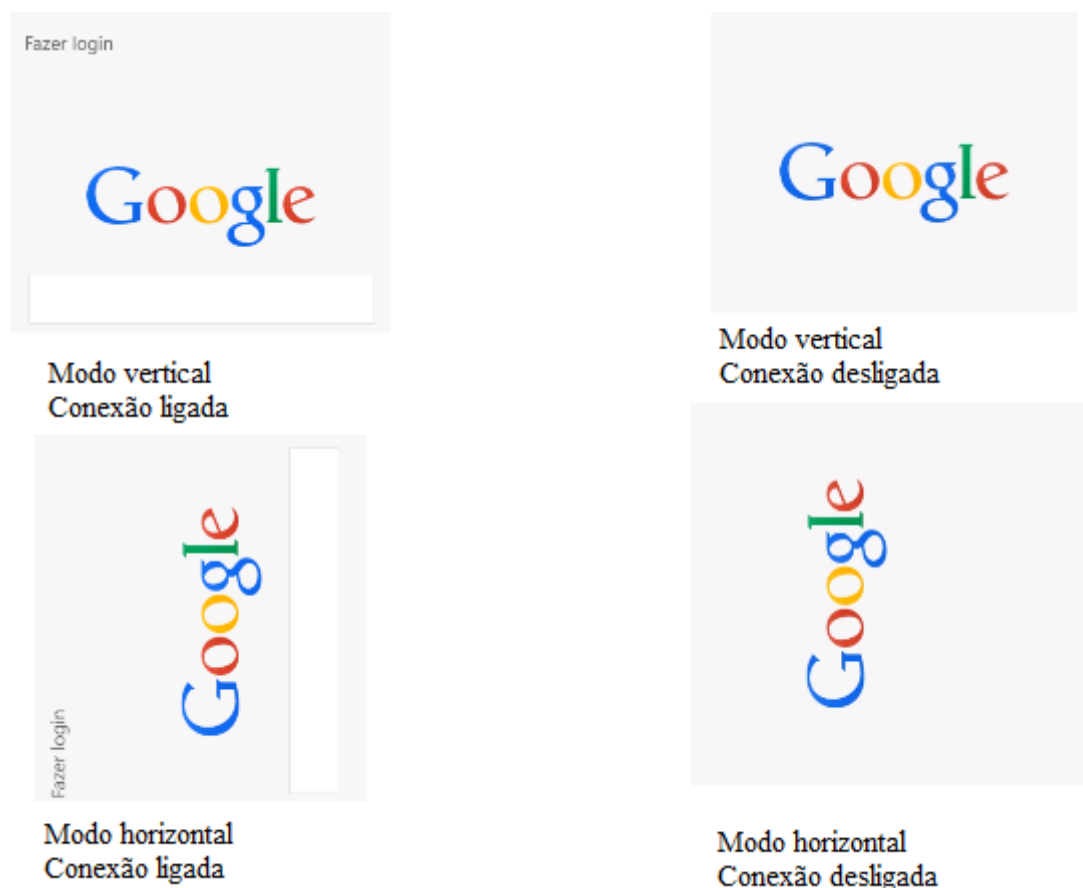


Figura 14 - Sensibilidade do aplicativo Google a mudanças de contexto

Fonte: Elaborado pela autora

A primeira automatização feita foi a ação de manter o celular na posição horizontal, seguida da ativação da Internet, o aplicativo então é aberto e se compara o comportamento esperado ao comportamento atual. A conexão com a Internet é então desativada e novamente verificada se a tela do aplicativo reage da maneira correta. O emulador simula o dispositivo na posição vertical com a conexão desligada e a observação do aplicativo é feita, finalmente se verifica o aplicativo no modo vertical com a conexão ligada. O fluxo pode ser observado na Figura 15 e o script pode ser consultado no apêndice C.

Após a finalização da função de teste para a aplicação móvel do Google, a Figura 16 exhibe o resultado. É importante ressaltar que o ambiente deve ter Wi-Fi ou o dispositivo tenha Internet móvel para que as telas se comportem da maneira prevista. Scripts para verificação de queda na conexão ou potência do sinal não foram implementados.

