UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Testes Exploratórios: Características, Problemas e Soluções

Ana Karina Silva de Castro

Natal - RN
Novembro de 2018

Ana Karina Silva de Castro

Testes Exploratórios: Características, Problemas e Soluções

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada (DIMAp) do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte.

Orientador(a)
Dra. Roberta de Souza Coelho

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada - DIMAp

Natal - RN
Novembro 2018

Castro, Ana Karina Silva de.

Testes exploratórios: características, problemas e soluções / Ana Karina Silva de Castro. - 2018.

92f.: il.

Monografia (Bacharelado em Ciência da Computação) - Universidade Federal do Rio Grande do Norte, Centro de Ciências Exatas e da Terra, Departamento de Informática e Matemática Aplicada, Curso de Ciência da Computação. Natal, 2018.

Orientadora: Roberta de Souza Coelho.

1. Computação - Monografia. 2. Testes exploratórios - Monografia. 3. Ferramentas para testes exploratórios - Monografia. 4. Mapeamento sistemático - Monografia. I. Coelho, Roberta de Souza. II. Título.

RN/UF/CCET CDU 004

Monografia de Graduação sob o título *Testes Exploratórios: Características, Problemas e Soluções* apresentada por Ana Karina Silva de Castro e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

_____

Dra. Roberta de Souza Coelho
Departamento de Informática e Matemática Aplicada - DIMAP
Universidade Federal do Rio Grande do Norte - UFRN

_____

Dra. Lyrene Fernandes da Silva
Departamento de Informática e Matemática Aplicada - DIMAP
Universidade Federal do Rio Grande do Norte - UFRN

_____

Dra. Marcia Jacyntha Nunes Rodrigues Lucena
Departamento de Informática e Matemática Aplicada - DIMAP
Universidade Federal do Rio Grande do Norte - UFRN

Agradecimentos

Agradeço à minha família, em especial minha mãe Maria Rilza, minhas tias Edite Alves, Raimunda Alves, Dalila Alves e Antônia Alves, e meus irmão Ana Kalina e Valdeliz Neto pelo apoio que me deram durante todos esses anos, e que nunca me deixaram desistir do meu objetivo. Agradeço à minha orientadora Roberta Coelho, por me guiar quando eu estava perdida quanto ao escopo e possibilidades desta pesquisa, pelas ensinamentos e principalmente pela paciência e incentivo,. Agradeço ao professores pelo conhecimento compartilhado e aos colegas e amigos, que me acompanharam durante a graduação, principalmente a minha amiga Elaine Figueiredo, minha dupla constante nos diversos trabalhos feitos durante a graduação.

Testes Exploratórios: Características, Problemas e Soluções

Resumo

O teste de software é uma das atividades mais adotadas durante o processo de desenvolvimento de software para se avaliar corretude e a qualidade de um sistema de software. Estes testes podem ser manuais ou automatizados, muitos testadores realizam seus testes tendo como base um script onde se descreve o passo a passo do teste a ser seguido pelo testador, que tipos de entradas devem ser fornecidas ao sistema e quais saídas são esperadas. Quando o testador não faz uso de um script, a abordagem adotada se chama Teste Exploratório, onde o testador explora o sistema livremente a procura de bugs. Este trabalho apresenta o resultado de uma revisão da literatura cujo objetivo foi levantar as características desta abordagem, que vem sendo cada vez mais adotada pela indústria, juntamente com os problemas identificados relacionados a sua utilização e as soluções que vêm sendo sugeridas e adotadas para atacar esses problemas. Destacaram-se como soluções adotadas às limitações do teste exploratório ferramentas de apoio que também foram utilizadas e analisadas no contexto deste trabalho.

Palavras-chave: Testes Exploratórios, Ferramentas para Testes Exploratórios, Mapeamento Sistemático.

Exploratory Tests: Features, Problems and Solutions

ABSTRACT

Software testing is one of the most common activities used in the software development process in order to evaluate the correctness and quality of a software. These tests can be manually performed or automated, many testers conduct their tests based on scripts. The script describes step-by-step of what is to be followed by the tester, what kind of input should be provided to the system and which output is expected. When the tester doesn't follow a script, the approach adopted is called Exploratory Test, where the tester freely explores the system looking for bugs. This paper presents the results of a literature review whose goal was to provide a survey of the approach's characteristics, which has been increasingly adopted by the industry, along with the identified problems related to its use and the solutions that have been suggested and adopted to attack these problems. Exploratory Testing supporting tools which stood out as adopted solutions to exploratóry testing limitations were also used and analyzed in this work's context.

Keywords: Exploratory Testing, Exploratory Testing Tools, Systematic Mapping.

Lista de Figuras

Figura 1. Papel do teste de software nas etapas de verificação e validação……...……...……17
Figura 2. Modelo V……...…...21
Figura 3. Scripted/Exploratory Testing Continuum……...28
Figura 4. Teststuff - Tela inicial/ gerenciamento do Teststuff……...65
Figura 5. Teststuff - Editor de testes do Teststuff……..65
Figura 6. Teststuff - Teste criado no Teststuff……...…...66
Figura 7. Teststuff - Test run……..66
Figura 8. Teststuff - Resultdo de teste mostrado no navegador……...……..67
Figura 9. Teststuff - Resultado de teste mostrado no próprio aplicativo…….....67
Figura 10. Rapise - Antes da inserção do passo a passo…….…...68
Figura 11. Rapise - Após da inserção do passo a passo…….69
Figura 12. Rapise - Interface visível durante os testes……..69
Figura 13. Testpad - Missões a serem testadas pelo usuário……..70
Figura 14. Testpad - Passo a passo dos testes/Charter das missões……..…71
Figura 15. Testpad - Teste completo……71
Figura 16. Bug Magnet - Interface mostrando tipos de dados fornecidos pela ferramenta para a inserção……...……...……...72
Figura 17. Testrail - Tela inicial……....73
Figura 18. Testrail - Adicionar / Editar projeto de testes……..74
Figura 19. Testrail - Overview do projeto…….....74
Figura 20. Testrail - Adicionar caso de teste - template exploratory session……...75
Figura 21. Testrail - Caso de teste adicionado……...75
Figura 22. Testrail - Adicionar test run I.……...76
Figura 23. Testrail - Adicionar Adicionar test run II……...……..76
Figura 24. Testrail - Resultados dos testes após a execução dos testes……...……..77
Figura 25. Wink - Tela inicial de configuração de teste……78
Figura 26. Wink - Screenshots……..78
Figura 27. Wink - Report Textual……...79
Figura 28. Zephyr Capture for Jira - Interface durante os testes - Anotações…….....……80
Figura 29. Zephyr Capture for Jira - Interface durante o teste……...80
Figura 30. Zephyr Capture for Jira. Detalhes do bug relatado……...81
Figura 31. Zephyr Capture for Jira. Criar sessão……...……81

Figura 32. Zephyr Capture for Jira. 1- Detalhes da sessão criada…….82
Figura 33. Zephyr Capture for Jira 1 - Convidar usuário……...……...82
Figura 34. TestAssistant - Interface inicial……....83
Figura 35. TestAssistant - Expert Recorder……...……...…….84
Figura 36. TestAssistant - Expert Player……...……...…….....84
Figura 37. Rapid Reporter - Interface e Funções...……...……...…….…….85
Figura 38. Test Studio - Criação do projeto…….... ……...…...86
Figura 39. Test Studio - Interface durante os testes……...87
Figura 40. Test Studio - Após a gravação do passo a passo do teste……87
Figura 41. Test Studio - Storyboard……...88
Figura 42. Exploratory Testing Chrome Extension - Interface inicial…….....89
Figura 43. Exploratory Testing Chrome Extension - Reporte de bugs……..89
Figura 44. Exploratory Testing Chrome Extension - Detalhe da Interface durante/após a sessão…....90
Figura 45. Exploratory Testing Chrome Extension - Relatório de teste……...…….90

.

Lista de Tabelas

Tabela 1. Ferramentas classificadas em relação às funcionalidades……...……...……...49
Tabela 2. Ferramentas classificadas em relação à plataforma testada…….……....50
Tabela 3. Ferramentas classificadas em relação às versões disponíveis e versões testadas.…….…..…..51
Tabela 4 - Problemas associados à adoção do teste exploratório…….....55
Tabela 5 - Relação entre ferramentas e problemas solucionados……..55
Tabela 6 - Relação entre ferramentas e abordagens estudadas……..91

Sumário

1 Introdução **12**
1.1 Contexto e Motivação 12
1.2 Motivação 13
1.3 Objetivo 13
1.4 Metodologia 13

2 Fundamentação Teórica **16**
2.1 Teste de Software 16
2.2 Conceitos Básicos 16
2.3 Tipos de Testes 19
2.3.1 Testes Funcionais ou Caixa Preta 19
2.3.2 Testes Estruturais ou Caixa Branca 19
2.3.3 Teste de Carga 20
2.3.4 Teste de Performance 20
2.3.5 Teste de Recuperação de Falhas 20
2.3.6 Teste de Resistência 20
2.3.7 Teste de Interface com o Usuário/ Usabilidade 20
2.4 Níveis de Testes 21
2.4.1 Teste de Unidade 22
2.4.2 Teste de Integração 22
2.4.3 Teste de Sistemas 22
2.4.4 Teste de Aceitação 22
2.4.5 Teste de Regressão 23

3 Testes Exploratórios, Problemas, Soluções e Desafios **24**
3.1 Definição: Testes Exploratórios 24
3.2 Entendendo o Teste Exploratório 28
3.2.1 Onde Aplicar o Teste Exploratório 30
3.2.2 Características 31
3.2.3 O Teste Exploratório na Prática 31
3.3 Problemas e Desafios 33
3.4 Soluções 34
3.4.1 Session Based Test Management - SBTM 35
3.4.2 Exploratory Testing in Pairs 37

3.4.3 Team Exploratory Testing Sessions - TET 38
3.4.4. Tours. 43
3.4.5 Rule-Based Exploratory Testing - R-BET. 45

4 Ferramentas de Apoio ao Teste Exploratório 48
4.1 Análise de Ferramentas 48

5 Considerações Finais 56
5.1 Discussões e Lições Aprendidas 56
5.2 Limitações do Estudo 57
5.3 Trabalhos Futuros 57

Referências Bibliográficas 59

Apêndice A - Análise Aprofundada das Ferramentas 64
Testsuff 64
Rapise 68
TestRail 72
Wink 77
Jira + Capture for JIRA (Zephyr Capture for JIRA) 79
BB TestAssistant 83
Rapid Reporter 84
Test Studio Explorer 85
Exploratory Testing Chrome Extension 88

Apêndice B - Link Para as Ferramentas 92

1 Introdução

Teste de software é uma das atividades mais adotadas durante o processo de desenvolvimento de software para se garantir a corretude e qualidade de um software [41][27]. Estes testes podem ser manuais ou automatizados, muitos testadores realizam seus testes tendo como base um script que descreve o passo a passo do teste que deve ser seguido pelo testador, quais tipos de entradas devem ser fornecidas ao sistema e quais saídas são esperadas. Quando o testador não faz uso de um script, a abordagem adotada se chama Teste Exploratório, onde o testador explora o sistema livremente a procura de bugs [20].

Este trabalho apresenta o resultado de uma revisão da literatura cujo objetivo foi levantar as características do Teste Exploratório, que vem sendo cada vez mais adotado pela indústria, como também os problemas relacionados a sua utilização e as soluções que vêm sendo sugeridas e adotadas para atacar esses problemas.

1.1 Contexto e Motivação

Sistemas de software estão cada vez mais presentes e são importantes em todas as atividades humanas, facilitando e agilizando tanto atividades corriqueiras, tais como: sistemas de software presente em aparelhos domésticos, na indústria, no setor de comunicações, educação, transporte, sistemas de software utilizados em indústrias para movimentar e controlar suas linhas de produção, em instituições de ensino, na manutenção de atividades hospitalares, no controle de aeronaves, presentes em automóveis, em satélite, etc.. Chegamos ao ponto em que muitas destas atividades não seriam possíveis de serem realizadas sem a utilização de algum tipo se sistema de software e/ou sequer existiriam, tal como a indústria de jogos digitais.

Na maioria dessas áreas, potenciais bugs podem causar enormes prejuízos materiais, financeiros e em alguns casos pôr vidas em risco. Por essa razão o teste de software é uma etapa importante do processo de engenharia de software, para prevenir erros que possam afetar criticamente a execução de suas funções e a experiência do usuário [4].

Uma das técnicas de testes que vem sendo mais adotadas pela indústria é o teste exploratório. que consiste em explorar o sistema alvo sem seguir nenhum script ou caso de

testes com o objetivo de encontrar possíveis bugs, ou como definido por James Bach em [2]: "*Teste Exploratório é aprendizagem, projeto e execução de testes realizados de maneira simultânea*."

1.2 Motivação

Apesar de ser muito utilizado, o teste exploratório pode ser considerado por alguns testadores como uma prática onde os testes são realizados de maneira aleatória, o que não é verdade. Apesar de ter algumas limitações/desvantagens associadas, como por exemplo: a dependência das habilidades e experiência do testador humano, a dificuldade em se julgar formalmente a corretude de uma saída dado que o próprio testador age como oráculo, documentação pobre, dificuldade em determinar a cobertura de testes [24], pode ser aplicado de maneira estruturada e se mostra bastante eficiente em determinadas situações aqui discutidas

1.3 Objetivo

O objetivo deste trabalho consiste em investigar características, e problemas e soluções associados à utilização dos testes exploratórios, respondendo às seguintes perguntas:

i) O que é o teste exploratório e como caracterizá-lo?
ii) Como aplicar o teste exploratório?
iii) Quais os problemas associados à sua adoção?
iv) Como solucionar os problemas associados à abordagem?

1.4 Metodologia

Para atingir o objetivo deste trabalho, a princípio tentamos realizar um mapeamento sistemático utilizando as string de busca "*Exploratory Testing*", "*Exploratory Testing Tools*", "*Tools for Exploratory Testing*". Usamos como base de dados o Google, e o Google Scholar procurando por trabalhos em inglês que tivessem os termos pesquisados em seu título o que resultou em inúmeros resultados. Desses resultados descartamos os que não se relacionavam a área de Teste de Software e selecionamos os que achamos relevante à pesquisa considerando o título do trabalho, restando ainda em um grande número de resultados o que

tornou o mapeamento impraticável. Nesta primeira fase, através da palavra chave "*Exploratory Testing"* se chegou a 240 trabalhos contendo o termo pesquisado em seu título, a palavra chave "*Exploratory Testing Tools"* produziu 5 resultados e a palavra chave "*Tools for Exploratory Testing*", mais 4 resultados.

Decidimos então utilizar outra estratégia e optamos por fazermos uma revisão bibliográfica. Para esta revisão bibliográfica foi selecionado um conjunto inicial de referências dentre as já selecionadas durante a primeira fase da pesquisa, dando preferência a livros, artigos que tivessem como fonte as bases IEEE Xplore Digital Library e ACM Digital Library e tutoriais, mas não se limitando a eles.

Realizamos, então, a leitura do abstract, quando presente, a fim de identificar a relevância dos trabalhos e quando ele se mostrasse promissor, checamos seu conteúdo, escolhendo os que respondiam a maior parte dos questionamentos levantados e descartamos trabalhos que não tratavam do assunto de maneira aprofundada. Devido a grande quantidade de trabalhos restantes decidimos aplicar a técnica de *snowballing sampling,* tendo como base os primeiros artigos lidos, com o objetivo de identificar trabalhos considerados indispensáveis à pesquisa, como os artigos dos idealizadores da abordagem.

Utilizando a técnica de *snowballing sampling* [37], onde a partir dos trabalhos identificados inicialmente chega-se a trabalhos relacionados checando-se os trabalhos referenciados neste primeiro grupo e assim sucessivamente, chegou-se ao conjunto de referências utilizadas neste estudo. Utilizando esta técnica chegou-se aos artigos escritos pelos idealizadores dos testes exploratórios, Jonathan Bach, seu irmão James Bach e Cem Kaner, referenciados por vários dos demais trabalhos pesquisados. Enquanto que os conceitos básicos contidos neste trabalho foram encontrados a partir da pesquisa do termo em inglês nas bases já citadas.

Durante a pesquisa dois tipos de soluções foram encontrados: melhorias na abordagem e um conjunto de ferramentas que podem ser utilizadas em apoio ao teste exploratório que por essa razão são analisadas neste trabalho, elas foram citadas nos artigos consultados para este trabalho, encontradas a partir da *string* de busca "*Exploratory Testing tools*", com exceção de uma delas (Exploratory Testing Chrome Extension) que foi descoberta durante a fase de análise das demais, tendo sido sugerida pelo próprio browser durante a instalação de outra extensão analisada neste trabalho.

Fez parte da metodologia deste trabalho instalar e usar a ferramenta para assim podermos escrever sobre características da mesma. Análise sobre suas funcionalidades foi realizada utilizando uma abordagem puramente exploratória como meio de, também, colocar em prática a abordagem estudada e descrita neste trabalho. Procurou-se utilizar as principais, senão todas, as funcionalidades disponíveis e pertinentes a atividade de testes mesmo quando a ferramenta (2 delas), também apresenta funções relacionadas a gerenciamento de testes. A análise é feita sob o ponto de vista de um "testador" iniciante, que utiliza as ferramentas pela primeira vez. Para algumas das análises, foi utilizado algum tipo de tutorial disponibilizado pelo site do fabricante sendo a maioria por meio de vídeos de demonstração da ferramenta sendo utilizada.

É importante salientar que o conjunto de ferramentas aqui apresentado é apenas uma amostra de ferramentas que apoiam o teste exploratório e que existem diversas outras que podem ser utilizadas para este fim.

1.5 Organização do documento

Este trabalho possui 6 capítulos, organizados da seguinte forma: No Capítulo 1, apresentamos o objetivo e a metodologia, no Capítulo 2 citamos uma breve fundamentação teórica sobre o teste de software, relembramos alguns conceitos básicos, tipos de testes e níveis de testes. No Capítulo 3, apresentamos a abordagem estudada, isto é, o Teste Exploratório, o conceito de *scripted/exploratory testing continuum*, como o teste exploratório funciona na prática, bem como seus problemas , desafios e soluções; O Capítulo 4 contém a análise das ferramentas de apoio ao teste exploratório e às soluções discutidas no capítulo anterior. No Capítulo 5 apresentamos as lições aprendidas, limitações deste estudo, as considerações finais. No apêndice A apresentamos as ferramentas mais detalhadamente.

2 Fundamentação Teórica

Para melhor entendimento do assunto tratado neste neste trabalho devemos revisar alguns conceitos para que não haja lacunas e prejuízos posteriores em nosso estudo e para fortalecer o entendimento dos conceitos previamente aprendidos. Neste capítulo relembramos o que é Teste de Software, alguns conceitos básicos, alguns tipos de testes e os níveis representados pelo modelo V.

2.1 Teste de Software

Parte importante no processo de engenharia de software, o teste de software tenta minimizar a ocorrência de erros nos sistemas de software identificando os riscos e problemas do produto que possam ameaçar a sua qualidade antes que o produto chegue ao cliente final.

Quando testamos um programa temos a intenção de agregar valor ao produto testado, aumentando sua qualidade e confiabilidade, para que isso ocorra testamos o produto não para provar que ele funciona, mas partindo do pressuposto de que os erros existem e que através do processo de testes a maior quantidade possível de erros serão encontrados.

Teste de software é a atividade de investigar um sistema a procura prováveis falhas no seu funcionamento, a fim de garantir que as necessidades e expectativas do cliente sejam cumpridas e a qualidade do software sendo desenvolvido, ou seja, é verificar se o programa cumpre com seus requisitos, que realmente realiza aquilo que se espera dele, da maneira esperada e que não apresenta nenhum comportamento não especificado.

Considerando-se que o custo de detectar os erros e corrigi-los nos estágios finais do desenvolvimento é bastante alto, correndo o risco de inviabilizar um projeto se esse custo for muito alto, não valer a pena, ou se não existir verba e/ou tempo necessários, se faz necessário que os testes de softwares sejam executados nos estágios iniciais do processo, para que os defeitos sejam corrigidos e se evite a sua propagação.

2.2 Conceitos Básicos

Nesta seção apresentamos conceitos bastante utilizados e outros que são comumente confundidos e em alguns utilizados erroneamente como sinônimos.

- **Erro x Falta x Falha**

Um programador comete um *erro* durante o fase de desenvolvimento, esse erro resulta em uma *falta* ou defeito de codificação [1]. *Falta* ou defeito pode ser definido como aquilo que altere o funcionamento normal de um sistema, uma falha também pode ser causada por fatores externos ao sistema e portanto fora do controle dos desenvolvedores, como por exemplo uma queda de energia, uma queda de rede, etc. A *falta* por sua vez se manifesta para o usuário da forma de *falha* no funcionamento, que ocorre quando o sistema não se comporta como o esperado produzindo resultados diferentes do padrão especificado,ou seja, a falha é o comportamento inesperado observado pelo usuário.

- **Verificação e Validação**

Embora sejam confundidas com frequência, não são a mesma coisa. *Verificação* é o processo de checar que o software atende a sua especificação, geralmente envolve a checagem de documentos de requisitos e arquivos, enquanto que v*alidação* é o processo de checar se as especificações capturam as necessidades do consumidor, envolve a execução do código, a utilização do produto. A verificação ajuda a determinar a qualidade do produto, a validação determina sua utilidade. De maneira mais simples e direta, elas respondem as perguntas:

Validação: Estamos construindo o sistema certo?

Verificação: Estamos construindo o sistema de maneira correta?

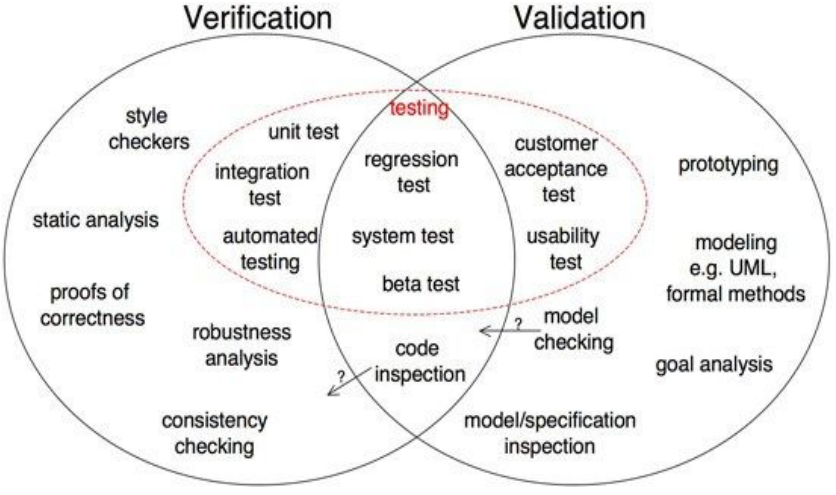

Figura 1:Papel do teste de software nas etapas de verificação e validação. Extraída de [9].

- **Casos de Teste X Cenário de Testes**

Casos de teste são um conjunto de variáveis, pré-condições, resultados esperados e pós-condições desenvolvidas para um cenário de testes a fim de verificar a conformidade do produto desenvolvido com os requisitos especificados. Esses dados especificados para cada caso de teste e aplicados durante o teste determinam se a aplicação se comporta da maneira pretendida. Formado por vários casos de testes, os *cenário de teste* consiste de um procedimento detalhado de testes

- **Cobertura**

É definido como uma medida que determina a completude de um conjunto de casos de testes. Mede quanto do código fonte de um programa foi executado quando um determinado conjunto de testes é executado, buscando sempre uma grande cobertura. Uma cobertura baixa significa que alguns caminhos lógicos não foram testados. Uma alta cobertura indica que a probabilidade de processamento correto é boa mas não implica, necessariamente, que todas as combinações de dados de entrada sejam processadas corretamente pelo código.