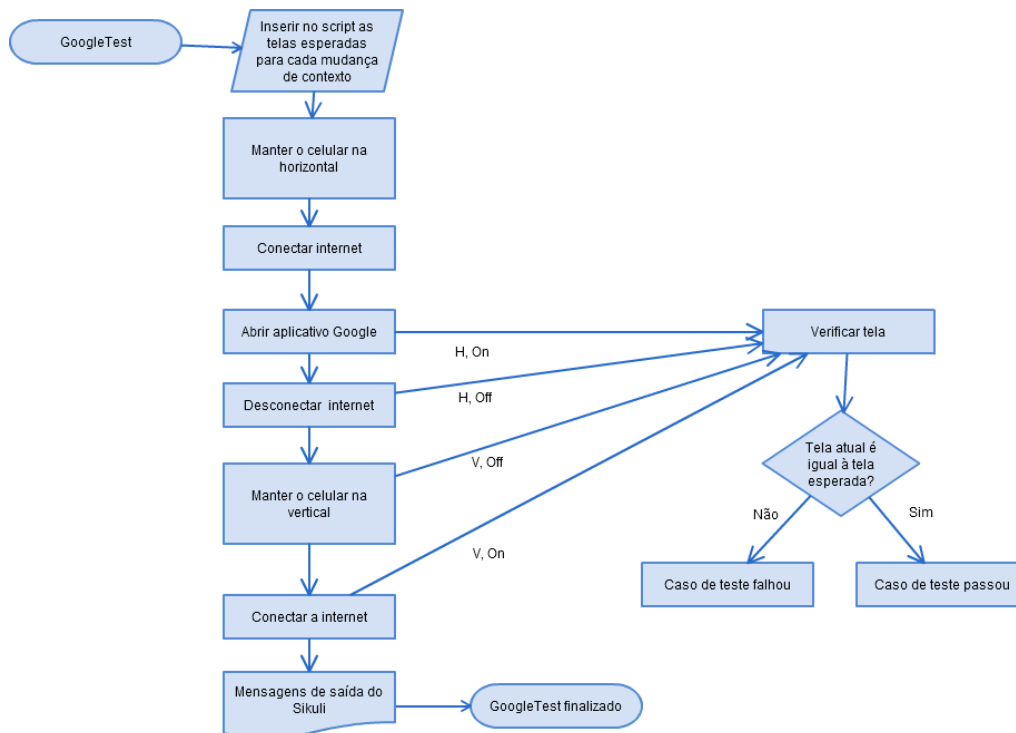


Figura 15 – Fluxo de automatização para o teste de orientação e conexão no aplicativo Google
 Fonte: Elaborado pela autora

Message	
[log] App.focus Project My Screen App(0) #0	
[log] TYPE "#LEFT."	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
Wifi connected	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
[log] CLICK on L(229,211)@S(0)[0,0 1600x900]	
[log] CLICK on L(371,286)@S(0)[0,0 1600x900]	
3G connected	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
[log] CLICK on L(166,309)@S(0)[0,0 1600x900]	
[log] CLICK on L(208,193)@S(0)[0,0 1600x900]	
[log] CLICK on L(186,277)@S(0)[0,0 1600x900]	
--Landscape connected--	
Passed	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
[log] CLICK on L(183,342)@S(0)[0,0 1600x900]	
[log] CLICK on L(234,192)@S(0)[0,0 1600x900]	
Wifi disconnected	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
[log] CLICK on L(229,211)@S(0)[0,0 1600x900]	
[log] CLICK on L(371,286)@S(0)[0,0 1600x900]	
[log] CLICK on L(254,192)@S(0)[0,0 1600x900]	
3G disconnected	
[log] CLICK on L(480,262)@S(0)[0,0 1600x900]	
[log] CLICK on L(166,309)@S(0)[0,0 1600x900]	
[log] CLICK on L(208,193)@S(0)[0,0 1600x900]	
[log] CLICK on L(186,277)@S(0)[0,0 1600x900]	
--Landscape disconnected--	
Passed	

(continuação)

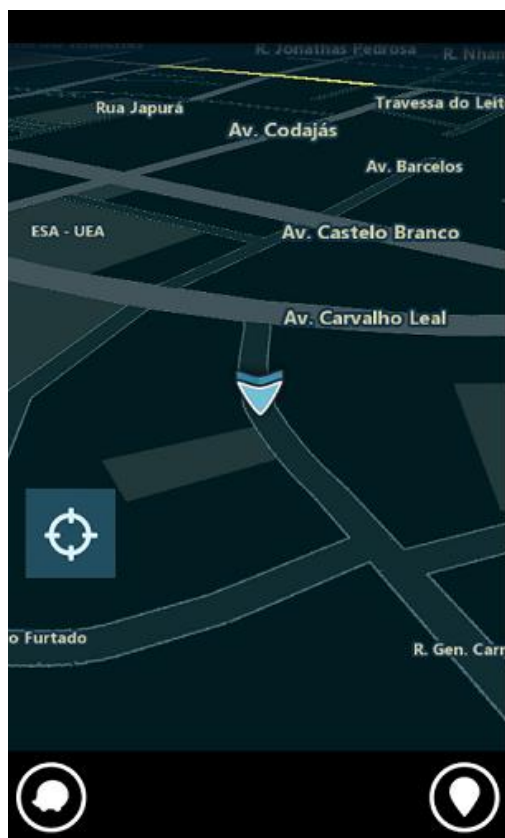
```

[log] TYPE "#DOWN."
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(271,109)@S(0)[0,0 1600x900]
[log] CLICK on L(387,149)@S(0)[0,0 1600x900]
[log] CLICK on L(300,126)@S(0)[0,0 1600x900]
--Portrait disconnected--
Passed
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(234,125)@S(0)[0,0 1600x900]
Wifi connected
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(367,164)@S(0)[0,0 1600x900]
[log] CLICK on L(314,308)@S(0)[0,0 1600x900]
[log] CLICK on L(391,192)@S(0)[0,0 1600x900]
3G connected
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(271,109)@S(0)[0,0 1600x900]
[log] CLICK on L(387,149)@S(0)[0,0 1600x900]
[log] CLICK on L(300,126)@S(0)[0,0 1600x900]
--Portrait connected--
Passed
  
```

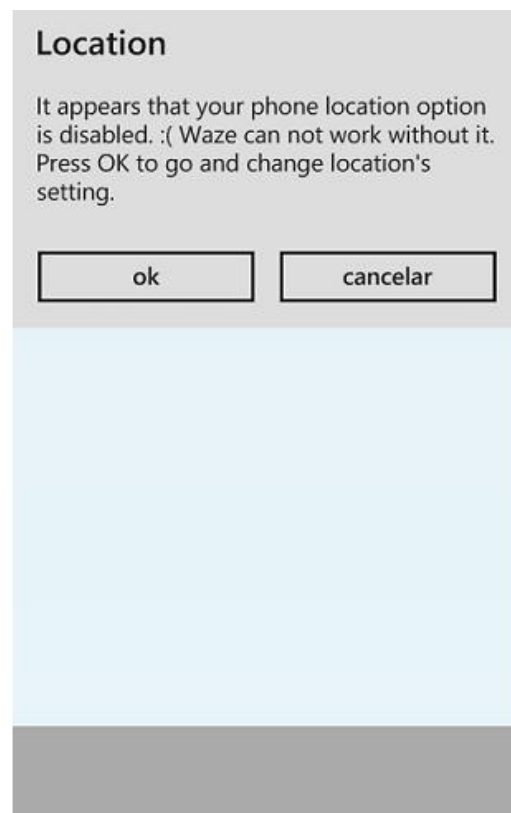
Figura 16 - Mensagem de saída do Sikuli para o teste do aplicativo Google
 Fonte: Elaborado pela autora

3.2.2 Avaliação da Função de Localização

Para avaliar como a mudança de contexto de localização afeta um aplicativo, foi estudado o comportamento do aplicativo *Waze*. Com a localização ativada, o aplicativo possui a opção de alertar e ir para o menu. Com a localização desativada, o aplicativo não permite o seu uso e uma mensagem de alerta é exibida ao usuário, conforme Figura 17. A mudança de contexto nesse aplicativo apenas é percebida quando ele é reaberto. O script pode ser consultado no apêndice C.



Localização ativada



Localização desativada

Figura 17 - Sensibilidade do aplicativo Waze a mudanças de contexto

Fonte: Elaborado pela autora

Após a finalização da função de teste para o Waze, a Figura 18 exibe o seguinte resultado esperado.

```
Message
[log] App.focus Project My Screen App(0) #0

[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(367,164)@S(0)[0,0 1600x900]
[log] CLICK on L(260,277)@S(0)[0,0 1600x900]
[log] CLICK on L(391,172)@S(0)[0,0 1600x900]
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(272,109)@S(0)[0,0 1600x900]
[log] CLICK on L(388,332)@S(0)[0,0 1600x900]
[log] CLICK on L(312,268)@S(0)[0,0 1600x900]
--Location connected--
Passed
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(367,164)@S(0)[0,0 1600x900]
[log] CLICK on L(260,279)@S(0)[0,0 1600x900]
[log] CLICK on L(388,172)@S(0)[0,0 1600x900]
[log] CLICK on L(361,163)@S(0)[0,0 1600x900]
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(273,109)@S(0)[0,0 1600x900]
[log] CLICK on L(388,332)@S(0)[0,0 1600x900]
[log] CLICK on L(299,269)@S(0)[0,0 1600x900]
--Location disconnected--
Passed
```

Figura 18 - Mensagem de saída do Sikuli para o teste do aplicativo Waze
Fonte: Elaborado pela autora

3.2.3 Avaliação da Função de Bateria

Por fim, para avaliar as alterações do elemento de contexto “bateria”, um intervalo é inserido pelo testador para determinar quais são os níveis alto, médio e baixo e qual a relação que esse intervalo tem com um aplicativo sensível ao contexto de bateria.