- **Performance x Escalabilidade**

De acordo com [17], performance, ou desempenho, mede a rapidez e eficiência com que um sistema de software realiza uma tarefas enquanto que escalabilidade mede a tendência do desempenho à medida que a carga cresce. Os dois conceitos são inseparáveis, não fazendo sentido falar em falar sobre escalabilidade se o sistema não desempenhar bem suas funções, no entanto um sistema pode ter um bom desempenho sem ser escalável.

- **Oráculo de testes**

O termo oráculo se refere ao método pelo qual se avalia a corretude da resposta/execução de um sistema. Contém duas partes essenciais: a saída correta esperada e um procedimento para a comparação entre a saída obtida e a esperada [22]. A existência de algum tipo de oráculo é de extrema importância pois a capacidade de reconhecer uma saída inesperada é crucial para qualquer tipo de teste.

- **Missão**

De acordo com [26], missão é uma declaração guia que descreve brevemente o que o testador deve testar, consiste de poucas linhas e deve ser geral o suficiente que permita o testador usar sua criatividade e conhecimento mas específica para evitar que o testador perca o foco durante o teste.

- **Testes Automatizados**

De acordo com [40]: "O gerenciamento e desempenho das atividades de teste, incluindo o desenvolvimento e execução dos scripts de testes, a fim de verificar os requisitos de testes, usando uma ferramenta de testes automatizada."

2.3 Tipos de Testes

2.3.1 Testes Funcionais ou Caixa Preta

Testes funcionais não se preocupam com o funcionamento interno do sistema, logo, os testadores não devem ter acesso ao código fonte do sistema, importando-se apenas com a saída obtida mediante determinada entrada resultado de sua execução [27]. O código é considerado uma grande caixa preta a qual o testador não deve conhecer seu conteúdo, apenas quais dados devem ser fornecidos ao sistema e quais resultados são esperados. Dessa forma os testes podem ser projetados baseados na especificação de requisitos.

2.3.2 Testes Estruturais ou Caixa Branca

Os testes estruturais se preocupam com o funcionamento interno do sistema e são escritos com base em informações extraídas do código fonte, na estrutura interna da implementação. O testador conhece o código, e muitas vezes é o próprio desenvolvedor do sistema, ele projeta testes e os executa fornecendo entradas pré-determinadas. Tais entradas têm como objetivo testar determinados caminhos lógicos através do código e produzir as saídas esperadas, levando em consideração o fluxo de controle e de dados do sistema [27].

2.3.3 Teste de Carga

O teste de carga avalia como o sistema se comporta dada uma grande demanda. Geralmente realizado após todas as funcionalidades serem completamente testadas [16], o teste de carga modela o uso esperado do sistema em desenvolvimento, e submete-o a simulações em que o sistema é utilizado por vários usuários simultaneamente, ou por grande volume de dados, testando assim a sua capacidade de atender a alta demanda, determinando se os recursos alocados são adequados para seu correto funcionamento e quando eles se tornam insuficientes.

2.3.4 Teste de Performance

Acontece em paralelo ao teste de carga. ele testa o tempo de resposta de uma funcionalidade aplicando incrementos na carga: cargas pequenas verificam se o sistema funciona como deveria em um nível mais básico, cargas médias verificam se ele funciona em um nível normal esperado, cargas altas verificam como ele reage a altas demandas, dessa forma identificando quando essa demanda pára de ser atendida e os motivos por trás disso.

2.3.5 Teste de Recuperação de Falhas

Teste de Recuperação de Falhas é uma técnica utilizada para verificar a capacidade de recuperação de um sistema após a ocorrência de uma falha, onde o sistema é submetido a uma falha que pode ser tanto de software ou hardware e sua capacidade e rapidez de recuperação é observada.

2.3.6 Teste de Resistência

Testa o funcionamento do sistema sob uso contínuo, a fim de determinar se o tempo de resposta do sistema permanece o mesmo ou melhora após um longo período e determinar o número de usuários e / ou transações, um determinado sistema suportará e atingirá as metas de desempenho.

2.3.7 Teste de Interface com o Usuário/ Usabilidade

Teste de Usabilidade é uma técnica utilizada para avaliar quão fácil de usar é um sistema, onde usuários reais são convidados a utilizar o sistema e executar determinadas

tarefas enquanto são observados por integrantes da equipe de desenvolvimento. O foco dessa técnica é medir quão fácil de usar e intuitivo é o produto, e a partir dos resultados coletados expor problemas de design que possam ser remediados pela equipe de desenvolvimento.

2.4 Níveis de Testes

O processo de testes acontece durante todo o desenvolvimento do software, pode ser classificado em diferentes níveis de testes, criando um certo paralelismo com os estágios do desenvolvimento do modelo cascata, podendo envolver testes de partes do sistema ou do sistema como um todo.

No modelo V os níveis de testes, com exceção do teste de regressão que ocorre cada vez que o sistema é modificado, são organizados em uma estrutura hierárquica. Cada uma das fases do desenvolvimento produz como um de seus artefatos os planos de teste para cada nível correspondente do processo de testes, aplicados posteriormente seguindo uma estratégia bottom up, onde os níveis superiores assumem que os níveis inferiores foram realizados com sucesso e completude.

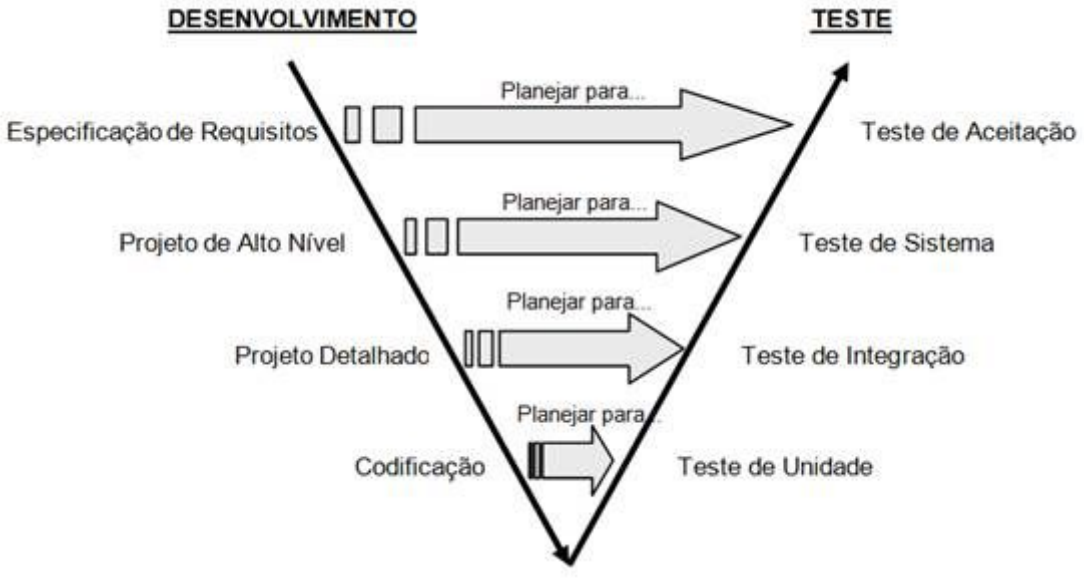

Figura 2 Modelo V . Extraída de [23]

2.4.1 Teste de Unidade

É realizado no menor componente independente e testável do código fonte, para verificar se o componente funciona da maneira correta cumprindo os requisitos. É a primeira etapa do processo e é geralmente realizada pelos desenvolvedores, sendo raramente feita por testadores. Dessa forma o número de casos de teste e os dados do teste são menores, nem sempre sendo possível verificar todos os cenários para o fluxo funcional e de informações do software. Portanto, existem muitos casos de teste que podem só ser testados após o *merge* com outras unidades para formar um componente maior.

Como são realizados muito cedo no processo de desenvolvimento, possuem a vantagem de minimizar os riscos, o tempo e dinheiro gastos, caso ocorra atraso em encontrar essas falhas e seja necessário repará-las mais tarde quando os sistema estiver quase completo.

2.4.2 Teste de Integração

Visa descobrir erros associados às interfaces entre os módulos. Os testadores verificam como os componentes interagem entre si e como produzem saídas para diversos cenários. Vários defeitos relacionados aos níveis funcionais, requisitos e desempenho são descobertos neste tipo de teste.

2.4.3 Teste de Sistemas

Uma vez que todos os módulos estão integrados e passaram pelo teste de integração, o sistema como um todo é testado pela primeira vez, permitindo a verificação e validação dos requisitos de negócio e da arquitetura da aplicação.

Os testes são conduzidos em um ambiente semelhante ao que realmente será utilizado, por uma equipe de testes especializada, não apenas em relação ao desempenho, mas também quanto a adesão aos padrões de qualidade conforme as especificações funcionais e técnicas e os padrões de qualidade definidos pela organização.

2.4.4 Teste de Aceitação

O teste de aceitação representa os interesses do cliente, dá ao cliente a confiança de que o sistema possui as características e a qualidade almejada, que ele se comporta corretamente [13] e satisfaz suas necessidade, validando o produto final. Vale ressaltar a

importância do processo de validação do sistema realizado durante o desenvolvimento para assegurar que o sistema especificado e construído pelos desenvolvedores é o mesmo que o cliente solicitou.

2.4.5 Teste de Regressão

Realizado sempre que alguma mudança/atualização é feita no software para garantir que o mesmo continua funcionando corretamente e que as mudanças não afetaram negativamente o seu funcionamento. Seu propósito é assegurar que novos defeitos não foram introduzidos na tentativa de consertar erros encontrados durante testes anteriores ou durante uma atualização de software, introdução de novas funcionalidades ou novas versões.

3 Testes Exploratórios, Problemas, Soluções e Desafios

Neste capítulo apresentamos o resultado da revisão da literatura realizado neste trabalho. Inicialmente, apresentamos a definição de testes exploratórios e como essa definição vem sendo desenvolvida e aprimorada ao longo dos anos. Apresentamos a motivação para sua criação e o desenvolvimento do conceito de *scripted/exploratory testing continuum* até chegar ao conceito adotado atualmente.

Em seguida, na revisão de literatura que realizamos procuramos investigar como os testes exploratórios são utilizados na prática, isto é, o que influencia nos resultados alcançados pela abordagem, os problemas relacionados a sua adoção e algumas das soluções sugeridas pelos teóricos e praticantes.

3.1 Definição: Testes Exploratórios

Teste exploratório é uma das formas mais intuitivas de se testar um sistema e por essa razão tem sido usado por testadores e desenvolvedores, seja de forma consciente ou não, quando se explora um sistema com o intuito de aprender sobre ele, ou de encontrar erros e defeitos em suas funcionalidades.

Em 1988, Cem Kaner, em seu livro "*Testing Computer Software"*[1], descreveu uma maneira de continuar testando o software, após a execução dos testes planejados e executados evitando, assim, o processo de projetar e documentar novos casos de testes enquanto o sistema ainda se encontra instável.

> *"No matter how many test cases of how many types you've created, you will run out of formally planned tests. At some later point, you'll stop formally planning and documenting new tests until the next test cycle. You can keep testing. Run new tests as you think of them, without spending much time preparing or explaining the tests. Trust your instincts. Try any test that feels promising, even if it's similar to others that have already been run." [1]*

Para James Bach, em seu artigo *Exploratory Testing Explained* [2], *"Teste Exploratório é aprendizagem, projeto e execução de testes realizados de maneira simultânea.*" , " *...é também conhecido como testes ad-hoc*." , "*Qualquer teste em que o testador ativamente controle o projeto dos testes enquanto eles são executados e utilize informação/conhecimento adquirido durante os testes para projetar testes novos e*

melhores", conceito também adotado por Tinkham e Kaner em seu artigo intitulado "*Exploring Exploratory Testing [2]*".

Bach também explica, no mesmo artigo, que o termo Exploratório foi proposto no início de 1990 por um grupo de metodologista em testes, que se auto denominava *Context-Driven Schoo*l, e foi primeiramente publicado por Cem Kaner em seu livro *Testing Computer Software [1]*. O termo foi proposto com o objetivo de quebrar essa suposição errada de testes superficiais e enfatizar que o processo é dominantemente caracterizado pela habilidade de aprendizagem e de tomada de decisões do testador.

Ao longo dos anos, o conceito de testes exploratórios foi evoluindo como mostra a cronologia abaixo, retirada de *History of Definitions of ET [8]*, James Bach:

1988	O primeiro uso conhecido do termo, definido de várias formas como "testes rápidos"; "o que vier a mente"; ataques de guerrilha - Cem Kaner *Testing Computer Software* [8], onde existe um texto explicativo sobre diferentes estilos de ET na edição deste mesmo ano, mas que o autor afirma que alguns trechos foram escritos ainda em 1983.
1990	*"Organic Quality Assurance", James Bach dá sua primeira palestra sobre testes ágeis filmada por Apple Computer, que discutiu testes exploratórios sem utilizar as palavras exploratório e ágeis.*
1993	*Junho: "Persistence of Ad Hoc Testing" palestra dada na conferência da ICST por James Bach onde ele inicia uma tentativa mal sucedida de reabilitar o termo "ad hoc".*
1995	*Fevereiro: Primeira aparição do termo "teste exploratório" no Usenet (sistema de discussão) em uma mensagem postada por Cem Kaner.*
1995	Teste Exploratório significa aprendizagem, planejamento de teste e execução dos testes, todos ao mesmo tempo. – James Bach *(Market Driven Software Testing class).*

1996	Exploração, planejamento e execução de testes de maneira simultânea James Bach *(Exploratory Testing class v1.0).*
1999	Processo interativo de exploração de produto, design de teste e execução de testes ocorrendo de forma concorrente. – James Bach *(Exploratory Testing class v2.0).*
2001(após WHET #1)	*O ponto de vista de Bach* Qualquer teste em que o testador controla ativamente o design dos testes enquanto estão sendo executados e usa informação obtida durante estes testes para projetar novos e melhores testes. *O ponto de vista de Kaner* Qualquer teste em que o testador controla ativamente o design dos testes enquanto estão sendo executados e usa informação obtida durante estes testes para projetar novos e melhores testes e onde as seguintes condições se aplicam: ● Não é exigido que o testador utilize ou siga nenhum procedimento ou material de testes em particular; ● Não é exigido que o testador produza nenhum material ou procedimento que permita o reuso do teste por outro testador ou uma revisão administrativa do trabalho realizado. – Resolução entre Bach e Kaner após WHET #1 e aula sobre BBST no Satisfice Tech Center. (Considerando os dois pontos de vista, Bach passou a utilizar o conceito do "continuum testes baseados em script/ testes exploratório" que auxiliou a explicar ET à testadores que utilizavam testes estilo de fábrica).
2003-2006	Aprendizagem, projeto e execução de testes realizados de maneira simultânea – Bach, Kaner.
2006-2015	Uma abordagem de testes de software que enfatiza a liberdade e responsabilidade pessoais de cada testador em otimizar constantemente a qualidade de seu trabalho tratando o aprendizado, o design e execução de

testes como atividades que acontecem paralelamente através do projeto. – (Bach/Bolton edição sugerida por Kaner).

2015 Teste exploratório agora é um termo obsoleto na metodologia do Rapid Software Testing. (Em outras palavras, todos os testes são considerados exploratórios em algum nível. A definição de teste no RST é agora: Avaliar um produto aprendendo sobre ele através de exploração e experimentação, incluindo em algum grau: questionamento, estudo, modelagem, observação, inferência, etc).

Toda essa discussão acerca da definição de teste exploratório levou ao desenvolvimento do conceito de *scripted/exploratory testing continuum*. Os teóricos e praticantes, após anos discutindo sobre o tema, chegaram a um consenso em 2015 [8], [30] de que todo teste é exploratório, o que restava definir era o quão exploratório ele seria, no sentido de que toda atividade de testes se encaixa no *scripted/exploratory testing continuum* que varia entre os dois extremos: do teste baseado em script, em que o testador faz exatamente o que foi especificado pelo script e nada mais, ao puramente exploratório ou *Free Style,* em que não existe nenhum atividade pré-planejada para ser executada pelo testador [3].

Kaner e Tinkham afirmam em [3], que até testes predominantemente baseados em script podem ser exploratórios se realizados por um testador habilidoso, pois um bom testador será capaz de reconhecer indícios de que existe algo errado com o software e neste caso ele deverá abandonar o script e investigar os indícios encontrados. James bach afirma em [2] que a maioria das situações se beneficiam de uma mistura das duas abordagens, os teste baseado em script e o teste exploratório.

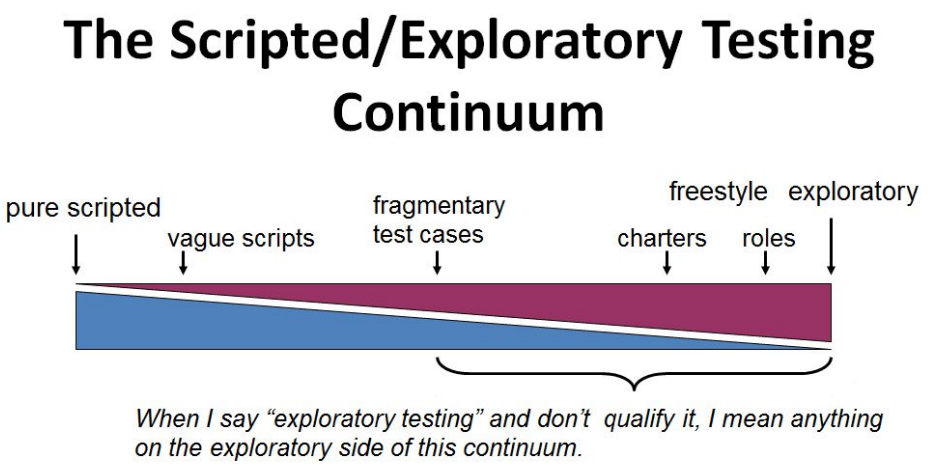

Figura 3 - Scripted/Exploratory Testing Continuum. Extraída de[31]

3.2 Entendendo o Teste Exploratório

Os testes exploratórios são baseados nas habilidades, conhecimento e capacidade analítica dos testadores em produzir e executar bons testes, o que não significa que estes testes sejam aleatórios ou realizados sem um planejamento prévio. Quando o testador adapta-se ao sistema sendo testado, cria e melhora os testes baseando-se no conhecimento adquirido durante a exploração do sistema, sem o auxílio de instruções, a abordagem exploratória pode ser tão produtiva quanto testes mais estruturados como por exemplo os testes baseados em scripts.

James Bach afirma, em seu artigo *Exploratory Testing Explained* [2], que daí vem o poder dos testes exploratórios, dessa capacidade de se usar o conhecimento adquirido durante os testes para construí-los e aperfeiçoá-los, e eventualmente criar novos testes, mas que pode perder sua força se esse feedback entre aprendizagem e design de testes for fraco, longo, lento ou caro demais. Esse poder pode ser otimizado durante o processo, enquanto que os testes baseados em script tendem a perder a eficiência com o passar do tempo, devido ao fato de que ele permanece imutável, após sua primeira execução que não encontrar nenhum defeito a probabilidade de encontrá-los em execuções seguintes diminui. o que não acontece com os testes exploratórios.

"ET is powerful because of how the information flows backward from executing testing to redesigning them. Whenever that feedback loop is weak, or when the loop is particularly long, slow, or expensive, ET loses that power". [2]

Faz parte desse conhecimento: informação sobre o mercado, sobre o produto testado e possíveis versões anteriores, produtos relacionados, plataformas de software e hardware onde o sistema será utilizado, riscos, falhas e comportamento do produto, podendo ser classificado em conhecimento sobre o domínio, conhecimento sobre o sistema, conhecimento sobre engenharia de software em geral.

Outros tipos de testes utilizam algum tipo de documentação, como por exemplo os casos de testes, o documento de especificação de requisitos, como oráculo, ou seja, como saber qual comportamento se espera do sistema, o que seria um bug, o que seria um comportamento correto ou um inesperado. ET utiliza o conhecimento e habilidades, prévias ou adquiridas, do testador como oráculo. Essa abordagem difere claramente do paradigma tradicional, o uso de casos de testes, no qual o resultado é especificado antes da execução do teste.

Apesar de ET ser uma abordagem de testes bastante poderosa é muitas vezes mal interpretada. Muitos testadores que não estão familiarizados com a prática acreditam se tratar de uma técnica, o que não chega a ser verdade, pois não possui passos bem definidos e depende bastante do tipo de sistema, da estratégia adotada, do testador, etc. Essa, aliás, é sua principal diferença dos outros tipos de testes: não segue scripts e não possui um plano de testes bem detalhado, mas como as abordagens mais praticadas, possui uma missão como seu objetivo.

Segundo [5][6] o teste exploratório é uma abordagem que pode ser aplicada a diferentes tipos de testes usando várias técnicas e métodos, mas que se adequa particularmente bem aos testes funcionais [7]. A abordagem é defendida pelos praticantes como uma parte valiosa do conjunto de práticas empregadas para garantir a qualidade do software e que ela não existe para substituir métodos mais estruturados, como as abordagens que se baseiam em casos de testes, mas para complementá-las em certas situações adequadas, onde outros métodos esgotaram as chances de encontrar novos *bugs* [2].

Pode-se utilizar o teste exploratório em inúmeras situações, como por exemplo quando o sistema em desenvolvimento pode ser utilizado de várias maneiras, com inúmeras combinações de suas funções tornando a criação de casos de testes ou scripts para cada uma delas é bastante difícil ou impossível.

3.2.1 Onde Aplicar o Teste Exploratório

Outras situações as quais o teste exploratório se aplicam citadas nos trabalhos pesquisados são:

- Quando um feedback em curto prazo ou uma rápida aquisição de conhecimento sobre o sistema são necessários [4][5][2];
- Quando o próximo teste a se executar não é óbvio ou quando se deseja ir além dos testes mais comuns [2];
- Quando não existe tempo suficiente para o desenvolvimento de casos de testes ou para aplicar outra técnica mais sistemática mas que é necessário realizar testes no sistema ou em partes dele [5];
- Realizar testes de regressão ou smoke testing em sistemas que sofreram algum tipo de atualização ou para verificar a persistência de algum defeito reportado previamente [5][2][37] ;
- Testes realizados do ponto de vista do usuário final, baseados no manual do produto[5];
- Oferecer uma maior diversidade aos testes de sistemas que já passaram por testes baseados em script [5][2], baseados em casos de testes pré definidos [4], ou simplesmente melhorar os testes já existentes [2];
- Quando é necessário interpretar instruções de testes muito vagas ou imprecisas [2][4];
- Quando não é possível escolher o próximo caso de testes, mas que deve ser baseado nos testes realizados previamente [5] ou quando se deseja improvisar baseando-se em scripts de testes já existentes [2];
- Em casos em que é necessário explorar o tamanho, abrangência e variações de um defeito já conhecido e fornecer feedback aos desenvolvedores [5];
- Quando nenhum erro é mais encontrado pelos testes baseados em script, o que não significa que eles não existam [5];
- Quando a documentação de requisitos não está disponível, pois auxilia o testador a entender o sistema ao invés de tentar criar testes baseados em script sem a presença de documentação [7];

- Quando se é necessário aprender sobre o software e utilizar esse conhecimento para outros fins, como na escrita de um manual [2];
- Quando se deseja encontrar o defeito mais importante num curto período de tempo ou se deseja investigar e isolar um defeito específico [2][4].

3.2.2 Características

A partir das definições propostas ao longo dos anos, os teóricos e praticantes do teste exploratório apontam como principais características que os diferenciam dos demais tipos:

- Os testes exploratórios são aprendizagem sobre o sistema sob teste, design e execução acontecendo simultaneamente [2];
- Diferentemente dos testes baseados em scripts, os testes exploratórios não seguem casos de testes ou planos detalhados pré-definidos, ao invés disso, exploram com o objetivo de cumprir uma missão geral, um objetivo [4];
- O principal foco está em detectar erros através da exploração do sistema ao invés de produzir casos de testes para serem utilizados posteriormente, logo considera-se que reportar os defeitos encontrados é mais importante do que planejar os scripts de teste passo a passo [4][2];
- São guiados pelos resultados de testes realizados previamente e pelo conhecimento adquirido durante sua realização [4];
- A eficácia dos testes depende do conhecimento, experiência e das habilidades do testador, ou seja, diferentes testadores obterão diferentes resultados condizentes com suas habilidades, por isso o testador é valorizado como parte integral ao processo [4][5] .