O aplicativo utilizado nesta avaliação foi o Nível de Bateria, que tem a finalidade de monitorar o uso de bateria, exibir notificações em determinado nível de bateria, apresenta dicas de economia, dentre outras funcionalidades que auxiliam o usuário a prolongar a vida útil da bateria. Um dos comportamentos dos aplicativos é a mudança de cor de acordo com o nível alto, baixo ou médio, sendo verde, amarelo e vermelho, respectivamente, conforme mostra a Figura 19.



Figura 19 – Sensibilidade do aplicativo Nível de Bateria a mudanças de contextos

Fonte: Elaborado pela autora

Ao executar o script, uma tela é exibida com a opção de escolha de validação dos níveis baixo, médio ou alto seguido da entrada dos valores mínimos e máximos de cada intervalo, conforme a Figura 20.

Input

Digite (baixa, media ou alta) para validar as cores da bateria

OK Cancel

Input

Entre com o valor minimo

OK Cancel

Input

Entre com o valor maximo

OK Cancel

Figura 20 - Entrada do usuário para o teste de bateria

Fonte: Elaborado pela autora

O script verifica se o nível de bateria atual do dispositivo está entre o intervalo inserido, para então abrir o aplicativo e validar se o comportamento esperado acontece. O script encontra-se no apêndice C. Ao finalizar, o resultado das mensagens geradas para um teste de bateria alta pelo Sikuli é demonstrado na Figura 21.

```
Message
[log] App.focus Project My Screen App(0) #0

[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(319,415)@S(0)[0,0 1600x900]
[log] CLICK on L(272,109)@S(0)[0,0 1600x900]
[log] CLICK on L(344,240)@S(0)[0,0 1600x900]
[log] CLICK on L(320,125)@S(0)[0,0 1600x900]
--High battery in green--
Passed
```

Figura 21 - Mensagem de saída do Sikuli para o teste do aplicativo Nível de Bateria
Fonte: Elaborado pela autora

3.3 Considerações sobre o uso das Funções Implementadas

Para implementar esta validação, foi necessária a criação de uma base de imagens para cada nível de bateria a fim de encontrar o nível de bateria atual e confirmar se o nível se encontra no intervalo dado na entrada, o que causou um grande tempo na elaboração do script. Isso foi necessário, pois o reconhecimento de texto dentro de uma imagem não sucedeu com sucesso pela ferramenta Sikuli.

Ficou estranho.
Revirar.

A fim de evitar erros que possam ser notados pelo usuário que utilizará a ferramenta, alguns cuidados foram tomados, tais como tratamentos de exceções para ocasionais imagens que não sejam encontradas, criação de regiões de imagens para elementos dificilmente identificados por conter imagens similares no restante da tela. Para imagens que possuem muitas correspondências na tela, o nível de similaridade deve ser ajustado para um número maior.

O padrão de estilo do dispositivo também deve receber atenção, dado que é possível mudar temas e planos de fundo. O estilo do dispositivo usado no trabalho tem a tela de fundo clara com a cor de destaque ciano. A troca no estilo deverá causar erros nos resultados dos scripts.

Inserção de comandos *waits()* com as imagens esperadas antes da verificação de igualdade entre as telas auxilia na prevenção de ocorrência de falhas devido a lenta ou

rápida execução das ações no emulador perdendo o tempo em que a imagem desejada se encontra na tela melhoram a eficácia dos scripts.

Verificar previamente se a imagem inserida no script será reconhecida pelo *matching preview* nas próximas execuções proporciona o reconhecimento prévio de falhas que podem ser geradas.

4 Considerações Finais

Os aplicativos móveis ganharam grande importância na engenharia de software devido à larga existência de dispositivos móveis que representam o principal recurso das atividades diárias tanto nas empresas como na vida pessoal. Métodos mais eficazes de desenvolvimento e teste são estudados continuamente para resultar na qualidade e correto funcionamento desses aplicativos.

Para contribuir com a metodologia de validação, o trabalho propõe utilizar uma ferramenta existente para automatização de testes Sikuli estendendo suas funções para a criação de outras novas que reduzam os esforços manuais para a atividade de teste. O trabalho foi desenvolvido especificamente para a plataforma Windows Phone, por possuir padronização de ícones e botões, porém sua lógica contribui para trabalhos futuros em outras plataformas, como por exemplo a plataforma Android, cuja automatização foi parcialmente desenvolvida.

Algumas limitações foram encontradas ao longo deste trabalho, dentre elas, as principais são notadas na instabilidade da ferramenta Sikuli, na restrita lista de aplicativos que poderiam ser automatizados visto que o emulador *Project My Screen* não ofereceu projeção adequada para aplicativos que possuíam animação, como os aplicativos *EasyTaxi* e *CandyCrush* – aplicativos que seriam inicialmente automatizados, e na restrição de sistema operacional para elaboração de scripts no Sikuli usando o *Project My Screen*, o qual deve ser obrigatoriamente Windows 8 ou superior, para suportar a funcionalidade de controlar o dispositivo pelo emulador.

Apesar das limitações existentes, o apoio oferecido para reduzir as tarefas manuais de teste comprova que é factível automatizar aplicações móveis sensíveis ao contexto observando suas características em mudanças de contexto. Os testes automatizados se transformam em solução para contribuir no desenvolvimento de um aplicativo confiável alcançado todos os possíveis eventos levando em consideração as entradas numerosas que podem existir entre a interface e o usuário.

- Dificuldades encontradas?

- Trabalhos futuros?

5 Referências

- ABDULRAHMAN, A. Test Automation for Graphical User Interfaces: A Review. 2014 World Congress on, Hammamet Computer Applications and Information Systems (WCCAIS), pp. 1-6, Jan. 2014
- ALEGROTH, E.; FELDT, R.; OLSSON, H.; Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study. 2013 IEEE Sixty International Conference on Luxemburgo Software Testing, Verification and Validation (ICST), pp. 56-65, Mar. 2013
- AMALFITANO, D.; FASOLINO, A.; TRAMONTANA, P.; AMATUCCI, N. Considering Context Events in Event-Based Testing of Mobile Applications. 2013 IEEE Sixty International Conference on, Luxemburgo Software Testing, Verification and Validation Workshops (ICSTW), pp. 126-133, Mar. 2013
- ASHRAF, M. U.; KHAN, N. A. Software Engineering Challenges for Ubiquitous Computing in Various Applications. 2013 11th International Conference on Frontiers of Information Technology (FIT), Islamabad, pp. 78-82, Dez. 2013
- BANERJEE, I.; NGUYEN, B.; GAROUSI, V.; MEMON, A. M. Graphical user interface (GUI) testing: Systematic mapping and repository. *Information and Software Technology*, v. 55, no. 10, pp. 1679-1694, 2013.
- BORJESSON, E.; FELDT, R. Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry. 2012 IEEE Fifth International Conference on, Montreal International Conference on Software Testing, Verification and Validation (ICST), pp. 350-329, 2012.
- CHANG, T. H.; YEH, T.; MILLER, R. GUI Testing Using Computer Vision. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, USA, pp. 1535-1544, 2010.
- FIGUEIREDO, C. F. P. A sensibilidade ao contexto na utilização de aplicativos móveis. 2011. 210 f. Dissertação (Mestrado em Comunicação Multimédia) – Departamento de Comunicação e Arte, Universidade de Aveiro, 2010.
- GAO, J.; BAI, X.; TSAI, W.; UEHARA, T. Mobile Application Testing: A Tutorial. *Computer*, vol. 47, no. 2, pp. 46-55, Fev. 2014.

- HOCKE, R. How Sikuli Works, 2014. Disponível em: <<http://sikulix-2014.readthedocs.org/en/latest/devs/system-design.html>>. Acesso em: 24 jan. 2015.
- HUY, N. P.; THANH, D. V. Evaluation of mobile app paradigms. MoMM, Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia. 2012, pp. 25-30.
- ISSA, A.; SILLITO, J.; GAROUSI, V. Visual testing of Graphical User Interfaces: An exploratory study towards systematic definitions and approaches. *2012 14th IEEE International Symposium on Web Systems Evolution (WSE)*, Trento, pp. 11-15, Set. 2012.
- KIRUBAKARAN, B.; KARTHIKEYANI, V. Mobile Application Testing – Challenges and Solution Approach through Automation. 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME), Salem, pp. 79-84, Fev. 2013.
- LOPES, J. L. B. *EXEHDA-ON: Uma Abordagem Baseada em Ontologias para Sensibilidade ao Contexto na Computação Pervasiva*. 2008. 128f. Dissertação (Mestrado em Ciência da Computação) – Escola de Informática, Universidade Católica de Pelotas, 2008.
- MEMON, A. M.; POLLAC, M. E.; SOFFA, M. L. Hierarchical GUI Test Case Generation Using Automated Planning. *IEEE Transactions on Software Engineering*, vol. 27, no. 2, pp. 144-155, Fev 2001.
- PINHEIRO, A. C. Subsídios para a aplicação de métodos de geração de casos de testes baseados em máquinas de estado. 2012. 95f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemática e de Computação, Universidade de São Paulo, 2012.
- RUIZ, A.; PRICE, Y. W. GUI Testing Made Easy. *Testing: Academic & Industrial Conference - Practice and Research Techniques*. IEEE 2008.
- WEISER, M. The Computer for the 21st Century. *Mobile Computing and Communications Review – Special Issue Dedicated to Mark Weiser*, vol. 3, no. 3, pp. 3-11, Julho 1999.