3.2.3 O Teste Exploratório na Prática

O Teste Exploratório é considerado uma prática profundamente situacional, onde vários tipos de detalhes afetam seu desempenho. James Bach [2], cita alguns desses detalhes:

- A missão do projeto de teste e da sessão de teste em particular;
- O papel do testador, seus talentos, habilidades e preferências;

- Disponibilidade de tempo, dados de testes, materiais, ferramentas e instalações adequadas, bem como de ajuda de outros testadores/desenvolvedores;
- Requisitos de prestação de contas;
- Com o que os clientes se interessam, se importam sobre o projeto;
- A estratégia atual de testes;
- O status de outros esforços de testes realizados anteriormente no mesmo produto;
- O produto em si: comportamento, interface com o usuário, estado atual de execução, seus defeitos, testabilidade, propósito…;
- O que os testadores conhecem sobre o produto: problemas passados, problemas conhecidos, o que aconteceu em testes anteriores, pontos fortes e pontos fracos, mudanças recentes, áreas de risco e magnitude de tais riscos, observações diretas do produto, rumores sobre o mesmo, a natureza de seus usuários e seu comportamento, como o produto deve funcionar, como foi montado/desenvolvido, quais suas semelhanças e diferenças em relação a outros produtos;
- O que os testadores gostariam de saber sobre o produto.

A estrutura elementar do teste exploratório pode ser descrita da seguinte maneira: O testador interage com o produto a procura de defeitos, a fim de verificar seu correto funcionamento e se as expectativas do cliente são atendidas, e reporta os resultados encontrados. Nesta descrição podemos encontrar os elementos básicos do teste exploratório: produto, testador, testes e resultados.

Durante os testes, o testador produz questões sobre o produto e tenta respondê-las de maneira satisfatória, essas questões são respondidas através de testes criados e executados pelo testador. Se essa resposta não é considerada satisfatória os testes são ajustados ou outros testes adicionais são produzidos, ou seja, o testador continua explorando o sistema e ao fim da exploração os resultados são reportados.

No teste exploratório *free style,* o único resultado oficial produzido é um relatório dos bugs encontrados. Outras modalidades da prática do teste exploratório produz outros artefatos além desse relatório como anotações feitas durante os testes que podem resultar em materiais atualizados ou novos dados sobre os testes.

As vantagens e os benefícios atribuídos a abordagem são vários, dentre os quais podemos citar::

- Aumenta a eficiência da detecção dos defeitos, considerando-se o número de defeitos, níveis de gravidade de defeitos encontrados e número de relatórios de falsos positivos[4][5];
- O tempo gasto e os custos associados ao desenvolvimento e manutenção dos casos de testes e scripts, documentação, etc, não são aplicados aos testes exploratórios;
- Como não necessita de requisitos detalhados, documentos de especificação ou casos de testes pré definidos, minimiza a documentação durante a preparação. Isto é uma vantagem quando os requisitos e o design do sistema mudam rapidamente, ou nos estágios iniciais do desenvolvimento o sistema ainda não foi totalmente implementado e ainda pode sofrer várias alterações [4] [5];
- Melhora as habilidades do testador, através do conhecimento e experiência adquiridos durante os testes, visto que o testador aprende sobre o sistema, sobre seu comportamento e suas falhas o que ajuda ao testador a criar testes melhores e mais poderosos. [4], [5];
- A capacidade de serem, facilmente, realizados mesmo sem a presença da documentação de especificação ou de requisitos, dada a experiência e conhecimento adquiridos pelos testadores durante os testes [5];
- A liberdade dada aos testadores possibilita a descoberta de erros que não seriam encontrados através de teste baseados em scripts permitindo que determinadas áreas do software, que possam ter sido negligenciadas pelos casos testes baseados nos requisitos, sejam testadas [30];
- Rápido fluxo de feedback entre desenvolvedores e testadores, que é atribuído à rápida capacidade de reação a mudanças no produto demonstrada pelos testadores[5].

3.3 Problemas e Desafios

Com a adoção cada vez mais frequente dos testes exploratórios percebeu-se a existência dos problemas a eles associados. Mesmo sendo economicamente viável, a abordagem possui alguns pontos fracos que podem causar um aumento na quantidade de trabalho que necessita ser refeito e afeta o custo total do sistema [24];

- Por depender fortemente da experiência e habilidades do testador, em alguns casos existe uma dificuldade de encontrar testadores com o conhecimento de domínio e as qualidades necessárias [24][5];
- Como o próprio conhecimento do testador substitui um oráculo formal, torna-se impossível julgar formalmente a corretude das respostas dadas pelo sistema. Dessa forma se uma saída for interpretada de maneira incorreta pode acarretar em defeitos residuais que podem permanecer no sistema ou serem , eventualmente detectados em futuros testes [24];
- A ausência de planos de testes pode resultar vários problemas como por exemplo: uma mesma funcionalidade ser testada mais de uma vez enquanto outras não serem testadas; uma funcionalidade importante pode deixar de ser testada ou um erro grave pode passar sem ser detectado pela fase de testes; dificuldade em garantir uma boa cobertura [24];
- Dificuldade de acompanhar o progresso dos testadores [29], produzir boas métricas para avaliar a qualidade e cobertura dos testes executados;
- A falta de uma definição de casos de testes torna difícil a reprodução dos testes realizados caso seja necessário, como nos testes de regressão [24];
- Documentação limitada acarreta diversos problemas durante a fase de testes: Como não existe um guia ou plano de teste e nenhum outro artefato é produzido a não ser o relatório de falhas é difícil saber o que foi ou não testado, os testes dificilmente podem ser reproduzidos o que torna os testes de regressão e a manutenção do sistema uma tarefa árdua para outros testadores [5];
- A abordagem não é indicada à sistemas de tempo real [24];
- O tempo e recursos gastos durante as execuções manuais dos testes poderia ser utilizado de outra maneira pela equipe, principalmente se a empresa não possui uma equipe de testes e estes também sejam realizados pela equipe de desenvolvimento [24].

3.4 Soluções

Várias soluções para os problemas apresentados pelos Testes Exploratórios foram propostas e experimentadas. A seguir estão descritas algumas delas, mais especificamente o

método proposto por James Bach e Jon Bach, Session Based Test Management - SBTM. O uso de ferramentas para auxiliar nos testes, e o uso de outras metodologias associadas ao Teste Exploratório com o objetivo de reduzirem alguns desses problemas.

Neste trabalho são descritos:

- A exploração dos sistemas em pares, que utiliza o conceito de programação em pares e o associa ao de SBTM;
- Exploração do sistema em equipe, que aplica o conceito de SBTM praticado em grupo;
- O conceito de tours como estratégias para explorar o sistema;
- Um método de automação de testes de software combinado com testes exploratórios, o de testes baseados em regras.

A inclusão desta última abordagem se deve ao fato de que o teste exploratório não foi totalmente automatizado. Apenas alguns aspectos da automação são utilizados como por exemplo a capacidade de repetição do testes, uma vez que a porção exploratória ainda seja realizada por um testador humano, e ainda depende de suas habilidades, como é o caso do teste baseado em regras.

3.4.1 Session Based Test Management - SBTM

Problema tratado: Auto-gerência e Gerência da equipe.

Para superar as desvantagens descritas anteriormente James Bach e Jonathan Bach desenvolveram uma forma de melhor gerenciar o teste exploratório, chamada *Session Based Test Management* - SBTM, descrita por Jonathan Bach em [29][26].

Eles propuseram a seguinte estrutura como forma de suprir a necessidade de gerenciamento e medição do teste exploratório: dividir as atividades de testes em sessões, que seriam a unidade básica de trabalho, estipular uma missão para cada sessão e adotar métricas do tempo relacionado às atividades de teste.

Uma sessão de testes deve ser executada ininterruptamente, começa com um documento, o *charter*, que contém a missão descrita de maneira sucinta e, às vezes, algumas

das táticas a serem usadas. Algumas empresas produzem casos de testes tão vagos que estes podem ser utilizados como missão para o teste exploratório. O motivo pelo qual as missões são descritas dessa maneira é para aproveitar ao máximo as habilidades e imaginação do testador de maneira que ele não fique preso ao um passo a passo e possa explorar livremente o produto.

O tempo médio de uma sessão deve ser de 90 minutos, podendo variar de acordo com a necessidade da missão, indo de 30 minutos até 2 horas. O que acontece em cada uma delas é de inteira responsabilidade do testador que deve cumprir a missão de acordo com suas habilidades. Uma sessão longa demais pode significar que a missão está vaga demais e fazer com que o testador perca o foco, e uma sessão muito curta pode significar que está específica demais e tirar a liberdade do testador.

As métricas TBS dizem respeito ao tempo estimado gasto pelo testador para realizar as três tarefas: ***S****etup*(S), design e execução de testes , do inglês ***T****est execution and design* (T), investigação de *bugs* e report, ***B****ug investigation and reporting* (B).

- Setup (S): Tempo estimado gasto na preparação da sessão tarefas como configuração da máquina, instalação de novas builds, atualização do aplicativo, impressão da documentação, etc, ou seja, qualquer atividade executada antes do teste, necessária para preparar o teste propriamente dito, ou durante o teste, caso alguma reconfiguração seja necessária.
- Design e execução de teste (T): Tempo estimado gasto durante o design e execução do teste, considerando que o teste e o design de novos testes acontecem simultaneamente durante o teste exploratório.
- Investigação de *bugs* e report (B): Tempo estimado gasto pelo testador para registrar os bugs encontrados no relatório e no banco de dados de erros.

Interpretar esses valores contribui para a melhoria do software, muito tempo gasto na configuração pode levar o gerente a solicitar builds melhores cuja configuração não demore tanto; muito tempo gasto para reportar bugs (B), significa que muitos erros foram encontrados e que o sistema ainda não tem uma versão estável; muito tempo gasto na execução de teste (T) representa o progresso feito, que o testador não encontrou muitos erros e o sistema cumpre suas funcionalidades.

Cada sessão produz um relatório que fornece informações sobre o que ocorreu durante a sessão e que possa ser examinado por terceiros. Após cada sessão o testador relata

sua sessão ao gerente ou líder de testes, (*debriefing*), com o objetivo de explicar o que foi feito e receber algum *feedback* sobre a sessão. Foi notado que essa prática constante desenvolve um profundo entendimento do processo, do quanto pode ser feito em cada sessão e da quantidade de trabalho envolvido em cada ciclo, permitindo até prever o quanto o teste vai demorar mesmo que não existe um plano de testes detalhado.

3.4.2 Exploratory Testing in Pairs

Problema tratado: Pouca Experiência do Testador, Falta de Foco da Equipe, Desconhecimento do Domínio da Aplicação, Documentação e Reprodução de Testes.

Baseado em suas observações e de seus colegas e nas vantagens da Pair Programming, Cem Kaner e James Bach, passaram a aplicar a prática de trabalho em pares aos testes. Em [28], eles descrevem o teste exploratório em pares que possui características semelhantes ao modo com que é praticado aplicando-se SBTM, com charter, missão, sessão com duração limitada.

Nas sessões de testes realizadas em pares, o teste é de responsabilidade de um dos testadores que recruta outro para a sessão existindo a possibilidade de troca, ou não, de parceiro. Os dois testadores dividem uma máquina, enquanto um utiliza o dispositivo o outro sugere testes, estratégias, faz anotações, faz perguntas. Os papéis devem ser revezados entre os parceiros.

A tarefa escolhida pelo par, dentre as detalhadas no projeto, deve levar um dia ou menos, o chater pode conter os riscos envolvidos, que ferramentas usar, táticas, saídas esperadas, etc e a sessão dura entre 60 e 90 minutos. A prática dos testes exploratórios em pares possui os seguintes benefícios, dentre os quais podemos citar:

- Força os testadores a compartilhar suas idéias de forma lógica e esse processo aumenta o foco na atividade realizada, estimula a criação de novas estratégias e o surgimento de novas e melhores idéias;
- Estimula a criatividade dos testadores;

- Encoraja o testador a permanecer focado no teste, seguindo sua linha de pensamento, enquanto o outro faz as anotações reduzindo as interrupções;
- O segundo testador pode tentar reproduzir os resultados encontrados em outra máquina, ou anotar ideias interessantes para outros testes;
- Desencoraja interrupções por parte de outros membros da equipe;
- Como tudo o que foi reportado é revisado pelos dois testadores, aumenta a qualidade do relatório de bugs e a capacidade de reprodução dos testes;
- Ajuda no treinamento de testadores principiantes, se estes forem pareados com um testador mais experientes, dada a constante troca de idéias;
- Mesmo testadores experiente podem se beneficiar se estiverem lidando com um domínio desconhecido.

Como qualquer técnica, também apresenta riscos, que nascem, principalmente, dos desafios de se trabalhar em equipe:

- O trabalho em par deve ser de realizado pelos dois testadores, devendo os dois contribuir com o teste, o testador menos experiente, deve expressar e testar suas idéias tanto quanto o seu parceiro e não ser tratado como um mero assistente;
- Algumas pessoas possuem personalidades fortes e não trabalham bem em grupo, havendo a necessidade de um *coach* que os ajude a entender suas responsabilidades e a trabalhar em conjunto;
- Algumas pessoas são mais introvertidas e necessitam de um tempo para trabalhar sozinhas e aliviar um pouco a pressão do trabalho em grupo.

3.4.3 Team Exploratory Testing Sessions - TET

Problema tratado: Pouca Experiência do Testador, Falta de Foco da Equipe.

Em [36] Soili Saukkoriipi e Ilkka Tervonen propõem o Team Exploratory Testing Sessions (TET) ou Teste Exploratório em Equipe. Os autores definem a abordagem e realizam um estudo de caso aplicando-a em uma empresa de telecomunicações, que desenvolvia seu próprio software, com o propósito de definir, testar a abordagem e de

aprimorá-la. A pesquisa é feita através da realização de três sessões ao longo do desenvolvimento do aplicativo e entrevistas com os participantes.

Em seu trabalho eles descrevem a abordagem, seus parâmetros, também descrevem como pode ser aplicado na prática, seus riscos e apresentam possíveis soluções. A abordagem consiste de três elementos principais: a equipe, o teste exploratório e a sessão, que combinados funciona como a aplicação de SBTM em grupo.

A Equipe: A equipe deve ser pequena, cinco à dez participantes, para que o grupo não fique muito disperso, com habilidades complementares e que possuem um objetivo em comum definido.

Os membros da Equipe devem possuir diferentes habilidades, experiência e papéis definidos no time. Um bom entrosamento é essencial para que todos os participantes se sintam à vontade para fazer sugestões, questionamentos e assim contribuir de forma positiva com o tarefa sendo realizada. É importante também que a tarefa, ou missão, seja desafiadora, importante para a empresa e/ou os clientes, para que a equipe permaneça estimulada pelo trabalho. Dentro da equipe, os membros possuem papéis diferentes, eles devem ter experiência em testes, conhecimento sobre o domínio e trabalhar bem em equipe.

- **O Facilitador:** Cuida do aspecto prático da sessão e certifica-se da qualidade dos resultados reportados. Para alcançar este objetivo, ele define como a abordagem será utilizada, ajusta o processo às necessidades do projeto e o mantém atualizado. Organiza as sessões, define seus parâmetros e o processo de execução, como por exemplo: selecionar cuidadosamente e convidar possíveis membros da equipe, providenciar o ambiente de testes, equipamento adequado e configuração do mesmo, analisa e reporta os resultados, recebe o *feedback* da equipe. Dá apoio durante as sessões, para isso deve ter conhecimento sobre o domínio da aplicação alvo, testes feitos previamente, além de conhecimento sobre TET. Se houver tempo pode agir como Participante;
- **O Especialista do Domínio:** Apresenta o software alvo, ou sua versão atual, aos participantes e dá apoio a sua utilização durante os testes. Comenta os defeitos e sugestões de melhorias, como falar a equipe se determinado erro já foi reportado ou planejado. Deve conhecer o sistema em desenvolvimento

detalhadamente e ser capaz de apresentar uma visão geral do domínio de forma resumida e compreensível. Se houver tempo pode agir também como Participante;

- **O Participante:** Comparece às sessões de testes, realiza os testes exploratórios no sistema alvo, compartilha informação com a equipe e reporta seus achados durante a sessão. Após o final da sessão, dá um feedback para o facilitador sobre a sessão e a abordagem em si;
- **O Gerente do Domínio:** Deve conhecer bem a abordagem a fim de sugerir candidatos adequados aos papéis de Facilitador e Especialista de Domínio, julgar seus resultados e determinar se outras sessões são recomendadas. Reserva o horário para os testes, paga por outras possíveis despesas, acompanha os resultados da utilização da abordagem, da um feedback para sua melhoria, mas não participa diretamente da sessão.

Os papéis de Facilitador e Especialista do Domínio devem ser permanentes para que sejam desempenhados de forma eficiente, pois juntos eles são os responsáveis pelo sucesso da sessão de testes.

A Sessão: Similar a sessão definida no Teste Exploratório Baseado em Sessão, uma unidade de trabalho de testes que possui uma missão, é realizada sem interrupções e produz um relatório que pode ser revisado posteriormente, cuja principal diferença é o maior número de participantes.

São importantes para os resultados da sessão a definição de alguns parâmetros. (*i*) O mais importante é o objetivo da sessão que tem grande impacto na definição dos outros parâmetros. (*ii*) Os participantes, deles dependem o sucesso da sessão, devem ser selecionados aqueles que trabalhem bem em equipe, com habilidades distintas para que o software seja testado por diferentes pontos de vista. (*iii*) O software alvo, sua versão, que deve ser a mesma utilizada por todos os membros e configurada da mesma forma, condições do software, a maturidade, a estabilidade, a testabilidade, e a área focada durante a sessão.

(*iv*) O tempo de duração deve ser no mínimo duas horas, a frequência a cada duas iterações do desenvolvimento, para que novas funções e as mudanças sugeridas tenham tempo de ser analisadas e implementadas, bem como os bugs serem corrigidos, e quando

realizar as sessões, aconselha-se que sejam no início ou no meio da iteração de desenvolvimento, para que não coincida com o período mais atarefado da equipe.

(*v*) As ferramentas de apoio devem ser providenciadas, preparadas e checadas pelo facilitador, devem ser conhecidas pelos membros da equipe ou de fácil aprendizagem. Contas de usuários que devem ser criadas para os participantes se necessário, nomes de usuário e outros parâmetros da conta devem estar corretos e no contexto do teste, não sendo apropriado o uso de contas pessoais. As técnicas de testes utilizadas durante a sessão também devem ser de conhecimento de todos, ou de fácil assimilação.

Como existe a possibilidade de que alguma técnica, abordagem ou ferramenta sejam desconhecidos, bem como o próprio teste exploratório e o TET, e para evitar que isto tenha alguma consequência negativa, no início de cada sessão deve ocorrer uma introdução feita pelo facilitador, que apresenta o software alvo, a abordagem, ou seja o próprio TET, os objetivos da sessão, o template do relatório e as ferramentas de apoio, e pelo especialista do domínio, que apresenta o domínio.

Os resultados de cada sessão devem ser reportados ao gerente do domínio e outros interessados. O planejamento de futuras sessões de testes bem como o acompanhamento da abordagem são feitos baseados nesses relatório.

Como toda abordagem, apresenta riscos, a maioria relacionados aos parâmetros listados, se tais parâmetros não forem considerados cuidadosamente podem afetar negativamente os resultados. Os autores reconhecem os riscos e apresentam possíveis soluções, dentre vários outros, temos:

- Se o objetivo da sessão não estiver claro, ou totalmente ausente ou grande número de objetivos e/ou contraditórios, os resultados serão mínimos e o tempo da sessão terá sido desperdiçado. Para prevenir que isto aconteça os objetivos devem ser claros, condizentes com o contexto, em pequeno número e relacionados entre si;
- Se os participantes possuem pouca experiência e o software alvo estiver em mau estado, não será possível encontrar defeitos e o relatório ficará incompleto. Para que isso não ocorra é necessário que a experiência do testador e as condições do software sejam levadas em consideração durante a seleção dos participantes, e que haja um equilíbrio entre elas;

- Se não houver um controle da versão e configuração do software testado os resultados não são confiáveis. Comportamentos inesperados podem ser causados problemas na configuração e o teste feito em uma versão antiga desperdiça tempo com erros já corrigidos. Para que isso seja evitado todos os participantes devem ter a mesma versão e configuração do software alvo, de preferência a mais atual;
- Se a sessão for longa demais pode ser difícil encontrar candidatos a participar, já que outras tarefas teriam que ser adiadas por muito tempo. A sessão também não pode ser muito curta pois causaria que um número de ciclos de teste insuficiente e tempo curto para reagir aos bugs encontrados. Durante o estudo, foi constatado que uma sessão deve durar no mínimo uma hora, que esse tempo deve ser ajustado de acordo com o feedback da equipe e que duas horas, em média, seriam suficientes;
- Se o especialista do domínio não tiver um conhecimento sólido e aprofundado sobre o domínio, os participantes não terão o todo apoio necessário o que pode afetar negativamente a qualidade dos resultados encontrados. Para que isto seja evitado a seleção do especialista do domínio deve ser bastante criteriosa, ele deve ser uma pessoa que conhece a situação atual do aplicativo testado, a área do domínio e ser capaz de se expressar claramente.

Outra similaridade com o SBTM é que a sessão também está dividida em três principais tarefas que englobam as demais: Setup, quando os parâmetros são definidos e ambientes preparados pelo facilitador; Design e execução de teste, principal fase da sessão, quando a introdução e exploração do sistema acontece, e Investigação de *bugs* e report, quando todos os achados são investigados, avaliados e os que forem julgados pertinentes são reportados.

3.4.4. Tours.

Problema tratado: Pouca Experiência do Testador, Falta de Foco, Aplicação de Testes Exploratórios a Sistemas de Tempo Real

Em seu livro Exploratory Software Testing, [32], James A. Whittaker, compara explorar um sistema com a exploração de um turista faz quando visita uma cidade, sem qualquer espécie de guia, o turista perambula a esmo, mas com a ajuda de um guia ele pode conhecer pontos interessantes da cidade visitada. O mesmo acontece com o testador, explorando um sistema com o qual não está familiarizado, sem nenhum guia ele pode passear pela interface do sistema procurando por erros que podem ou não estar lá e nunca fazer nenhum progresso real.

Para ajudar a superar este problema ele propõe o que ele chama de *tours,* que fornece objetivos e uma estratégia aos testadores. Nos guias turísticos, as atrações das cidades geralmente são divididas em distritos: o distrito de negócios/comercial, o distrito do entretenimento, o dos teatros, o histórico e assim por diante, com esses segmentos frequentemente representando limites físicos. Para o testador tais limites são lógicos separam as funcionalidades em grupos que possuem características comuns.

Os Tours são agrupados em seis grupos principais de acordo com suas características, são eles:

- O Distrito Comercial (Business district): Em uma cidade é onde ficam os bancos, lojas, escritórios. No software é onde o "negócio" acontece, são as funcionalidades pelas quais o usuário utiliza o aplicativo, as tours desse distrito guiam os testadores por caminhos que utilizam essas funcionalidades. Para citar algumas, temos a Tour do dinheiro, a Tour do livro-guia, a Tour do intelectual, a Tour FedEx;
- O Distrito Histórico (Historical district): Nas cidades é onde existem lugares históricos ligados à eventos do passado. No software o distrito histórico diz respeito ao código legado e histórico de bugs e funcionalidades defeituosas. Geralmente de difícil compreensão quando o código legado foi submetido a modificações, utilizado em novos aplicativos, etc. Tours por esse distrito são destinadas a esse tipo de código;

- O Distrito Turístico (Tourist district): Nas cidades, os distritos turísticos são frequentados predominantemente por turistas, sendo evitado pelos moradores que preferem evitar as multidões. No software, novos usuário geralmente são atraídos por funcionalidades que os usuários mais experientes não mais utilizam;
- O Distrito do Entretenimento (Entertainment district): Nas cidades além de visitar o lugares turísticos, o visitante também procura outras opções de entretenimento para complementar sua experiência. No software, são funcionalidades que suportam e complementam as outras funcionalidades, e tais tours são destinadas a testá-las, complementando os tours pelos demais distritos;
- O Distrito Hoteleiro (Hotel district): Os hotéis são lugares onde os turistas descansam ao final do dia. Os softwares , mesmo aparentemente "em descanso", possuem vários processos em funcionamento, os tours desse distrito testam essas funcionalidades;
- O Distrito Suspeito (Seedy District): Áreas suspeitas fazem parte das cidades, são onde as pessoas cometem atos e operações ilegais. No software, as tours deste distrito são destinadas a encontrar vulnerabilidades no produto que possam prejudicar seus usuários.