XIE, Q.; MEMON, A. M. Studying the Characteristics of a "Good" GUI Test Suite.
2006 Proceedings of the 17th International Symposium on Software Reliability
Engineering (ISSRE). Raleigh, NC, pp. 159-168, Nov. 2006

APÊNDICE A - Script UtilWP




```
def OpenApp (letra, app) :  
    Home ()  
  
    dragDrop ( ,  )  
  
    click (  )  
  
    for i in range (22) :  
        if (letra==alfa[i]) :  
            click (tecla[i])  
            break  
  
    click (app)  
    sleep (5)
```

Figura 22. Função OpenApp(letra,app)




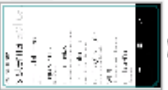


```
def GoToSettings() :  
    rotation=VerifyRotation()  
    if rotation=='portrait':  
        try:  
            OpenNotification()  
            click (  )  
            wait (  )  
        except FindFailed:  
            print 'Portrait - Settings not found'  
  
    if rotation=='landscape':  
        try:  
            OpenNotification()  
            click (  )  
            wait (  )  
        except FindFailed:  
            print 'Landscape - Settings not found'
```

Figura 23. Função GoToSettings()

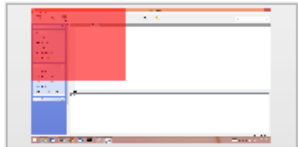



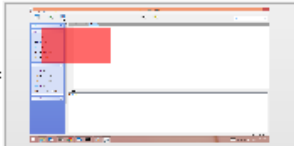



```

def Home():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            click()
        if rotation=='landscape':
            click()
    except FindFailed:
        print 'Home not found'

```

Figura 24. Função Home()



```



def OpenNotification():
    rotation=VerifyRotation()
    if rotation=='portrait':
        try:
            Home()
            r=

            r.dragDrop(, )
            wait()
        except FindFailed:
            print 'Notification not found'
    if rotation=='landscape':
        try:
            Home()
            r=

            r.dragDrop(, )
            wait()
        except FindFailed:
            print 'Notification not found'

```

Figura 25. Função OpenNotification()

```

def DisconnectWifi():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            OpenNotification()
            if exists():
                click()
                click(habilitado)
                if exists (desabilitado):
                    print "Wifi disconnected"
            else:
                print "Wifi disconnected"

        if rotation=='landscape':
            OpenNotification()
            if exists():
                click()
                click(habilitadoH)
                if exists (desabilitadoH):
                    print "Wifi disconnected"
            else:
                print "Wifi disconnected"
    except FindFailed:
        print 'Wifi not found'

```

Figura 26. Função DisconnectWifi()

```

def Disconnect():
    DisconnectWifi()
    Disconnect3G()

```

Figura 27. Função Disconnect()

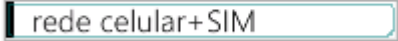
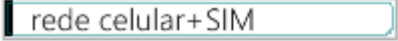
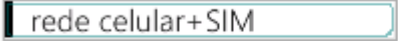
```

def Connect():
    ConnectWifi()
    Connect3G()

```

Figura 28. Função Connect()



```

def Disconnect3G():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            GoToSettings()
            for i in range(2):
                if exists():
                    break
                else:
                    flick_up()
            if exists():
                click()
                if exists(habilitado):
                    click(habilitado)
                    if exists (desabilitado):
                        print "3G disconnected"
                else:
                    print "3G disconnected"

```

Figura 29. Função Disconnect3G()

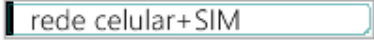
```

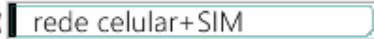
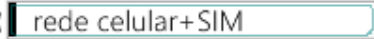
def VerifyRotation():
    sleep(2)
    if exists ():
        rotation = 'landscape'
        return rotation
    if exists ():
        rotation = 'portrait'
        return rotation

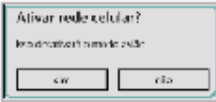
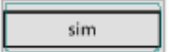
```

Figura 30. Função VerifyRotation()

```

def Connect3G():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            GoToSettings()
            for i in range(2):
                if exists():
                    break
                else:
                    flick_up()

            if exists():
                click()
                if exists(desabilitado):
                    click(desabilitado)

                if exists():
                    click()


                if exists (habilitado):
                    print "3G connected"
            else:
                print "3G connected"

```

Figura 31. Função Connect3G()

```






def ChangeToPortrait():
    #Troca posicao horizontal para vertical

    if exists ():
        type(Key.DOWN)
        rotation = 'portrait'
        return rotation
    else:
        rotation='portrait'
        return rotation

```

Figura 32. Função ChangeToPortrait()


```

if rotation=='landscape':
    GoToSettings()
    for i in range(2):
        if exists():
            break
        else:
            flick_left()
    if exists():
        click()
        if exists(desabilitadoH):
            click(desabilitadoH)
            #desabilitando modo avião
            if exists():
                click()
            if exists (habilitadoH):
                print "3G connected"
        else:
            print "3G connected"
except FindFailed:
    print '3G not found'

```

Figura 33. Função Connect3G() landscape

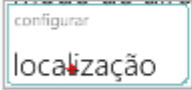

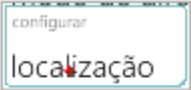

```

def ChangeToLandscape():
    #Troca posicao vertical para horizontal
    if exists (  ):
        type(Key.LEFT)
        rotation = 'landscape'
        return rotation
    else:
        rotation = 'landscape'
        return rotation

```

Figura 34. Função ChangeToLandscape()




```

def LocationOn():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            GoToSettings()
            for i in range(7):
                if exists(  ):
                    break
            else:
                flick_up()
        if exists(  ):
            click(  )
            if exists (desabilitado):
                click(desabilitado)
                if exists(habilitado):
                    location='on'
                    return location
            else:
                location='on'
                return location

```

Figura 35. Função LocationOn()

```

if rotation=='landscape':
    GoToSettings()
    for i in range(7):
        if exists():
            break
        else:
            flick_left()
    if exists():
        click()
        if exists (desabilitadoH):
            click(desabilitadoH)
            if exists(habilitadoH):
                location='on'
                return location
        else:
            location='on'
            return location
except FindFailed:
    print 'Location not found'

```

Figura 36. Função LocationOn() landscape

```

def AssertEqual(img):
    screen = Screen()
    file = screen.capture(screen.getBounds())


    shutil.move(file,os.path.join(getBundlePath(),)


    imagem = Finder()

    imagem.find(img)
    if imagem.hasNext():
        print ("Passed")
    else:
        print ("Failed")

```

Figura 37. Função AssertEqual(img)

```

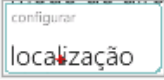
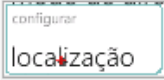
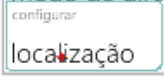



def LocationOff():
    rotation=VerifyRotation()
    try:
        if rotation=='portrait':
            GoToSettings()
            for i in range(7):
                if exists(

                ):
                    break
                else:
                    flick_up()
            if exists(

            ):
                click(

                )
                if exists (habilitado):
                    click(habilitado)
                if exists(desabilitado):
                    location='off'
                    return location
            else:
                location='off'
                return location
    
```

Figura 38. Função LocationOff()

```

if rotation=='landscape':
    GoToSettings()
    for i in range(7):
        if exists(

        ):
            break
        else:
            flick_left()
    if exists(

    ):
        click(

        )
        if exists (habilitadoH):
            click(habilitadoH)
            if exists(desabilitadoH):
                location='off'
                return location
        else:
            location='off'
            return location
except FindFailed:
    print 'Location not found'

```

Figura 39. Função LocationOff() landscape

```

def VerifyBattery(min,max):
    level=0
    rotation=VerifyRotation()
    sleep(1)
    if rotation=='landscape':
        try:
            OpenNotification()
            screen = Screen()
            file = screen.capture(screen.getBounds())

            shutil.move(file,os.path.join(getBundlePath(),

            tela = Finder(

            for i in range(min,max):
                imagem=NivelBateriaH.vH[i]
                tela.find(imagem)
                if tela.hasNext():
                    level='true'
                    break
                else:
                    level='false'
        except FindFailed:
            print "Failed"

```

Figura 40. Função VerifyBattery(min,max)

```

elif rotation=='portrait':
    try:
        OpenNotification()
        screen = Screen()
        file = screen.capture(screen.getBounds())

        shutil.move(file,os.path.join(getBundlePath(),

        tela = Finder(

        for i in range(min,max):
            imagem=NivelBateria.v[i]
            tela.find(imagem)
            if tela.hasNext():
                level='true'
                break
            else:
                level='false'
    except FindFailed:
        print "Failed"
return level

```

Figura 41. Função VerifyBattery(min,max) landscape

APÊNDICE B – Script scroll

```
def flick_up():
    x1, y1, x2, y2 = (370, 372, 370, 150)
    start = Location(x1, y1)
    end = Location(x2, y2)
    stepX = 0
    stepY = 20
    run = start
    mouseMove(start); wait(0.5)
    mouseDown(Button.LEFT); wait(0.5)
    run = run.right(stepX).above(stepY * 12)
    mouseMove(run)
    wait(0.05)
    mouseUp()
    wait(0.5)
```

Figura 42. Função flick_up()

```
def flick_left():
    x1, y1, x2, y2 = (400, 272, 670, 272)
    start = Location(x1, y1)
    end = Location(x2, y2)
    stepX = 0
    stepY = 20
    run = start
    mouseMove(start); wait(0.5)
    mouseDown(Button.LEFT); wait(0.5)
    run = run.right(stepX).left(stepY * 12)
    mouseMove(run)
    wait(0.05)
    mouseUp()
    wait(0.5)
```