Uma das desvantagens mencionadas anteriormente é que os testes exploratórios não são aplicáveis a sistemas de tempo real mas Phil Laplante, em [33], inspirado nessa analogia, expande os casos em que os tours podem ser aplicados e sugere tours que poderiam ser utilizadas para explorar sistemas críticos, de tempo real e embarcados e assim aplicar o teste exploratório a esse tipo de sistema.

Ele considera as fontes de incertezas nos sistemas para criar tours que as explore e desenvolver um conjunto de testes para encontrá-las. Essas explorações podem ser convertidas em casos de uso para testes automatizados e de regressão. O autor ainda afirma que essas tours foram extensivamente utilizadas por ele e colegas no teste de vários sistemas de tempo real para aplicações de aviação, sistemas de satélite e outros sistemas de navegação e que pretendem aplicá-las a outros tipos de sistemas embarcados e de tempo real. Os tours propostas em [33] são:

- Tour do Mau Tempo (Bad Weather Tour): Realiza explorações no ambiente. Cria simulações que modelam qualquer tipo de adversidades e incertezas no ambiente onde

o sistema opera, que podem surgir de anomalias ou perturbações no funcionamento do sistema operacional como quedas de energia causadas por fatores externos, etc;

- Tour de Murphy (Murphy's Tour): Possui esse nome porque tudo que pode dar errado é feito para dar errado. Realiza explorações baseados em incertezas nos valores de entradas provocadas por interrupções perdidas, dados anômalos ou dados propositalmente errados que podem causar falhas em cascata e sobrecarregar o sistema;
- Tour Mistérios Mágicos (Magic Mystery Tours): Exploram as saídas fornecidas pelo sistema de controle do software ao aplicativo sob controle, que podem ser grosseiramente ou ligeiramente defeituosas. Tais saídas causam um comportamento inesperado do aplicativo que por sua vez fornece entradas erradas ao sistema, esse feedback anômalo pode causar uma falha no sistema de controle;
- Tour Medo e Ódio em Las Vegas(Fear and Loathing in Las Vegas Tour): Exploram o estado do sistema: falhas internas, como falhas nos contadores de programas, pode levar a incertezas no estado do programa difíceis de diagnosticar e quase impossíveis de recuperação do programa. O comportamento errático (bêbado) do sistema é a causa do nome dado a esse tipo de tour;
- Shakeout Tours: Exploram erros de linguagem, que possam ter sido introduzidos durante a compilação do programa, onde uma linha do código fonte ausente ou alterada pode modificar a saída do compilador e levar a efeitos colaterais indesejados. Logo existe a necessidade de testes que testem o compilador e os outros programas relacionados à produção de código executável do sistema;
- Reagan's Tours: "Confie mas Verifique". Explora a interação entre os sistemas. Procuram erros relacionados a programas de terceiros, comerciais ou open-source, que interagem com o sistema sendo testado.

3.4.5 Rule-Based Exploratory Testing - R-BET.

Problema tratado: Reprodutibilidade dos Testes

Em [34], Theodore D. Hellmann, e [35] em parceria com Frank Maurer, descrevem uma abordagem que combina Testes Exploratórios e Testes Baseados em Regras (

Rule-Based Testing) para testes em interfaces de usuário (Graphical User Interface - GUI), com o objetivo de criar testes relevantes e eficazes contando com inteligência dos testadores aplicada no primeiro e com a capacidade de verificação e repetição do segundo.

Segundo os autores "60% dos defeitos nos aplicativos podem ser encontrados na codificação da interface e 65% desses erros resultam em perda de funcionalidade" o que tem grande impacto na experiência do usuário. Ainda, considerando que a interface de um aplicativo pode sofrer inúmeras alterações durante seu desenvolvimento, durante as atualizações e manutenção, os testes de regressão podem, frequentemente resultar em falsos positivos mesmo que a codificação da funcionalidade esteja funcionando corretamente.

Hellmann e Maurer definem Testes Baseados em Regras, ou Verificação Baseada em Regras (Rule- Based Verification), como uma modalidade de testes automatizados que pode simplificar a lógica da validação e verificação dos testes, bem como reduzir as chances do teste falhar como consequência de uma mudança na interface.

A aplicação da abordagem proposta se dá da seguinte forma: O testador interage com a aplicação durante uma sessão de testes exploratórios, que é gravada por uma ferramenta de capture/replay, essa sessão posteriormente é utilizada como script para o teste automatizado. Em seguida, um conjunto de regras curtas e automatizadas é criado pelo testador que pode ser utilizado para definir o comportamento esperado do aplicativo. As regras e o script são combinados em um teste de regressão automatizado que amplia o espaço de estados do sistema que é testado, aumentando a quantidade de verificação quando o teste é repetido.

Utilizando essa abordagem o testador utiliza o teste exploratório para identificar regiões do sistema que podem necessitar de mais testes. Essa região de interesse é, então, submetida a testes adicionais minuciosos utilizando as regras criadas para guiar os testes automatizados. As regras definidas possuem um conjunto de pré-condições e efeitos. As pré-condições diminuem a ocorrência de falsos positivos provocados pela mudança na GUI, já que o teste não é disparado caso a pré-condição não seja atendida.

As regras seguem a forma de uma declaração "*if..try..catch"*. A pré-condição, "*if*", garante que o teste só será disparado se ela for atendida, podendo existir mais de uma pré-condição ligadas por um "and", que devem ser todas atendidas para que o teste seja realizado. A mesma pré condição pode ser aplicada à mais de uma regra.

A ação, "*try*", representa o corpo principal da regra que será executado mediante as pré-condições. As consequências, "*catch*", determina o que acontece se o *try* falha ou lança

uma exceção. Neste caso falhas indicam bugs no aplicativo e *warnings* indicam que o comportamento precisa ser tratado.

A utilização desta abordagem reduz a quantidade de trabalho que deve ser realizado por um testador humano ao mesmo tempo que aumenta o número de espaços diferentes a partir dos quais essa verificação pode ser realizada. Ainda, regras podem ser definidas para realizar testes que não seriam feitos por falta de tempo, pouca experiência do testador ou erros que possam passar despercebidos.

Os autores afirmam que R-BET não só permite que um sistema seja testado mais detalhadamente do que seria possível em um determinado prazo como também libera os testadores para realizar testes mais interessantes. Ele não reduz os problemas encontrados nos testes de GUI, apenas representa uma maneira de simplificá-los, tal que tais dificuldades possam ser atenuadas.

4 Ferramentas de Apoio ao Teste Exploratório

Uma das formas de tentar superar os problemas dos testes exploratórios, principalmente a dificuldade de medir a cobertura dos testes e de reproduzir os testes, como também produzir reports e relatórios, foi a utilização de ferramentas durante a realização do testes. Neste capítulo iremos apresentar as ferramentas identificadas através da revisão bibliográfica realizada neste trabalho, relacionando-as aos problemas que podem ser atenuados com sua utilização. Uma vez identificadas na revisão as ferramentas foram utilizadas e uma análise de cada ferramenta foi realizada.

Do total de 11 ferramentas estudadas, parte delas (7) são ferramentas pagas, por essa razão a análise foi realizada em suas versões *trial*, que podem ou não disponibilizar todas as suas funcionalidades, um menor número (4) consiste de ferramentas open source.

4.1 Análise de Ferramentas

As ferramentas analisadas para este trabalho foram classificadas de acordo com sua funcionalidade e utilização em relação aos testes exploratórios, como vemos na Tabela 1. Na Tabela 2 vemos como elas são classificadas em relação a plataforma testada enquanto que na Tabela 3 temos as versões disponíveis e versões testadas. Na Tabela 4 temos um resumo da análise das ferramentas relacionando-as aos problemas por elas atacados.

FERRA-MENTAS	Report Textual	Report Textual + Captura de Tela	Report Textual + Captura de Tela+ Videos	Capture Replay	Gravação de Audio	Permite anexos ao report e/ou ao planejamento	Permite importar planos de testes/ missões	Permite exportar planos de testes/ missões	Geração/ Injeção de dados	Cronômetro
TEST STUFF*	X	X	X			X	X			X
RAPISE				X						
TESTPAD	X					X				
BUG MAGNET									X	
TESTRAIL*	X					X	X	X		X
WINK	X	X	X		X	X	X	X		
JIRA + JIRA CAPTURE*	X					X				
BB TEST ASSISTANT	X	X	X		X			X		
RAPID REPORTER	X	X								X
TEST STUDIO EXPLORER				X						
EXPLORATORY TESTIG CHROME EXTEN-SION	X	X				X		X		

Tabela 1 - Ferramentas classificadas em relação às funcionalidades. (*) Identificam ferramentas que possuem funcionalidades que auxiliam no gerenciamento dos testes.

FERRAMENTAS	Testes para Web	Testes para Desktop	Testes para Mobile
TESTSTUFF	X	X	X
RAPISE	X	X	X
TESTPAD	X	X	X
BUG MAGNET	X		
TESTRAIL	X	X	X
WINK	X	X	X
JIRA + JIRA CAPTURE	X		
BB ASSISTANT	X	X	
RAPID REPORTER	X	X	X
TEST STUDIO EXPLORER	X	X	X

Tabela 2 - Ferramentas classificadas em relação à plataforma testadas.

Abaixo vemos as versões disponíveis e versões testadas:

FERRAMENTAS	Trial	Open Source	Versão Mobile	Versão Desktop	Versão Web	Versão Testada
TESTSTUFF	X			X	X	Desktop
RAPISE	X		X	X	X	Desktop
TESTPAD	X		X	X	X	Web
BUG MAGNET		X			X	Web
TESTRAIL	X			X	X	Web
WINK		X		X		Desktop
JIRA + JIRA CAPTURE*	X			X	X	Web
BB ASSISTANT	X			X		Desktop
RAPID REPORTER		X		X		Desktop
TEST STUDIO EXPLORER	X			X	X	Desktop/Web**
EXPLORATORY TESTING CHROME EXTENSION		X			X	Web

Tabela 3 - Ferramentas classificadas em relação às versões disponíveis e versões testadas(*)Jira possui versões web e desktop, mas o Jira capture possui apenas a extensão para navegador. (**)Necessário usar ambos, para a utilização correta da ferramenta

A seguir temos uma breve apresentação das ferramentas analisadas e os possíveis problemas por elas atacados. Devemos levar em consideração que diferentes testadores utilizam as ferramentas de formas diferentes e que a utilidade das ferramentas não se limita às aqui descritas, podendo ajudar a solucionar outros problemas não listados, de formas inesperadas. Para mais detalhes sobre as ferramentas ver Apêndice A.

F1: Teststuff

Descrição: Ferramenta desenvolvida para execução de testes, oferece ao testador, entre outras funcionalidades, uma forma de realizar o report mais facilmente. Põe em prática conceitos do teste exploratório baseado em sessão, com medição de tempo. O que permite a utilização da métrica TBS utilizando o tempo da sessão como parâmetro para determinar a cobertura, possíveis causas dos bugs, entre outros.

Problemas solucionados: Estruturação do teste exploratório, medição da cobertura baseada na métrica TBS, report produzido.

F2: Rapise

Descrição: Ferramenta desenvolvida para auxiliar testes automatizados, como a ferramenta grava o passo a passo do testes, permite que o testador revise o teste, faça um report mais completo e repita o teste realizado.

Problemas solucionados: Auxiliar na documentação, no report produzido e repetição dos testes.

F3: TestPad

Descrição: Aplicação web que oferece uma maneira rápida e eficiente de organizar planos de testes em uma checklist, no caso do teste exploratório baseado em sessão, pode ser usado para organizar as missões da sessão com descrições breves do que deve ser testado.

Problemas solucionados: Auxiliar na documentação, no report produzido, estruturação do teste exploratório, medição da cobertura.

F4: Bug Magnet

Descrição: Extensão para Chrome e Firefox, utilizado para testes de aplicações web. Funciona como fonte de dados que possam ser utilizados durante os testes, auxilia a diminuir a pouca experiência no domínio da aplicação.

Problemas solucionados: Falta de experiência no domínio da aplicação.

F5: TestRail

Descrição: Ferramenta web e desktop, auxilia no gerenciamento e realização de testes. Auxilia na cumentação, no report produzido, estruturação do teste exploratório, medição da cobertura baseada na métrica TBS sessões de testes exploratórios, com medição de tempo que pode ser utilizado como parâmetro na métrica TBS, como forma de avaliar o sistema alvo.

Problemas solucionados: Documentação, report produzido incompleto, estruturação do teste exploratório, medição da cobertura baseada na métrica TBS

F6: Wink

Descrição: Ferramenta desenvolvida para criação de tutoriais e apresentações cujas funções podem ser utilizadas por testadores para auxiliar no reporte de erros, pois são similares às funcionalidades apresentadas por ferramentas dedicadas a testes de software.

Problemas solucionados: Auxilia no report do teste.

F7: Jira Capture

Descrição:O capture for Jira é uma extensão do JIRA que pode ser adicionada ao browser do usuário, permite que o usuário faça o report da sessão. Possui recursos que permite fazer anotações sobre os bugs encontrados, capturar tela e anexar aos relatos de bugs.

Problemas solucionados: Auxilia no report do teste.

F8: BB Assistant

Descrição: Ferramenta cuja principal funcionalidade é a captura em vídeo do teste realizado, permite que o testador realize a exploração do sistema sendo testado sem interrupções. Permite que o testador revise o teste, faça um report de bugs mais completo e repita o teste realizado.

Problemas solucionados: Auxiliar na documentação, no report produzido e reprodução dos testes.

F9: Rapid Reporter

Descrição: Ferramenta desenvolvida para dar suporte ao teste baseado em sessão, onde suas funcionalidade refletem as principais características da abordagem, como a capacidade de cronometrar a sessão e dessa forma aplicar a métrica TBS, que pode ser utilizada na avaliação do sistema alvo.

Problemas solucionados: Auxiliar na documentação, no report produzido, estruturação do teste exploratório, medição da cobertura baseada na métrica TBS

F10: Test Studio Explorer

Descrição: Ferramenta desenvolvida para realização de testes automatizados para dispositivos móveis, web e desktop, como sua principal funcionalidade é gravar o passo a passo do testes, permite que o testador revise o teste, faça um report mais completo e repita o teste realizado.

Problemas solucionados: Auxiliar na documentação, no report produzido e reprodução dos testes.

F11: Exploratory Testing Chrome Extension

Descrição: Extensão para Chrome para testes em aplicações web, ferramenta desenvolvida para auxiliar nos testes exploratórios, permite que o testador faça o report da sessão, onde o usuário é capaz de reportar bugs, fazer anotações, anotar idéias/ oportunidades.

Problemas solucionados: Auxilia documentação produzida e no report do teste.

Na tabelas seguintes apresentamos um resumo da análise relacionando as ferramentas analisadas aos problemas por elas solucionados, a tabela 4 mostra os problemas, onde cada problema é identificado pela letra P acompanhada de um número, e a tabela 5 mostra a relação entre os problemas e as ferramentas analisadas.

P1:	Qualidade da documentação produzida, reporte de erros incompletos.
P2:	Impossibilidade de precisar a cobertura devido a não existência de uma métrica.
P3:	Desconhecimento do domínio da aplicação, pouca experiência do testador.
P4:	Reprodução de testes, devido a pouca documentação.
P5:	Falta de estrutura aplicada ao teste exploratório, gerenciamento do teste exploratório.

Tabela 4 - Problemas associados à adoção do teste exploratório.

FERRAMENTAS	P1	P2	P3	P4	P5
TESTSTUFF	X	X			X
RAPISE	X			X	
TESPAD	X	X			X
BUG MAGNET			X		
TESTRAIL	X	X			X
WINK	X				
JIRA CAPTURE	X				
BB TEST ASSISTANT	X			X	
TEST REPORTER	X	X			X
TEST STUDIO EXPLORER	X			X	
EXPLORATORY TESTING CHROME EXTENSION	X				

Tabela 5 - Relação entre ferramentas e problemas solucionados.

5 Considerações Finais

Este trabalho foi realizado com o propósito de se fazer um estudo a respeito dos problemas associados à adoção do teste exploratório como um método utilizado nos testes de software, as soluções encontradas e quais ferramentas o apoiam. Através dele espera-se ter contribuído com a compilação das soluções apresentadas, bem como incentive novos adeptos a prática dos testes exploratórios.

5.1 Discussões e Lições Aprendidas

O que se aprendeu durante sua realização foi que os testes exploratórios não são considerados um tipo de testes mas sim uma abordagem, bastante eficiente e que pode adotar várias estratégias metodologias na sua utilização. Pode também ser utilizado em conjunto com outras metodologias de testes como uma espécie de complemento, em casos em que esta outra metodologia esgotou suas possibilidades de encontrar novos bugs no sistema em desenvolvimento.

Os testes exploratórios se mostram bastante eficientes em várias situações aqui discutidas e seu uso pode ser feito de várias maneiras, já que a abordagem se mostra bem flexível, principalmente em relação ao grau de exploração que se deseja dar ao teste do sistema. Devido a essa flexibilidade houve a criação do conceito de *scripted/exploratory continuum,* afirmando que todo teste é exploratório de alguma forma, o que resta decidir é o quão exploratório ele é.

A abordagem apresenta vários problemas associados à sua adoção. A maioria relacionada, principalmente, a pouca documentação, que torna difícil as tarefas de repetir os testes, precisar a sua cobertura e ao fato do teste depender fortemente do testador, suas experiência e habilidade.

Inúmeras soluções são sugeridas na literatura relacionada, dentre elas estão as apresentadas neste documento: a combinação dos testes exploratórios com outras metodologias, formas de gerenciamento do teste exploratórios e o uso de ferramentas como apoio a abordagem. As presentes neste trabalho tratam principalmente dos problemas

relacionados acima e fornecem uma forma de estimar a cobertura, documentar os bugs, o passo a passo e compensar a pouca experiência do testador.

5.2 Limitações do Estudo

Acreditamos que podem existir soluções para o teste exploratório que não foram identificadas neste trabalho, principalmente em relação à possibilidade de combinar o Teste Exploratório com de outras abordagens. O objetivo do nosso trabalho não foi levantar a lista completa de soluções existentes, mas sim de fazer um levantamento inicial destas soluções. Além disso esse trabalho tem como foco as soluções em que o testador humano tem uma participação ativa no processo, isto é, a solução tira vantagem da habilidade e conhecimento do testador humano que é quem realiza a exploração do sistema. Essa decisão foi tomada pois consideramos esta ser a característica mais importante dos testes exploratórios e adotada como condição para que abordagens fossem incluídas ou não no estudo.

Outra limitação encontrada neste estudo diz respeito ao conjunto de ferramentas analisadas. Esse conjunto está limitado as ferramentas citadas nas referências levantadas inicialmente. O foco foi analisar ferramentas que pudessem ser utilizadas sem a necessidade de algum outro tipo de aplicativo, repositório, banco de dados, etc, além do que é oferecido pela própria ferramenta, para seu funcionamento correto. Essa decisão foi tomada considerando a infinidade de ferramentas disponíveis e que analisar todas seria impossível mesmo levando em consideração esse requisito.

As ferramentas selecionadas foram testadas/analisadas diversas vezes e em diferentes momentos da elaboração deste trabalho, algumas delas sofreram atualizações e a análise aqui apresentada é da versão mais atual da data em que os testes foram realizados. Algumas das ferramentas apresentavam atualizações apenas na interface mantendo as mesmas funcionalidades da versão anterior, enquanto outras afirmam ter sofrido atualizações nas suas funções.

5.3 Trabalhos Futuros

Possíveis desdobramentos de trabalhos futuros relacionados a este trabalho incluem: (i) dar continuidade a revisão de literatura; (ii) disponibilizar a análise das ferramentas realizadas neste trabalho em um site ou artigo para que testadores exploratórios tenham

acesso, com o objetivo de auxiliar a comunidade de testes; (iii) a partir dessa pesquisa podemos elaborar abordagens de apoio ao teste de software que usem suites de ferramentas compostos pelas ferramentas analisadas; (iv) este trabalho também define características que são pouco cobertas pelas ferramentas existentes o que pode apontar para a necessidade para outras ferramentas de apoio ao teste exploratório.

Referências Bibliográficas

[1]KANER, Cem; FALK, Jack; NGUYEN, Hung Q. **Testing Computer Software**. John Wiley & Sons, 1999.

[2]BACH, James. **Exploratory testing explained**. 2003 Disponível em: <http://www.satisfice.info/articles/et-article.pdf > Acessado em: 25 abr 2018

[3]FIT, Andy Tinkham; FIT, Cem Kaner. **Exploring Exploratory Testing**. 2003. Disponível em: < http://kaner.com/pdfs/ExploringExploratoryTesting.pdf> Acessado em: 20 jun 2018

[4]RASHMI, N.; SUMA, V. Exploratory Testing: An Overview. **International Journal of Computer Applications**, v. 131, n. 10, p. 21-27, 2015.

[5]ITKONEN, Juha; RAUTIAINEN, Kristian. Exploratory testing: a multiple case study. In: **Empirical Software Engineering, 2005. 2005 International Symposium on**. IEEE, 2005. p. 10 pp.

[6]SVIRIDOVA, Tatyana; STAKHOVA, Daryna; MARIKUTSA, Ulyana. Exploratory testing: Management solution. In: **Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2013 12th International Conference on the**. IEEE, 2013. p. 361-361

[7]HAYES, Charlotte. **Structured Exploratory Testing - Where? What? How?.** 2016. Disponível em: <https://www.ten10.com/structured-exploratory-testing-where-what-how/>. Acessado em: 06 mar. 2018

[8]BACH, James. **History of Definitions of ET**. 2015. Disponível em: <http://www.satisfice.com/blog/archives/1504 > Acessado em: 06 mar 2018.

[9] EASTERBROOK, Steve. **The difference between verification and validation**. 2010 Disponível em: <http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/ > Acessado em: 07 mar 2018.

[10] SHARMA, Lakshay. **Difference between verification and validation.** 2017. Disponível em: < http://toolsqa.com/software-testing/difference-between-verification-and-validation/> Acessado em: 07 mar 2018.

[11] **Software Testing Levels**. Disponível em: <http://www.test-institute.org/Software_Testing_Levels.php > Acessado em: 04 mai 2018

[12] ERIKSSON, Ulf. **Differences Between the Different Levels of Tests.** 2014. Disponível em: <http://reqtest.com/uncategorised/differences-between-the-different-levels-of-tests/ > Acessado em: 05 mai 2018

[13] MILLER, Roy; COLLINS, C. Acceptance testing. **Proc. XPUniverse**, v. 238, 2001.

[14] WILLIAMS, Laurie ; SMITH, Ben; HECKMAN, Sarah. **Test coverage with EclEmma.** 2009. Disponível em:<http://realsearchgroup.org/SEMaterials/tutorials/eclemma/> Acessado em: 15 mai 2018

[15]**Top 17 Exploratory Testing Tools.** 2016. Disponível em <https://www.softwaretestinghelp.com/tools/top-17-exploratory-testing-tools/> Acessado em: 15 mar 2017

[16] AVRITZER, Alberto; WEYUKER, Elaine J. Generating test suites for software load testing. In: **Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis**. ACM, 1994. p. 44-57.

[17] LIU, Henry H. **Software performance and scalability: a quantitative approach**. John Wiley & Sons, 2011.

[18] **Scalability Testing: Complete Guide.** Diponível em: <https://www.guru99.com/scalability-testing.html> Acessado em: 20 jul 2018

[19]**Load Testing Tutorial: Tools, Process & Examples** Disponível em: < https://www.guru99.com/load-testing-tutorial.html> Acessado em: 20 jul 2018

[20]SAMEER, Syed Shujauddin. A Strategic Review of Exploratory Testing Techniques. **International Journal of Engineering Trends and Technology (IJETT)–Volume**, v. 10, 2014.

[21] KOHL, Elizabeth. **Session-Based Test Management.** Disponível em: <http://www.satisfice.com/sbtm/> Acessado em: 12 ago 2018

[22] MEMON, Atif; BANERJEE, Ishan; NAGARAJAN, Adithya. What test oracle should I use for effective GUI testing? In: **Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on**. IEEE, 2003. p. 164-173.

[23] **Processo de Testes**. 2011. Disponível em: <https://testesw.wordpress.com/processo-de-testes/> Acessado em: 12 abr 218

[24]SHAH, Syed Muhammad Ali et al. **Exploratory testing as a source of technical debt**. **IT Professional**, v. 16, n. 3, p. 44-51, 2014

[25] BOLTON, Michael. **Evolving understanding about Exploratory Testing**. 2008. Disponível em: < http://www.developsense.com/blog/2008/09/evolving-understanding-about/ > Acessado em 06 jun 2018

[26] BACH, Jonathan. **How to manage and measure exploratory testing**. 2006.

[27] NIDHRA, Srinivas; DONDETI, Jagruthi. Black box and white box testing techniques-a literature review. **International Journal of Embedded Systems and Applications (IJESA)**, v. 2, n. 2, p. 29-50, 2012.

[28]KANER, Cem; BACH, James. Exploratory testing in pairs. In: **STARWest 2001 Conference**. 2001.

[29]BACH, Jonathan. Session-based test management. **Software Testing and Quality Engineering Magazine**, v. 2, n. 6, 2000.

[30]GHAZI, Ahmad Nauman et al. Exploratory Testing: One Size Doesn't Fit All. **arXiv preprint arXiv:1704.00537**, 2017.

[31]BACH, James; BOLTON, Michael. **Exploratory Testing 3.0.**2015. Disponível em: < http://www.satisfice.com/blog/archives/1509> Acessado em: 07 jun 2018.

[32]WHITTAKER, James A. **Exploratory software testing: tips, tricks, tours, and techniques to guide test design**. Pearson Education, 2009.

[33]LAPLANTE, Phil. Exploratory Testing for Mission Critical, Real-Time and Embedded Systems. **IEEE Transactions on Reliability**, p. 449-482, 2009.

[34]HELLMANN, Theodore D. **Enhancing Exploratory Testing with Rule-Based Verification**. 2010. Tese de Doutorado. UNIVERSITY OF CALGARY.

[35] HELLMANN, Theodore D.; MAURER, Frank. Rule-based exploratory testing of graphical user interfaces. In: **2011 Agile Conference**. IEEE, 2011. p. 107-116.

[36]SAUKKORIIPI, Soili; TERVONEN, Ilkka. Team exploratory testing sessions. **ISRN Software Engineering**, v. 2012, 2012.

[37] GOODMAN, Leo A. Snowball sampling. **The annals of mathematical statistics**, p. 148-170, 1961.

[38] CAETANO, Cristiano. **Testes Exploratórios (parte 2): Gestão de testes exploratórios.** 2017. Disponível em: http://www.qualister.com.br/blog/testes-exploratorios-parte-2-gestao-de-testes-exploratorios- Acessado em: 25. 04. 2017

[39]BACH, Jonathan; BACH, James. How to Measure Ad Hoc Testing. **Software Testing and Quality Engineering**, 2000.

[40]DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John. **Automated software testing: introduction, management, and performance**. Addison-Wesley Professional, 1999.

[41]HOODA, Itti; CHHILLAR, Rajender Singh. Software test process, testing types and techniques. **International Journal of Computer Applications**, v. 111, n. 13, 2015.

Apêndice A - Análise Aprofundada das Ferramentas

Testsuff

Foi projetada para ser usada tanto por testadores quanto gerentes de testes e auxilia no gerenciamento dos testes pois suas funcionalidades ajudam a superar vários dos problemas enfrentados pelos testes exploratórios e põem em prática algumas das características dos testes baseado em sessão, tais como: através da criação de projetos que reúnem suites de testes, figura 6, o usuário pode criar ou editar um plano de teste já existente, especificar o tempo estimado de cada teste (ou duração da sessão), prioridade, pré-condições (Set-up), o passo a passo detalhado ou apenas a missão, dependência de outros testes, screenshot, adicionar *hyperlinks*, entre outros, como mostra a figura 5.

O usuário pode criar uma lista de requisitos e atribuir testes a cada um deles,dessa forma definir o escopo e determinar a cobertura dos testes realizados no sistema. O testador possui a lista de tarefas pendentes e clica em cada uma para dar início a suas respectivas sessões de testes figura 6. Durante a execução dos testes a ferramenta permite ao testador selecionar a área da tela que a ferramenta irá gravar e o vídeo é anexado ao teste automaticamente o que facilita a reprodução do teste quando necessário, através do Test runner, permite também que o testador reporte erros não relacionados ao passo a passo (Oportunidades), no formulário de report de bugs, ambos mostrados na figura 7.

Os testes executados podem ser tanto especificados na própria ferramenta, quanto importados do excel, figura 5, os resultados produzidos durante os testes são organizados de maneira clara e intuitiva, figuras 8 e 9, o aplicativo inclui detalhes de sistema e de hardware da máquina onde foi realizado, figura 9. Possui uma funcionalidade chamada Labs onde o gerente de teste especifica cada ciclo de testes, o que vai ser testado, qual testador faz determinada tarefa acessado a partir da tela inicial, figura 4. A ferramenta possui versões web e desktop e ambas estão disponíveis ao usuário, podendo ser usadas de forma intercambiável.

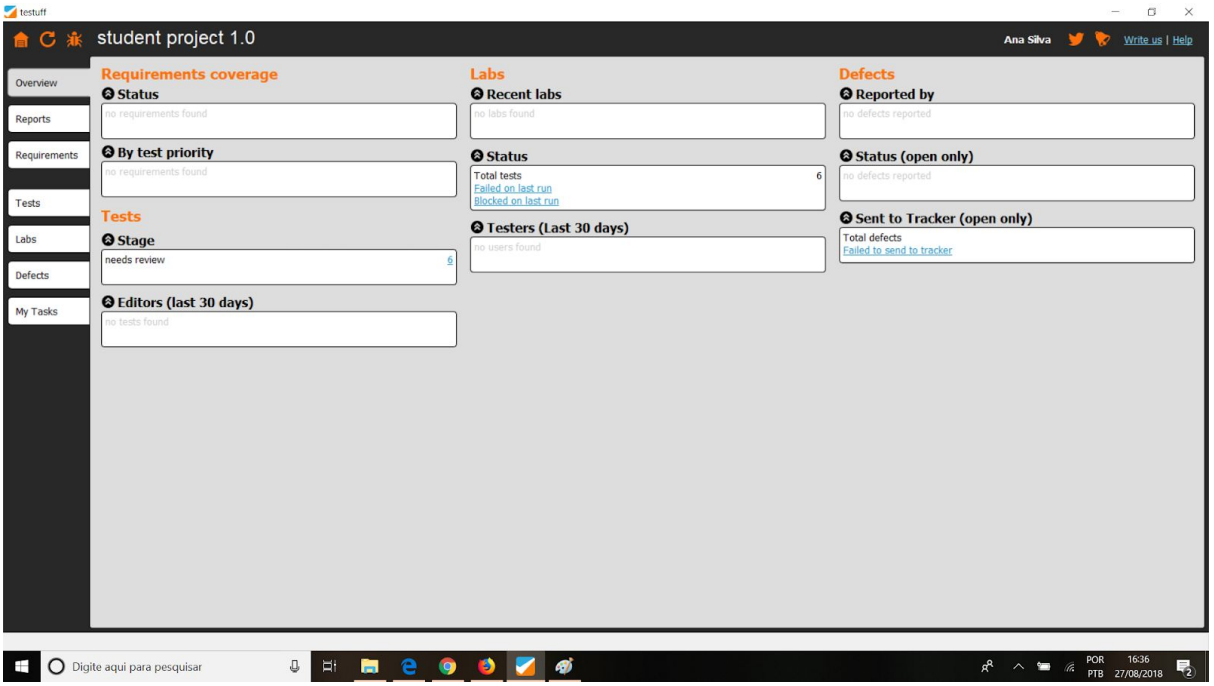

Figura 4: Teststuff - Tela inicial/gerenciamento.

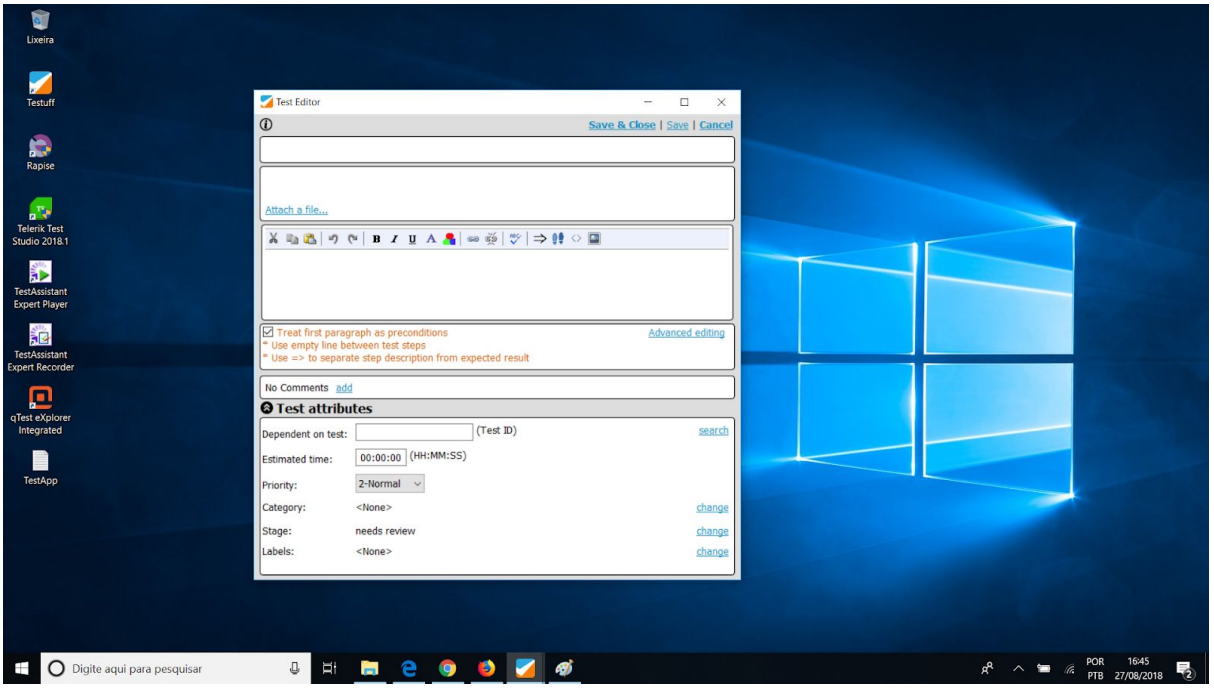

Figura 5: Editor de testes do Teststuff.

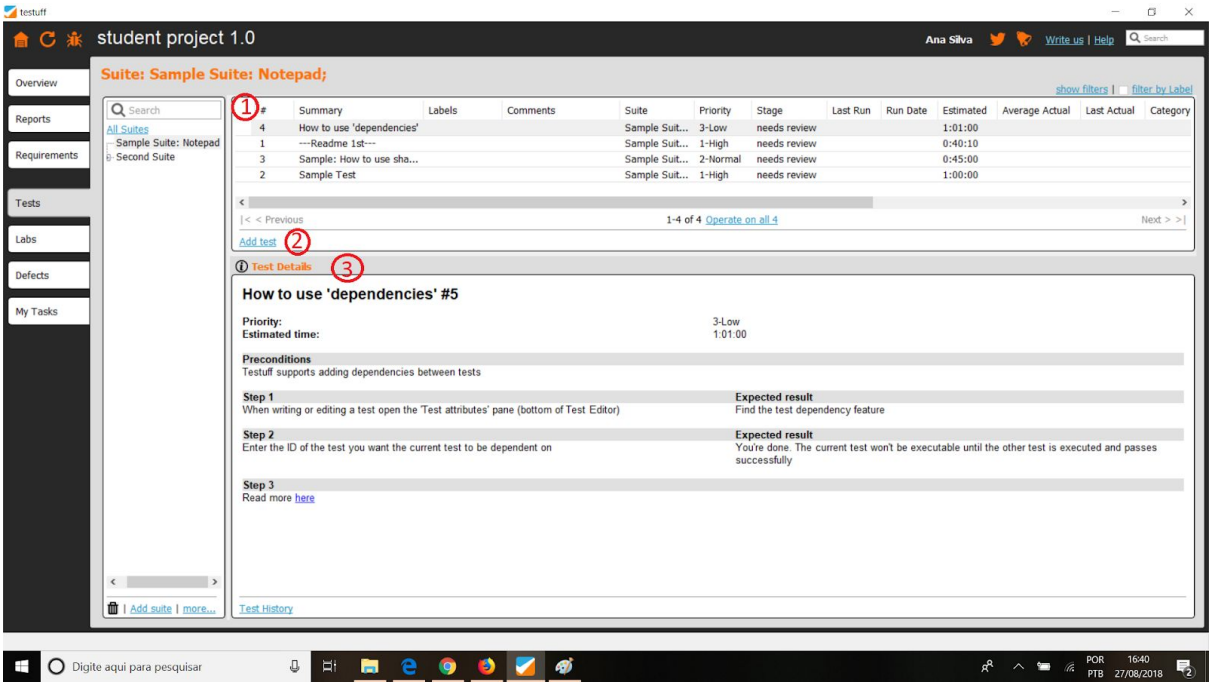

Figura 6: Teste criado no Teststuff 1 - Lista de testes da suite de testes; 2 - Criar teste; 3 - Detalhes do teste como: prioridade, duração, pré-condições, passos, resultados esperados.

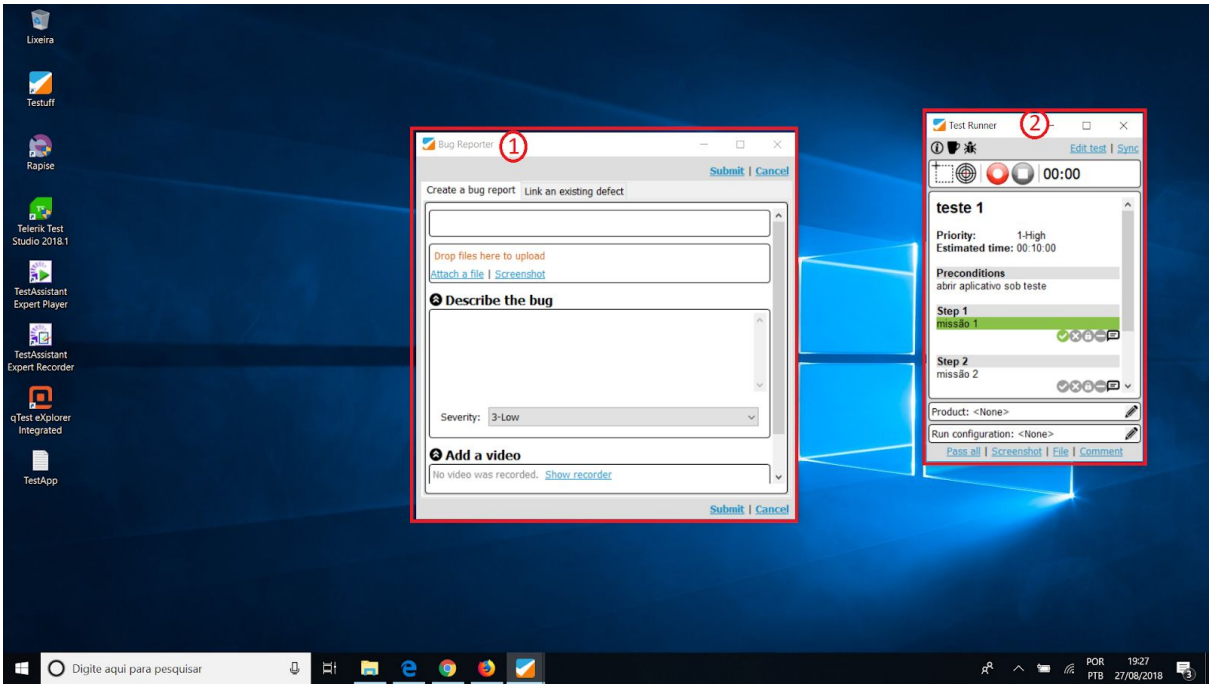

Figura 7: Teststuff - Test run. 1- Formulário de report; 2 - Test runner.

Os relatório do teste pode ser exibido de duas maneiras, no próprio aplicativo ou através do navegador.

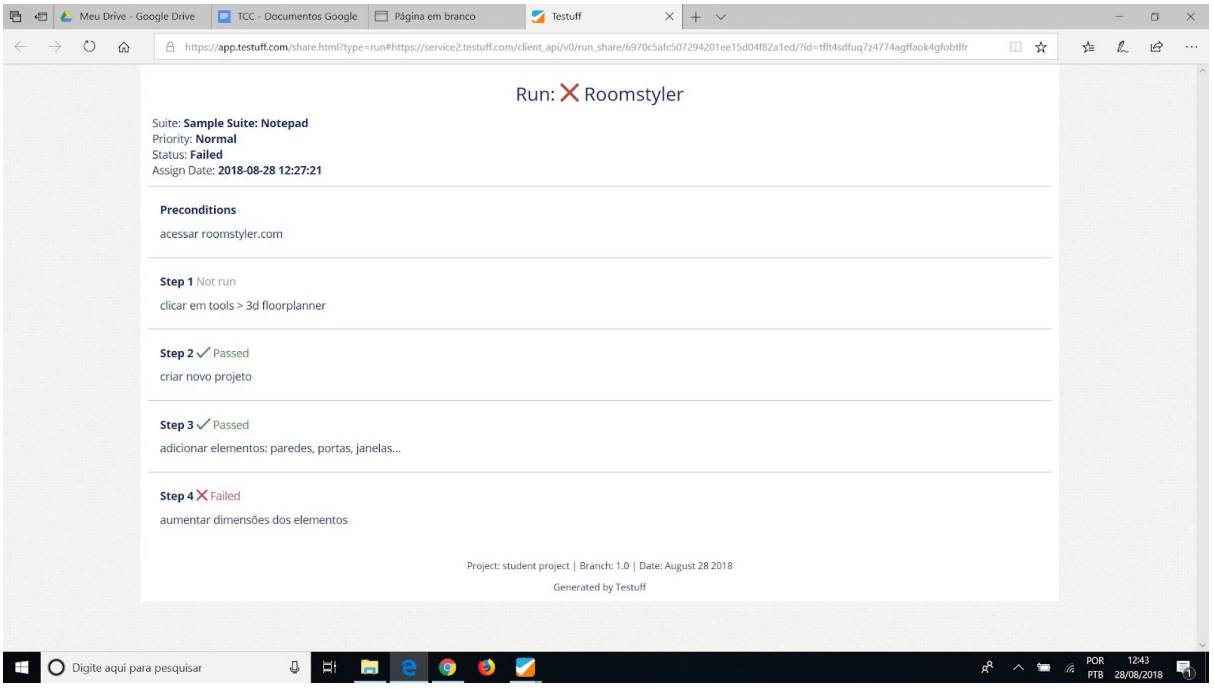

Figura 8: Teststuff - Resultado de teste feito no aplicativo desktop mas mostrado no navegador, link no resultado mostrado no aplicativo.

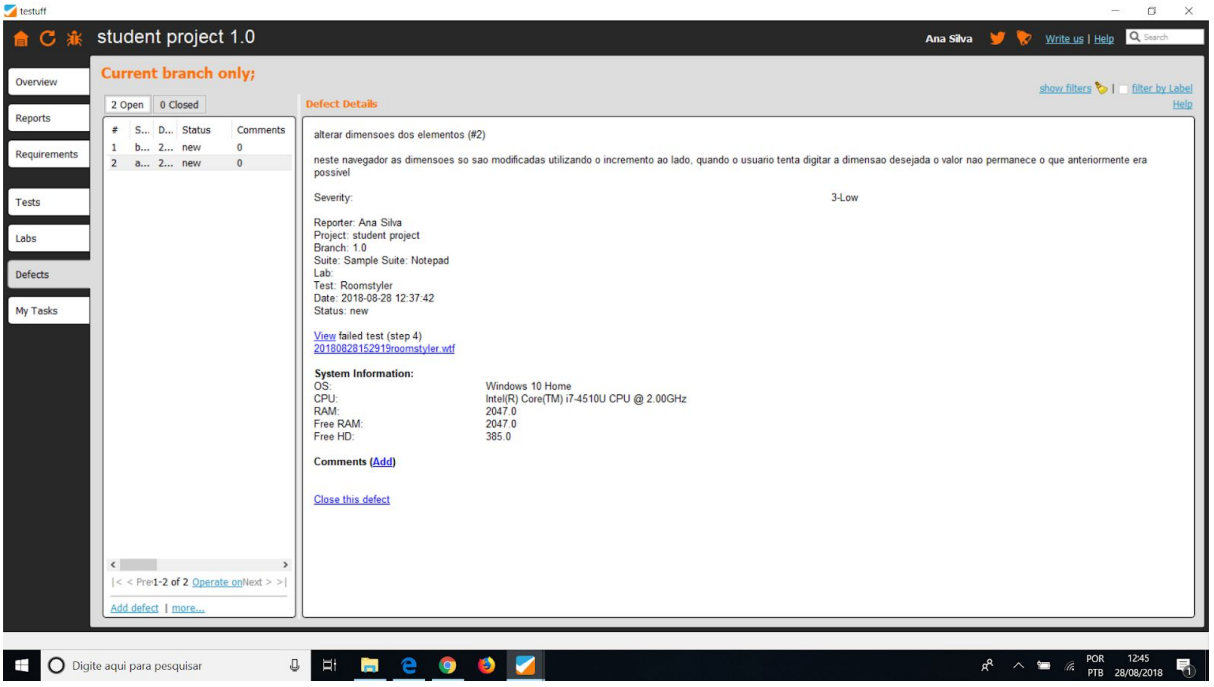

Figura 9: Teststuff - Resultado de teste feito no aplicativo desktop mostrado no próprio aplicativo.

Rapise

Ferramenta desenvolvida para auxiliar testes automatizados, foi citada por oferecer funcionalidades que também podem ser utilizadas em testes exploratórios e auxiliar na documentação e repetição dos testes. Inicialmente, o usuário escolhe o tipo de teste, entre testes para mobile, para aplicativos desktop, testes manuais e para web, podendo escolher o browser nos testes web. Também é possível escolher o tipo de linguagem de script, entre JavaScript e uma linguagem própria chamada Rapise Visual Language. Na figura 10, vemos o aplicativo no início dos testes, antes da inserção do passo a passo. Os testadores podem realizar seus testes livremente, a figura 12 mostra o aplicativo durante a execução, enquanto o Rapise captura as ações dos testador e as transforma no passo a passo do teste automatizado, figura 11. A ferramenta refaz o teste seguindo as instruções gravadas e fornece o resultado dessa sessão de testes.