Figura 43. Função flick_left()





```
def flick_right():  
    x1, y1, x2, y2 = (260, 248, 265, 248)  
    start = Location(x1, y1)  
    end = Location(x2, y2)  
    stepX = 0  
    stepY = 20  
    run = start  
    mouseMove(start); wait(0.5)  
    mouseDown(Button.LEFT); wait(0.5)  
    run = run.right(stepX).right(stepY * 8)  
    mouseMove(run)  
    wait(0.05)  
    mouseUp()  
    wait(0.5)
```

Figura 44. Função flick_right()

APENDICE C – Script PC

```
import UtilWP
reload (UtilWP)
from UtilWP import *

def GoogleTest():
    #Google
    #Orientação e Conexão
    #H on, H off, V off, V on

    landscape_Con=
    landscape_Coff=
    portrait_Con=
    portrait_Coff=

    switchApp("Project My Screen App")
    ChangeToLandscape()
    Connect()
    OpenGoogle()
    print '--Landscape connected--'
    wait(landscape_Con,5)
    AssertEqual(landscape_Con)

    Disconnect()
    OpenGoogle()
    wait(landscape_Coff,5)
    print '--Landscape disconnected--'
    AssertEqual(landscape_Coff)
```

Figura 45. Função GoogleTest()

```

ChangeToPortrait()
OpenGoogle()
print '--Portrait disconnected--'
wait(portrait_Coff,5)
AssertEqual(portrait_Coff)


Connect()
OpenGoogle()
print '--Portrait connected--'
wait(portrait_Con,5)
AssertEqual(portrait_Con)

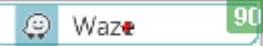
```

Figura 46. Função GoogleTest() continuação

```

def WazeTest():
    #Waze
    #Localização
    #On, Off

    locOn=
    locOff=

    switchApp("Project My Screen App")
    location=LocationOn()
    if (location=='on'):
        OpenApp('w',)
        print '--Location connected--'
        wait(locOn,5)
        AssertEqual(locOn)
    else:
        print 'Connect location failed'

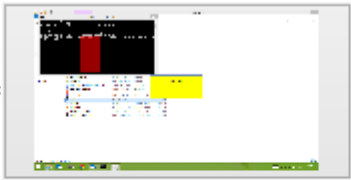
```

Figura 47. Função WazeTest()


```

location=LocationOff()
r=

```



```

try:
    if (location=='off') :
        dragDrop(
            90, 90
        )
        for i in range (5):
            flick_right()
            if exists(
                80, Waze
            ):
                r.click(
                    X
                )
                break
            OpenApp( 'w',
                Waze, 90
            )
            wait(locOff,5)
            print '--Location disconnected--'
            AssertEqual(locOff)
except FindFailed:
    print 'Disconnect location failed'

```

Figura 48. Função WazeTest() continuação

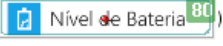
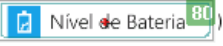
```

def NivelBateriaApp():
    #Entre 70% e 100%:
    green=
        80
    #Entre 35% e 70%
    yellow=
        80
    #Entre 35% e 2%
    red=
        80

```

Figura 49. Função NivelBateriaApp() - cores

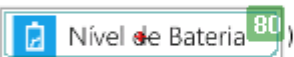
```

switchApp("Project My Screen App")
choice=input("Digite (baixa,media ou alta) para validar as cores da bateria")
if choice=='baixa':
    min=int(input("Entre com o valor minimo"))-1
    max=int(input("Entre com o valor maximo"))-1
    level=VerifyBattery(min,max)
    if level=='true':
        OpenApp('n', )
        print '--Low battery in red--'
        wait(red,5)
        AssertEqual(red)
    elif level=='false':
        print 'Battery is not in range'
elif choice=='media':
    min=int(input("Entre com o valor minimo"))-1
    max=int(input("Entre com o valor maximo"))-1
    level=VerifyBattery(min,max)
    if level=='true':
        OpenApp('n', )
        print '--Medium battery in yellow--'
        wait(yellow,5)
        AssertEqual(yellow)
    elif level=='false':
        print 'Battery is not in range'

```

Figura 50. Função NivelBateriaApp()

```

elif choice=='alta':
    min=int(input("Entre com o valor minimo"))-1
    max=int(input("Entre com o valor maximo"))-1
    level=VerifyBattery(min,max)
    if level=='true':
        OpenApp('n', )
        print '--High battery in green--'
        wait(green,5)
        AssertEqual(green)
    elif level=='false':
        print 'Battery is not in range'

```

Figura 51. Função NivelBateriaApp() - continuação