A interface inicial da ferramenta antes e após a inserção do passo a passo :

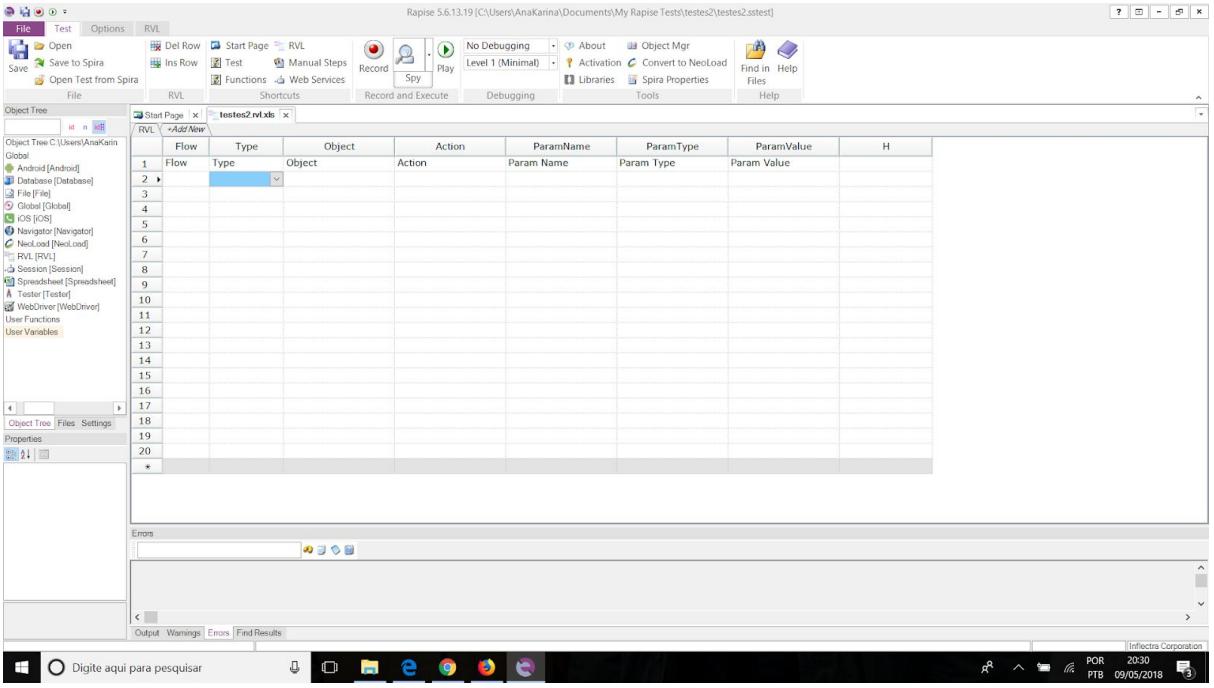

Figura 10: Rapise - Antes da inserção do passo a passo.

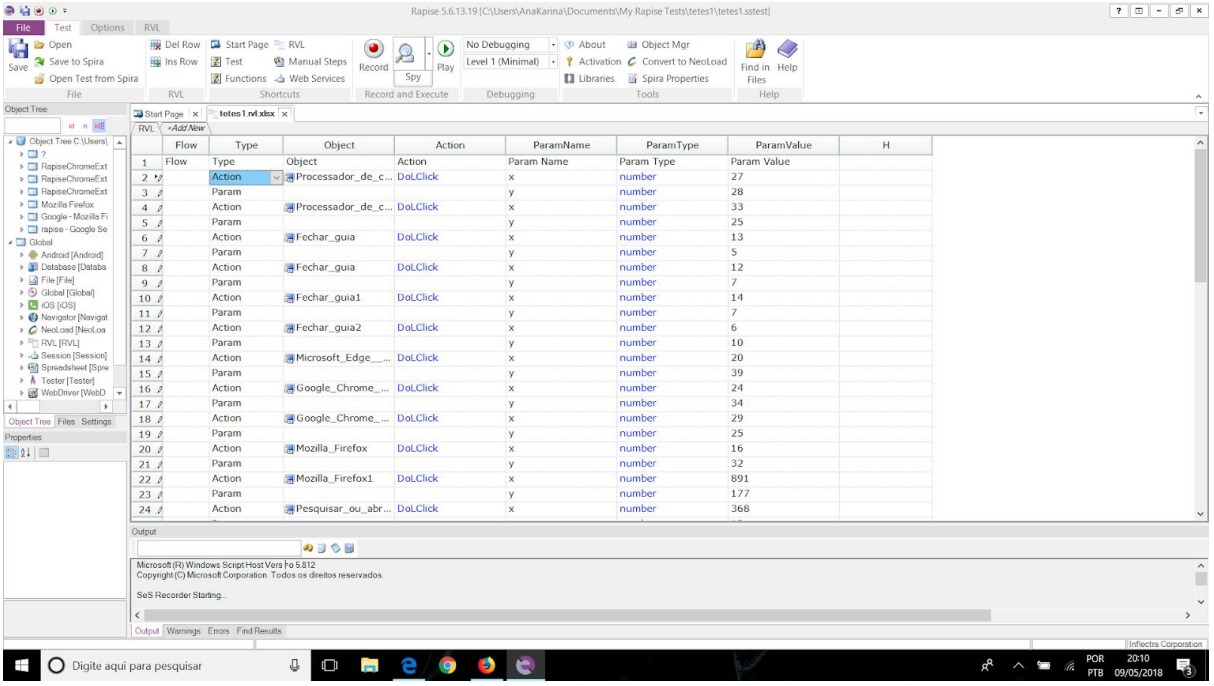

Figura 11: Rapise - Após a inserção do passo a passo.

Durante os testes apenas uma pequena interface/aba fica visível onde o passo a passo do teste é gravado, como é visto na figura abaixo.

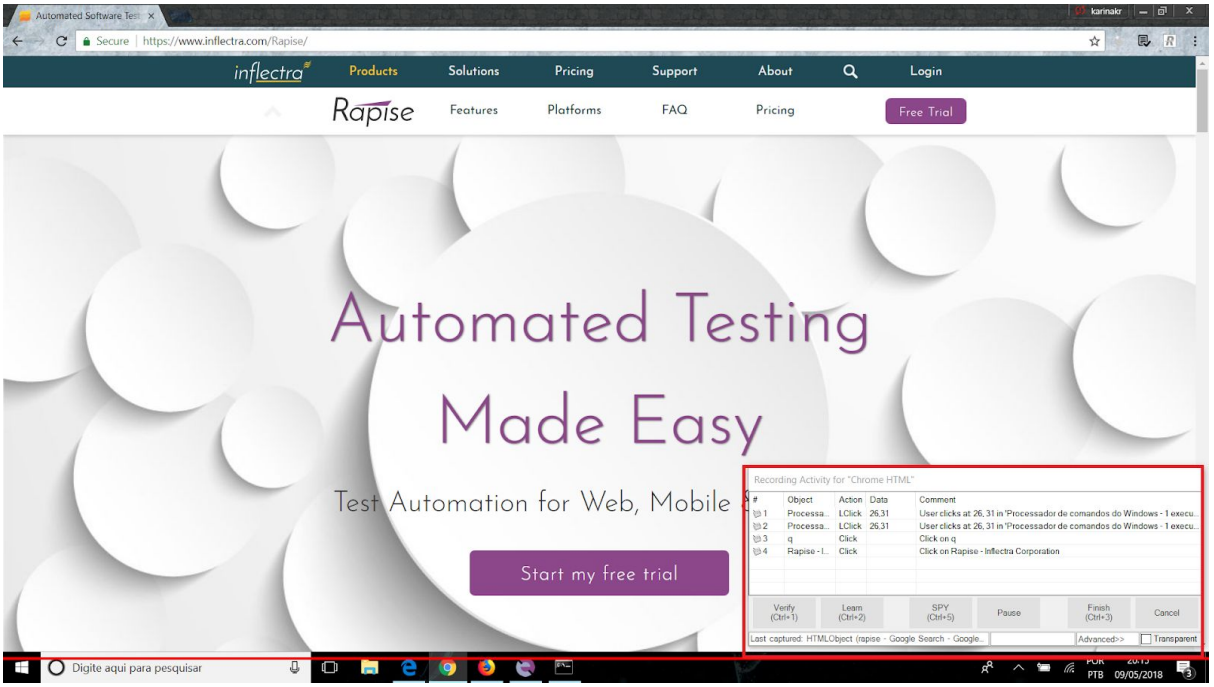

Figura 12 Rapise - Interface visível durante os testes.

Testpad

Aplicação web cuja principal funcionalidade é prover uma maneira rápida e eficiente de escrever planos de testes organizando uma checklist, figura 13, que pode ser usada para descrever o passo a passo dos testes ou, no caso do teste exploratório baseado em sessão, pode ser usada para organizar a missão da sessão com descrições breves do que deve ser testado. Durante a execução do teste o usuário pode fazer anotações pertinentes a cada um dos casos de teste/itens da missão seja na forma de relatos de bugs, notas, anexar arquivos a cada sessão de testes, a figura 14 mostra o formulário de relato de bugs, e cada plano pode ser testado várias vezes, como vemos na figura 13. A figura 15 mostra o aplicativo após a execução da primeira das três sessões criadas. Como a ferramenta não é voltada ao SBTM, não possui um meio de cronometrar o tempo da sessão. O aplicativo também pode ser usado em plataforma mobile.

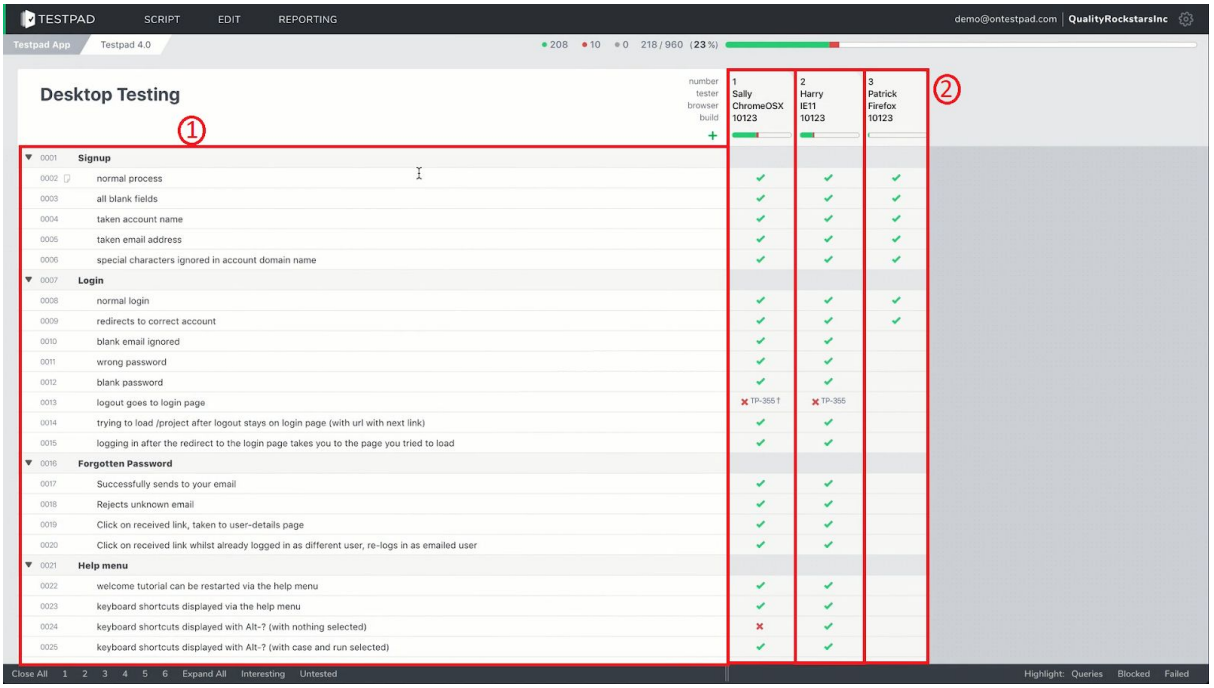

Figura 13: Testpad - Tela contendo testes ou missões a serem testadas pelo usuário. 1- testes/missões 2 - Sessões ou instâncias do mesmo teste, podem ter sido realizadas por mais de um testador. Mostra o resultado de cada teste.

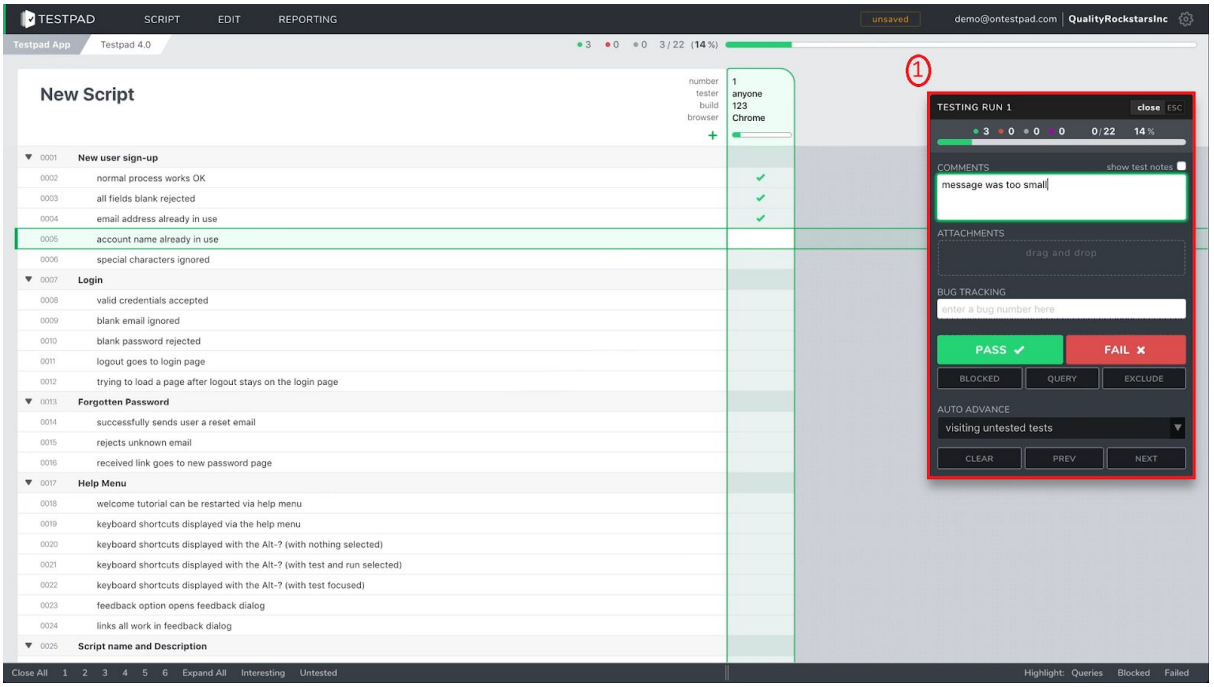

Figura 14 - Testpad - Passo a passo dos testes/Charter das missões e à direita (1) temos o teste que está sendo executado, relatório de bugs.

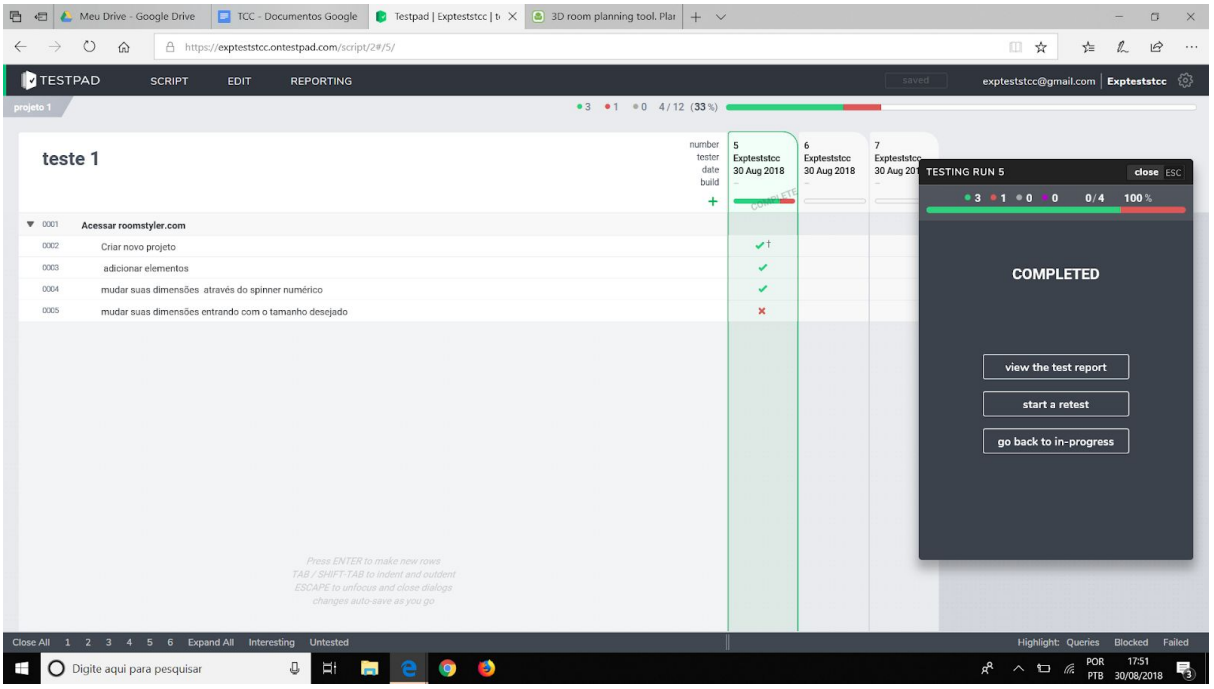

Figura 15 - Testpad - Teste completo.

Bug Magnet

Extensão para Chrome e Firefox, utilizado para testes de aplicações web. Funciona como fonte de dados que possam ser utilizados durante os testes, como por exemplo valores

limites, dados que possam causar algum erro na aplicação, dados para testar a segurança do aplicativo web através de sql injection. Bastante simples e ùtil, depois de instalada a extensão, o usuário clica com o botão direito no campo que deseja preencher, no caso da figura 16 é o campo de busca, e o símbolo do Bug magnet deve aparecer entre as opções, daí o usuário escolhe que tipo de dados deseja injetar e checar a resposta do site se ele tiver algum tipo de validação de dados. Apesar de não colocar em prática os conceitos de testes exploratórios, auxilia em sua realização principalmente quando o testador não possui experiência no domínio da aplicação sendo testada e necessita injetar dados durante a sessão agilizando assim a realização do teste.

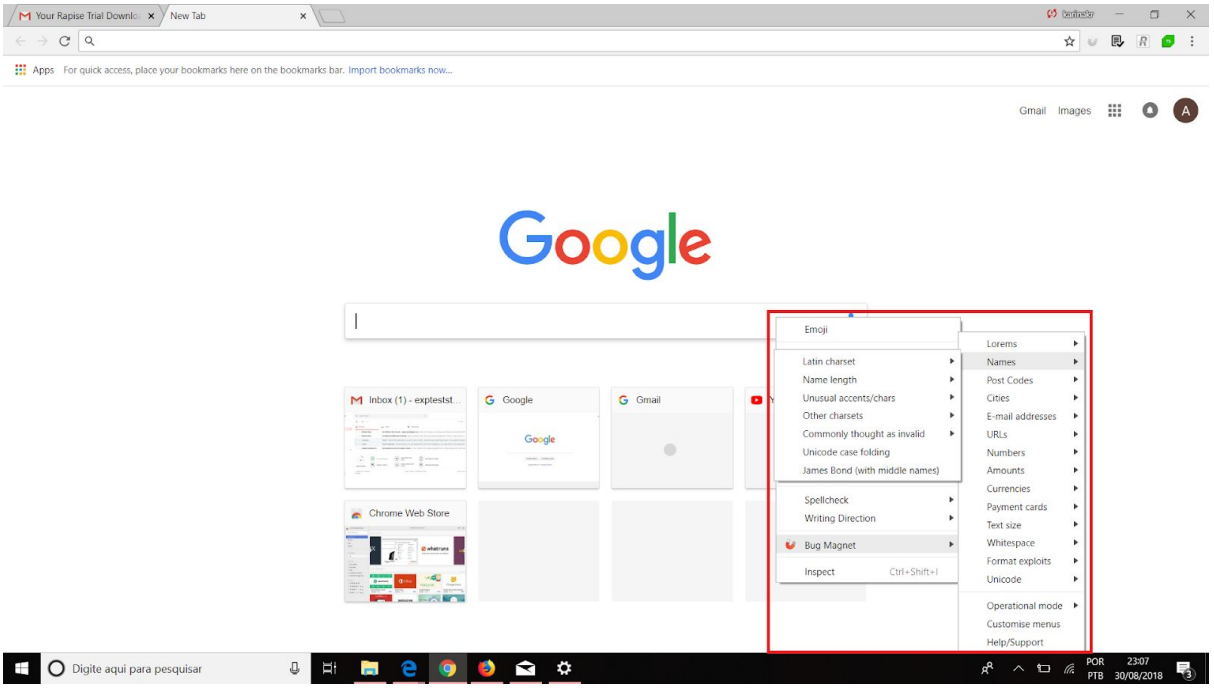

Figura 16: Bug Magnet - Interface mostrando parte dos tipos de dados fornecidos pela ferramenta para a inserção

TestRail

Ferramenta web e desktop, auxilia no gerenciamento e realização de testes. O usuário cria projetos, figura 18, e casos de testes que podem ser executados em sessões de testes. Para cada caso de teste são especificados o template: Exploratory session, na figura 20, Test Case (Steps) ou test case (text); o tipo (aceitação, acessibilidade, performance, regressão, etc); a prioridade(crítica, alta, média, baixa). Para o template Test cases (text) o usuário preenche

uma lista de pré-condições, passos, resultados esperados e descreve a missão e os objetivos, para o template Test cases (steps) o usuário fornece as pré-condições e passos, já para o template Exploratory session descreve a missão e objetivos, o usuário também tem a opção de importar/exportar casos de testes, figura 20.

Após a especificação do caso de teste/missão o usuário especifica o Test Run que deve ter nome, milestone, testador que realizará o teste, descrição e escolhe se todos os casos de teste serão aplicados ou apenas parte deles, visto na figura 23. Durante a execução do teste (função Test run) o usuário pode cronometrar a duração dos testes, documentar bugs e anotar os resultados, anexar imagens, tabelas ou importar arquivos e anexá-los à execução. Os resultados indicam a percentagem dos casos que passaram, falharam, foram bloqueados e que necessitam de reteste, figura 24. O mesmo caso de teste pode ser executado várias vezes, cada Test run possui um status e é possível monitorar seu progresso, esse progresso é atualizado com o resultado dos testes a cada execução, atividade e quantidade de defeitos.

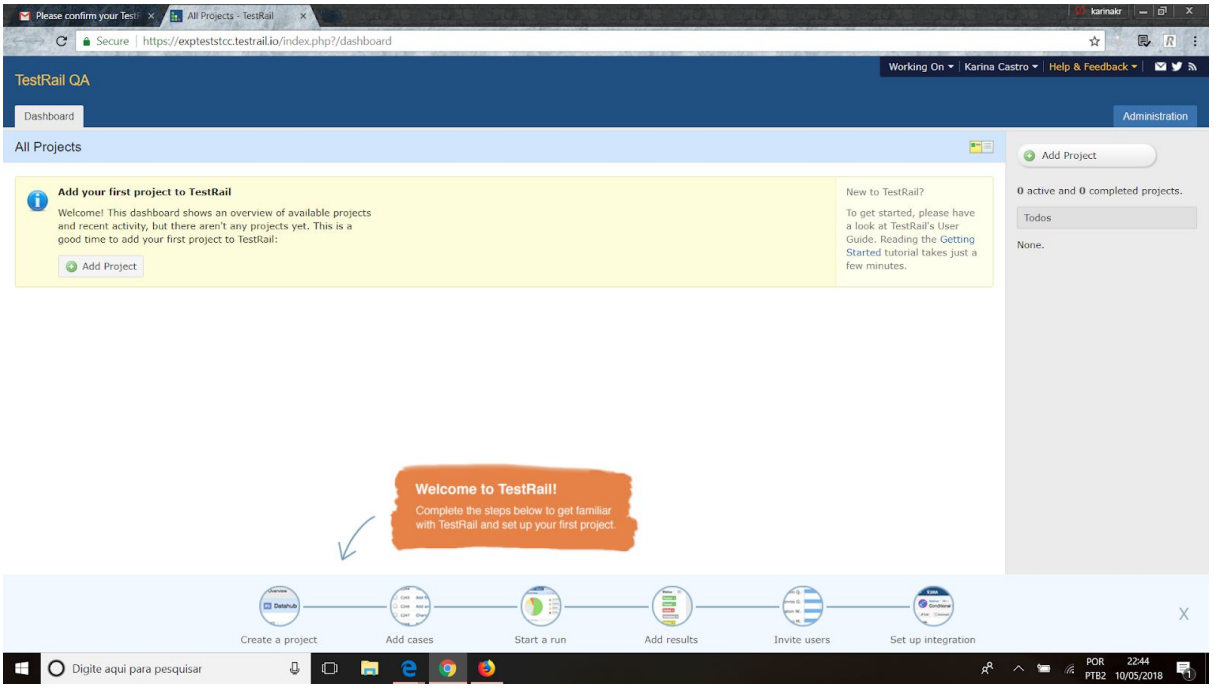

Figura 17: Testrail - Tela inicial.

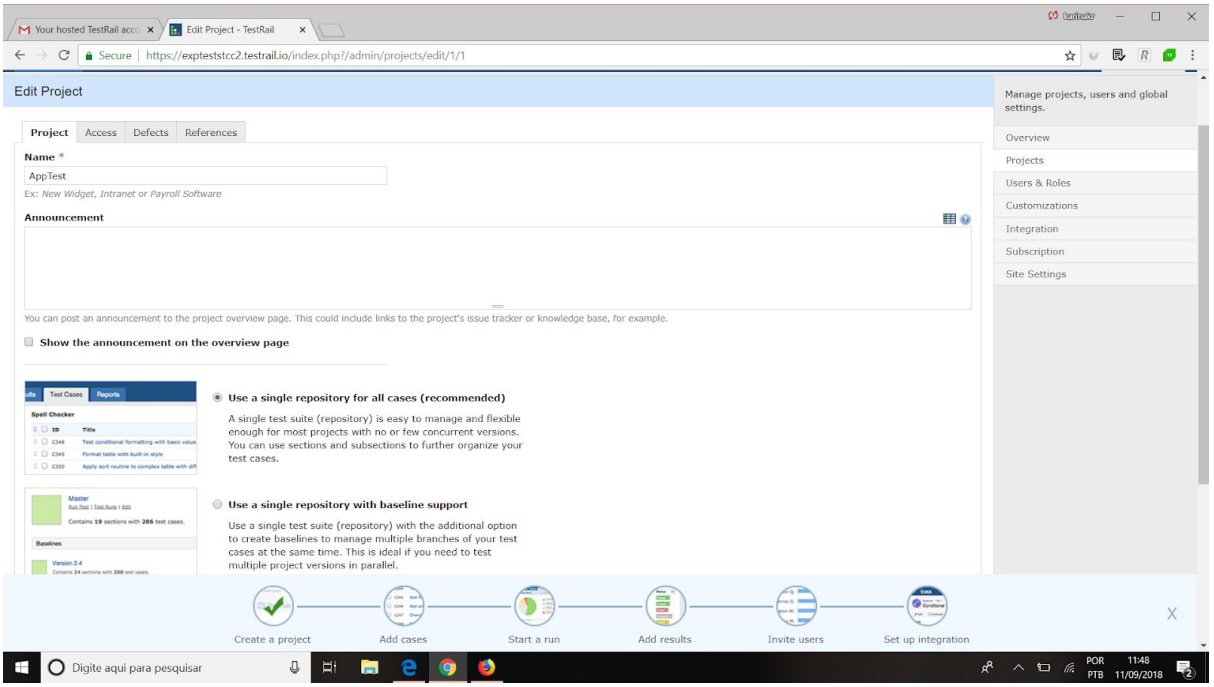

Figura 18: Testrail - Adicionar / Editar projeto de testes.

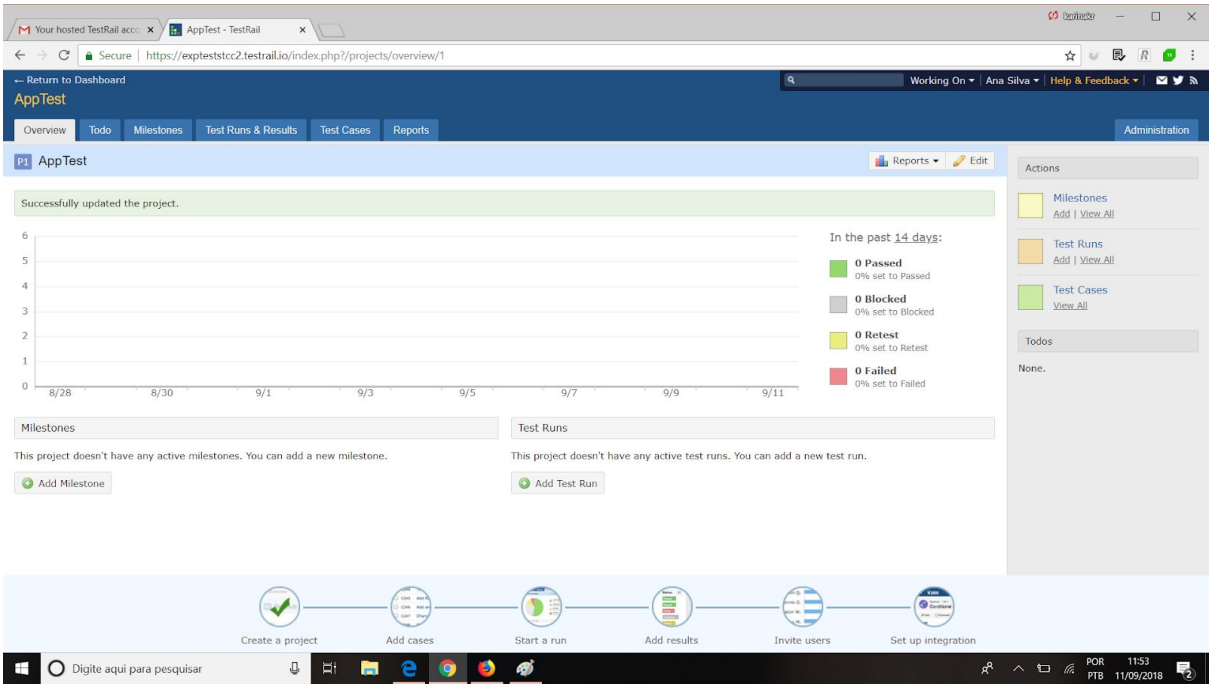

Figura 19: Testrail - Overview do projeto .

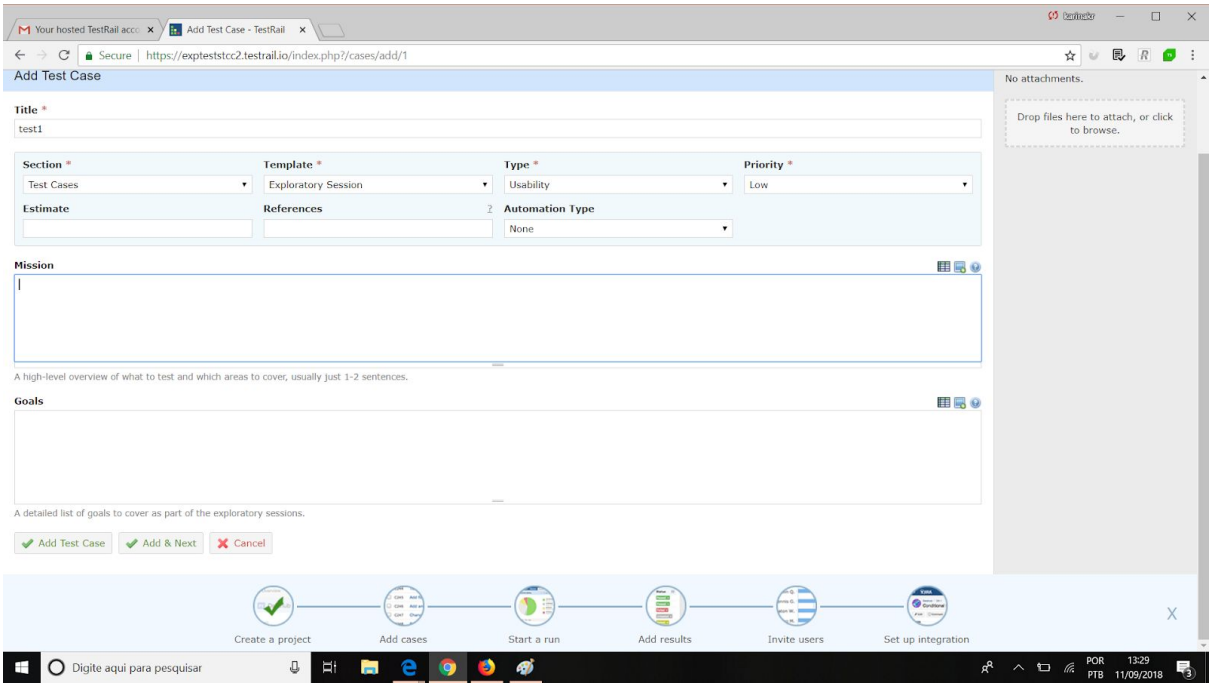

Figura 20: Testrail - Adicionar caso de teste - template exploratory session.

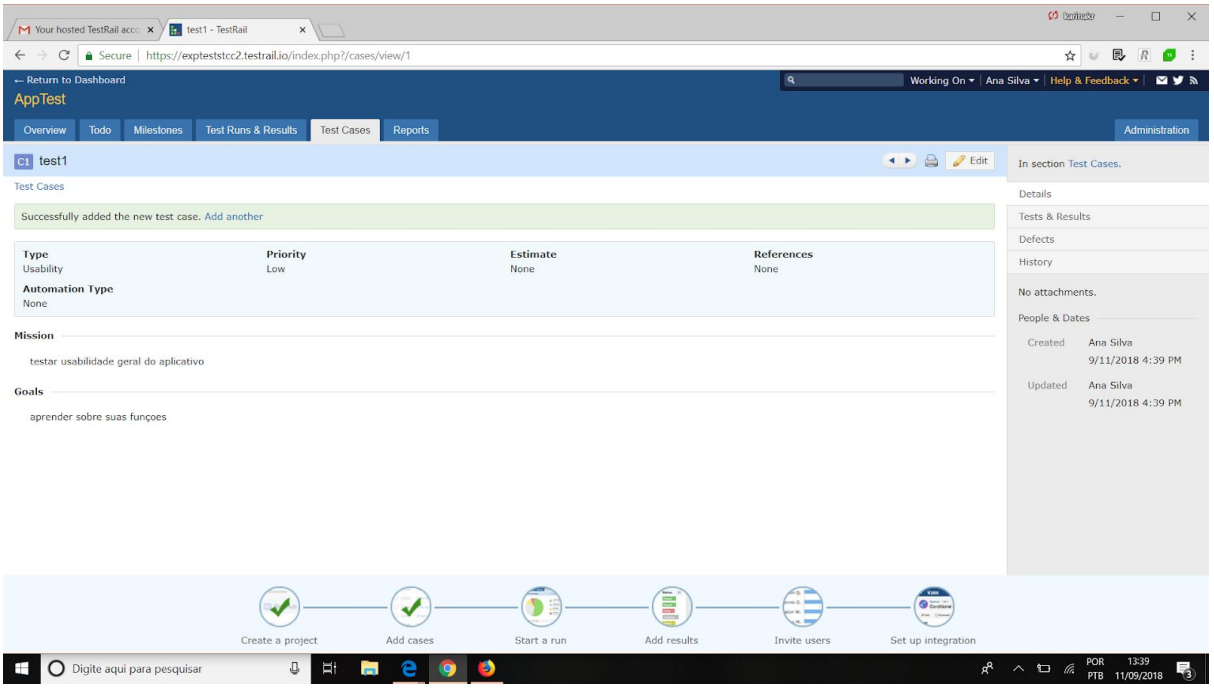

Figura 21: Testrail - Caso de teste adicionado.

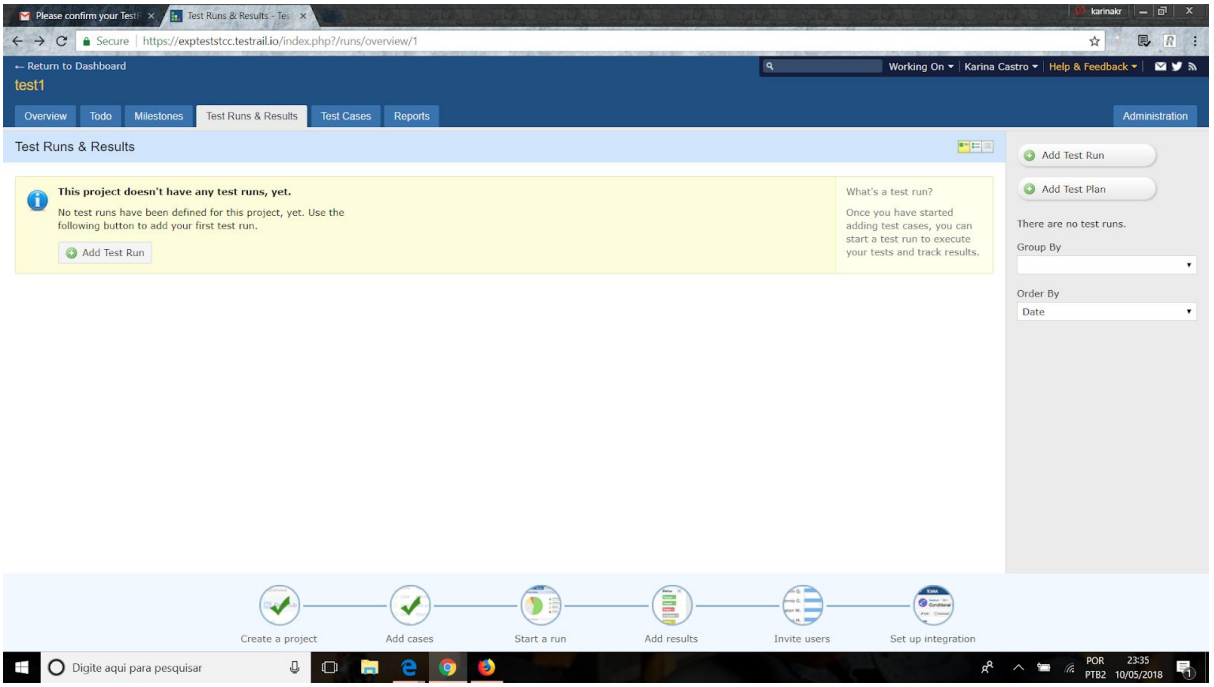

Figura 22: Testrail - Adicionar test run I

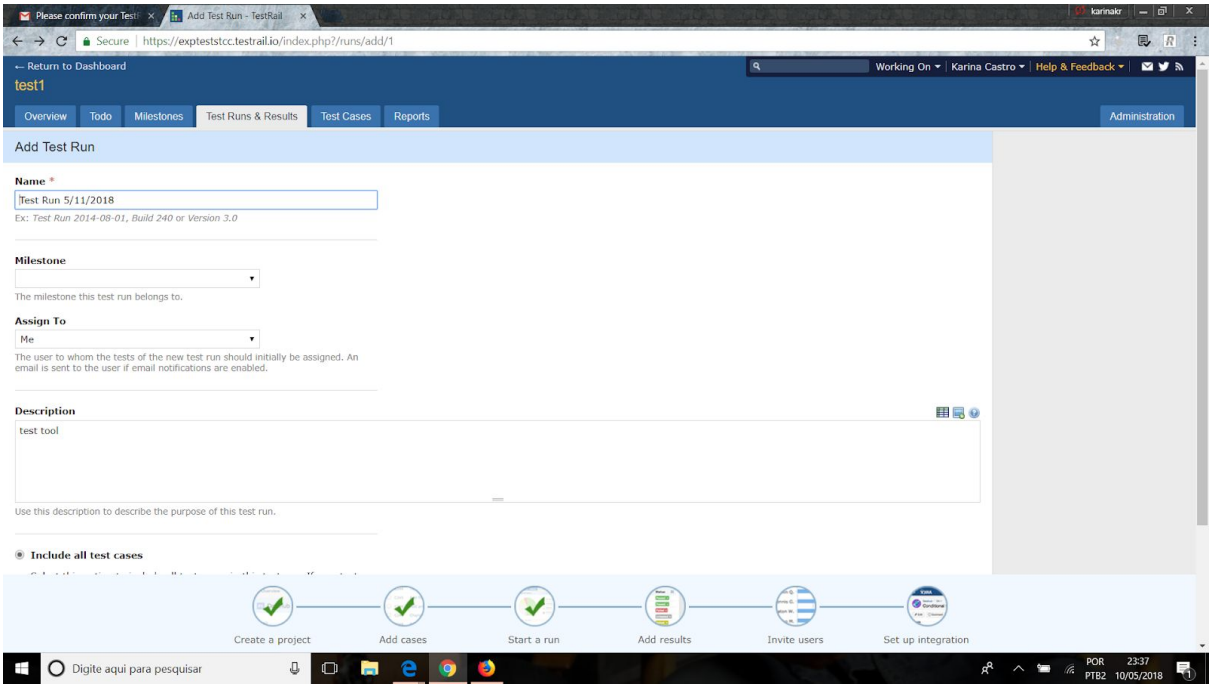

Figura 23: Testrail - Adicionar de test run II- Configuração

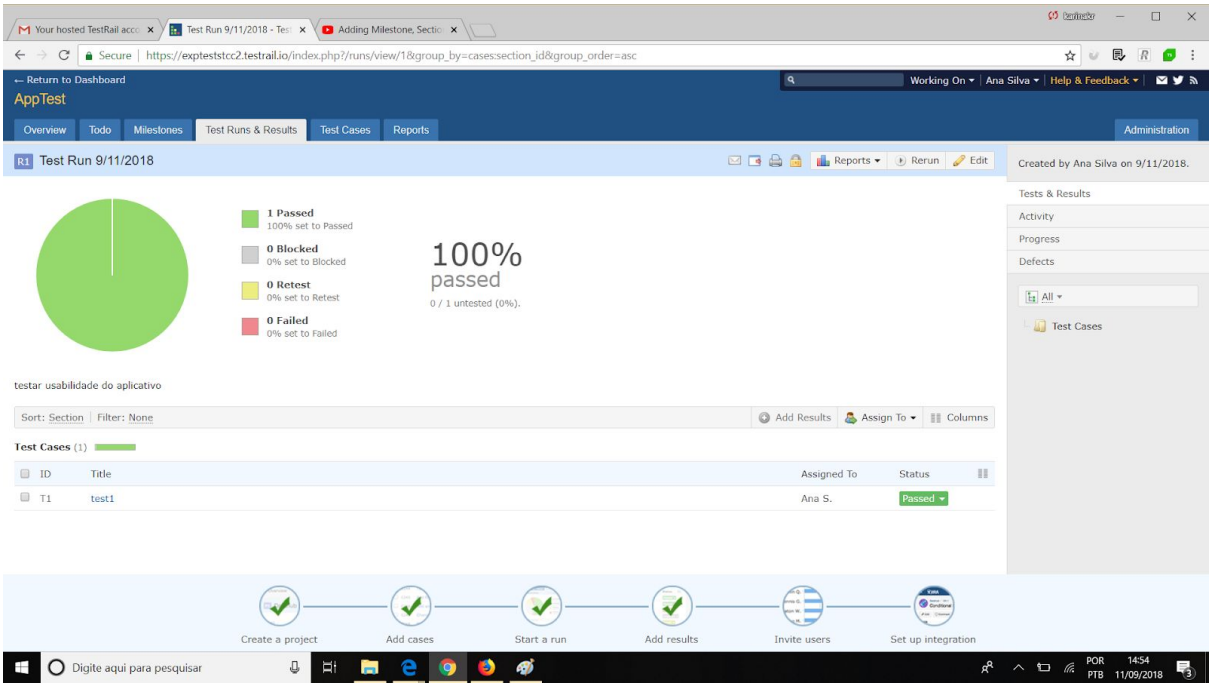

Figura 24: Testrail -Execução e Resultados dos testes após a execução dos testes

Wink

Ferramenta desenvolvida para criação de tutoriais e apresentações, que possui algumas funções que podem ser utilizadas por testadores para auxiliar no relato de erros como por exemplo: captura de telas onde o usuário escolhe se quer capturar a janela do navegador, a tela ou apenas uma região dela, mostrado na figura 25, o usuário também pode utilizar alguns recursos para fazer anotações textuais nas capturas de tela, figura 27, gravação de vídeos, de áudios, acrescentar botões de navegação entre os screenshots, escolher a ordem das imagens capturadas, visto na figura 26, entre outros recursos. O projeto pode ser salvo em PDF, HTML, PostScript ou como uma animação em Flash.

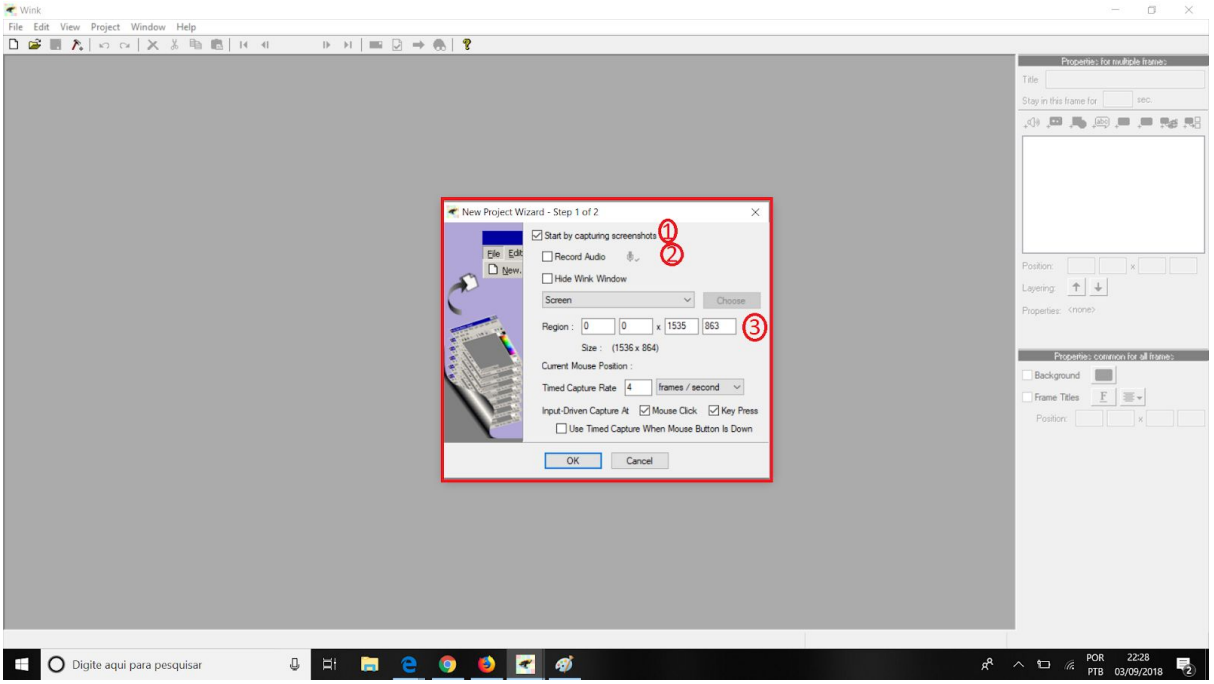

Figura 25: Wink - Tela inicial durante a configuração do teste - escolha do tipo de captura. 1 - Screenshots; 2 - Áudio; 3 - Podendo capturar a tela inteira ou apenas uma região, o usuário entra com as coordenadas.

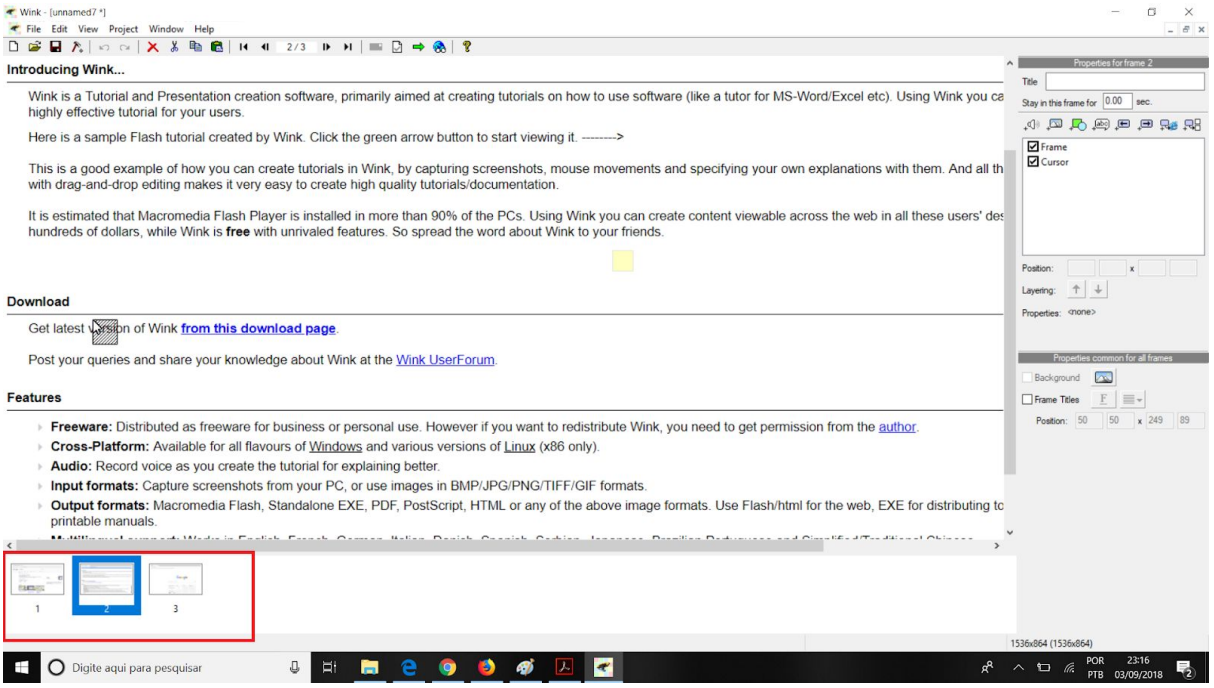

Figura 26: Wink - Screenshots.

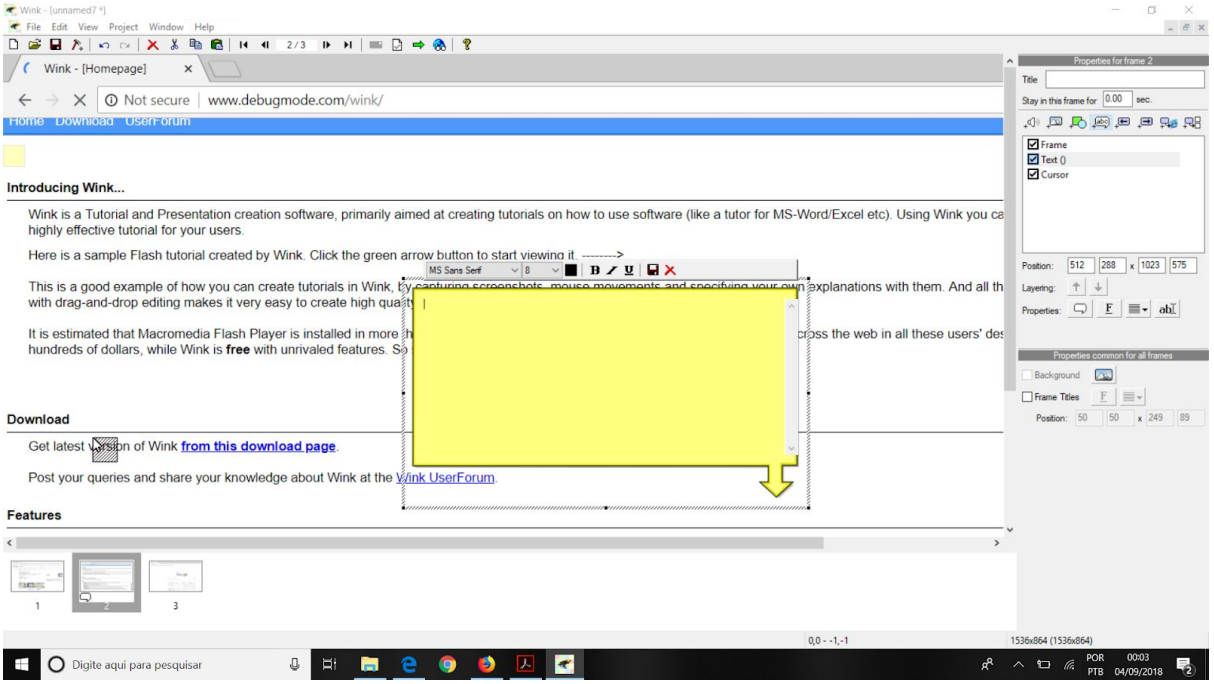

Figura 27: Wink - Report Textual

Jira + Capture for JIRA (Zephyr Capture for JIRA)

Jira é uma ferramenta web e desktop desenvolvida para auxiliar no rastreamento de defeitos. Fornece rastreamento de bugs, rastreamento de problemas e funções de gerenciamento de projetos. O usuário pode criar e relatar problemas especificando o passo a passo de como ele ocorreu, prioridade, anexar arquivos ao problema. Entretanto, esta análise tem como foco principal o Capture for Jira, ou como ela se chama atualmente Zephyr Capture for Jira. O capture for Jira é uma extensão do Jira que pode ser adicionada ao browser do usuário, possui suporte para vários navegadores modernos. O usuário pode utilizá-lo enquanto explora sites, nas figuras 28 e 29 vemos a principal interface do aplicativo. Possui recursos que permite fazer anotações sobre os bugs encontrados, na forma de relatos textuais e de vários tipos de anotações na imagem capturada , figura 28 ; capturar tela e anexar aos relatos de bugs, figura 29; oferece o recurso de criação sessões de testes, figuras 31 e 32, e convite a outros usuários à participar das sessões, figura 33. Além disso o aplicativo também fornece, no resultado do testes, detalhes sobre o dispositivo/ambiente no qual o teste foi realizado, visto na figura 30.

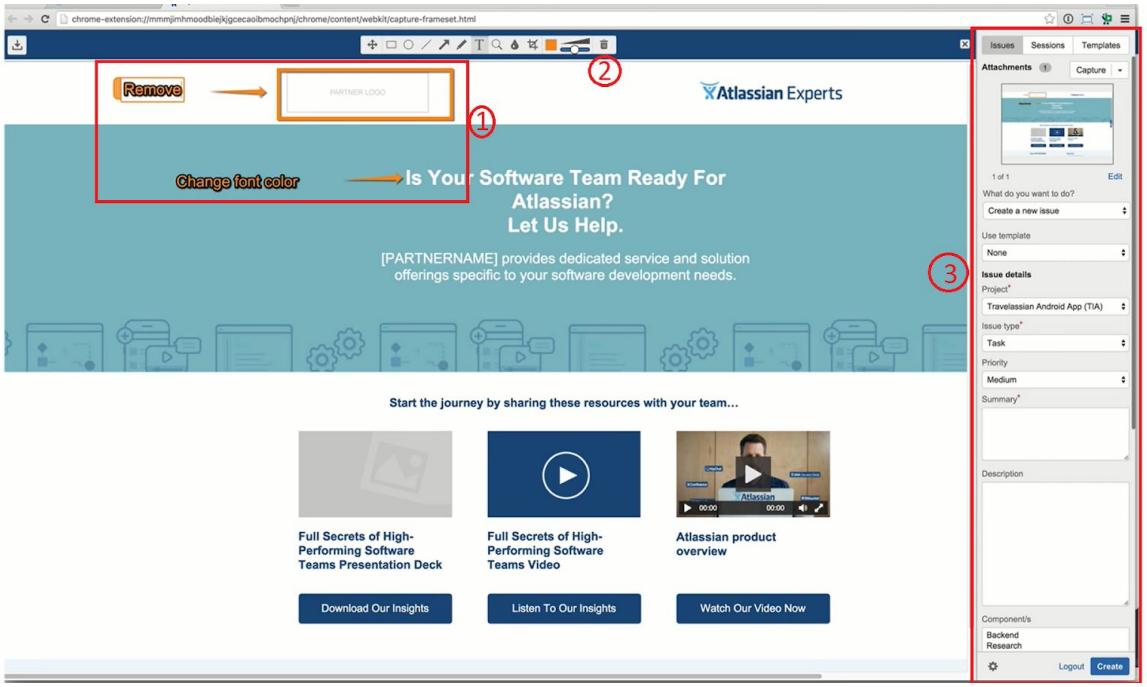

Figura 28: Zephyr Capture for Jira - Interface durante os testes. Anotações. 1- Anotações que podem ser feitas na imagem capturada, 2 - Tipos de anotações: texto, setas, caixas, 3 - Interface do Capture.

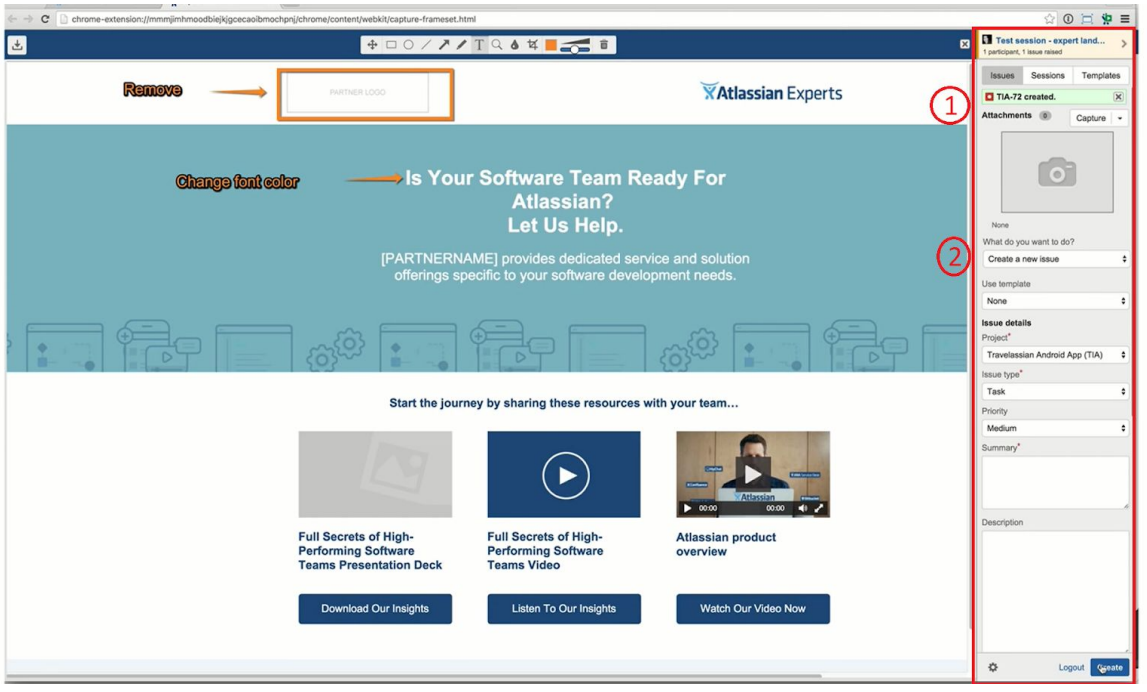

Figura 29: Zephyr Capture for Jira - Interface durante o teste. 1- Anexar imagem/arquivo 2 - Detalhes do bug relatado: projeto, tipo, prioridade, resumo, descrição.

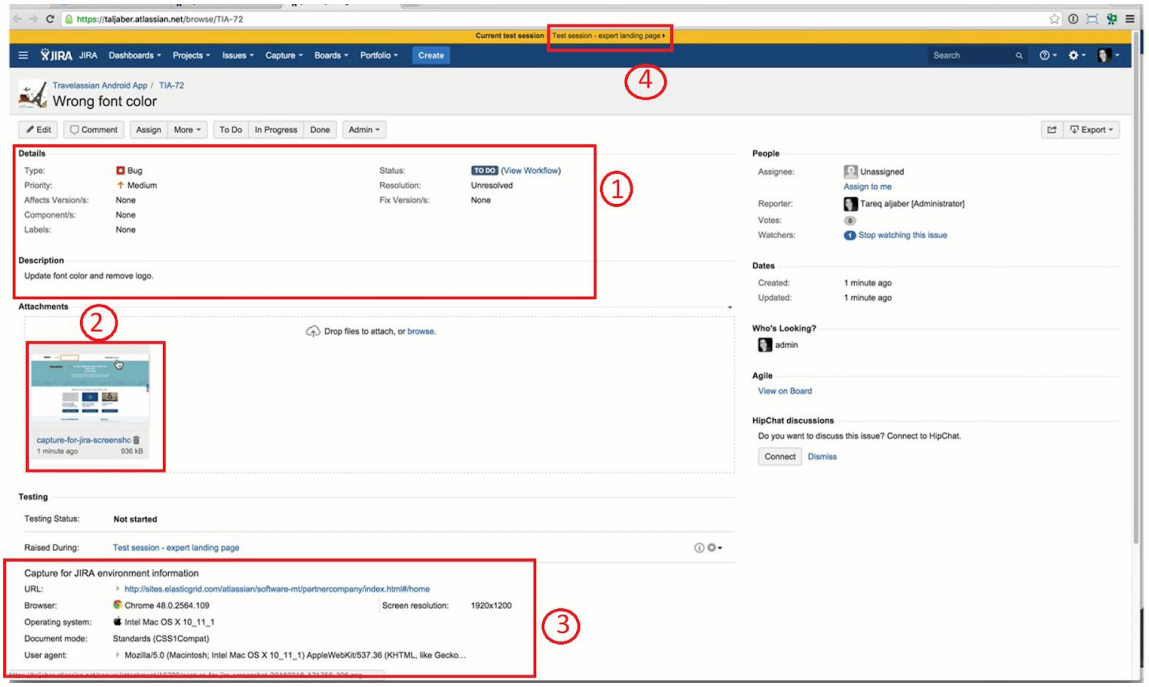

Figura 30: Zephyr Capture for Jira. Tela que mostra os detalhes do bug relatado 1 - Detalhes do bug relatado, 2 - Screenshot do bug onde clicando-se no bug o usuário maximiza a imagem 3 - Detalhes do ambiente, 4- Clicando-se em Test Session - expert landing page o usuário passa à tela onde é possível convidar outros usuários a participar da sessão.

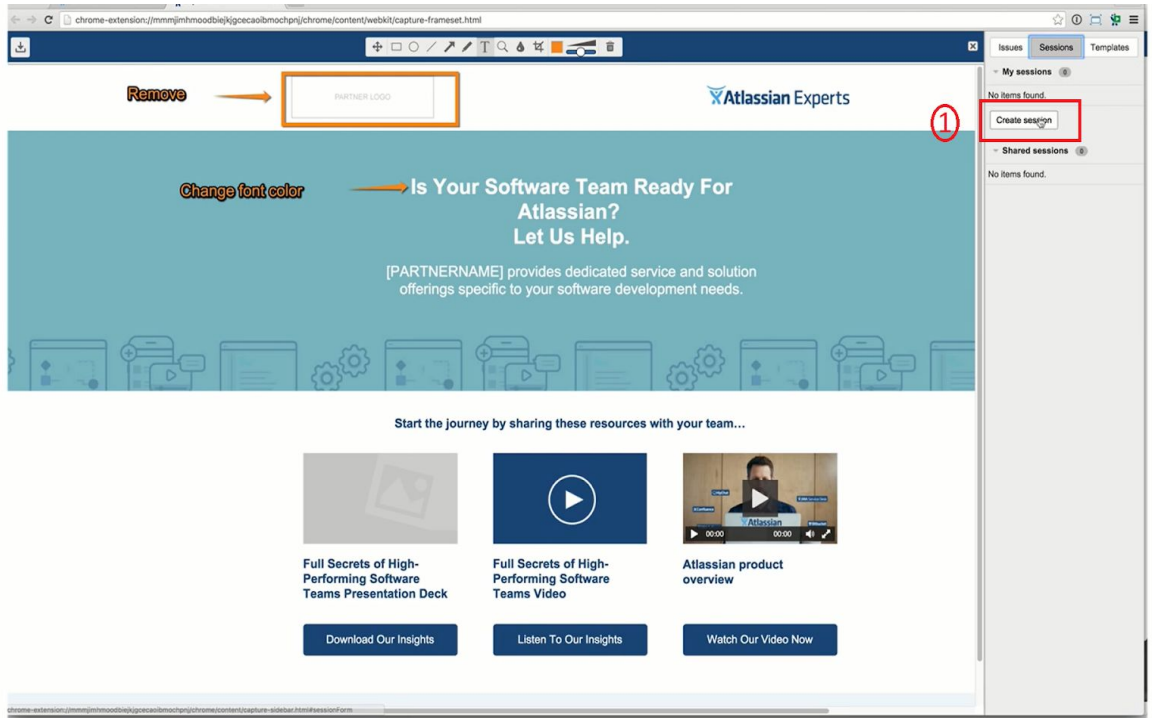

Figura 31: Zephyr Capture for Jira. Criar sessão.

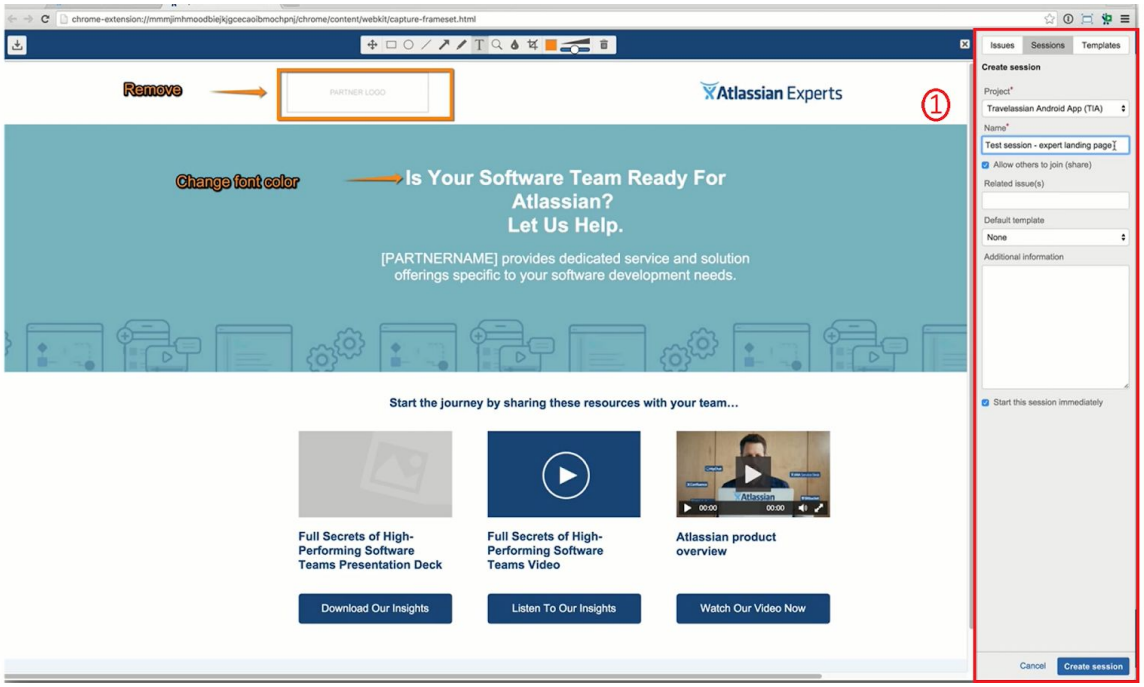

Figura 32: Zephyr Capture for Jira. 1- Detalhes da sessão sendo criada: Projeto, nome, problemas/bugs relacionados, template e informações adicionais.

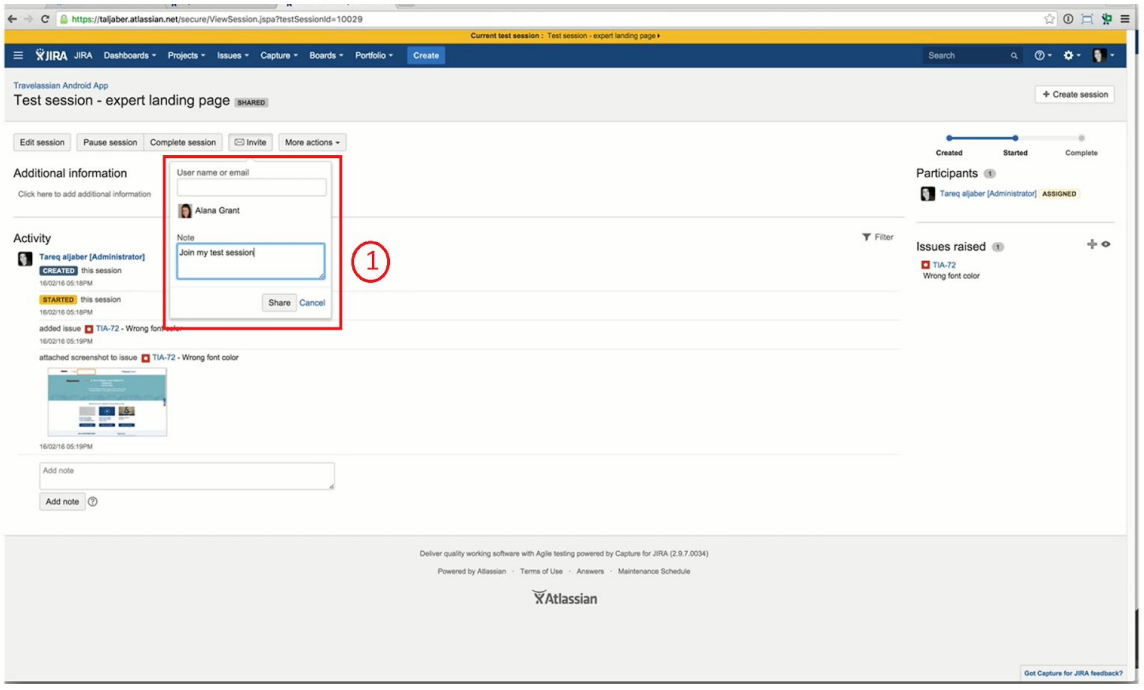

Figura 33: Zephyr Capture for Jira 1 - Convidar usuário.

BB TestAssistant

Ferramenta cuja principal utilidade é permitir que o testador realize a exploração do sistema sendo testado sem interrupções e após terminado o teste o usuário pode rever o vídeo do teste e fazer suas anotações e relatório. O TestAssistant faz a gravação da tela enquanto o testador realiza o teste, é possível escolher entre gravar a tela, apenas uma janela ou uma região da tela, gravar o som do microfone, dos auto-falantes, ou a imagem da webcam.

Após o teste finalizado o usuário decide se deseja salvar ou descartar o vídeo gravado e se deseja que o aplicativo inclua informações sobre a configuração da máquina e arquivos de log, é possível ainda escolher se o vídeo será aberto para que as anotações sejam feitas imediatamente, se o vídeo será compartilhado via youtube ou flashback connect, ou se ele será exportado e neste caso o usuário escolhe o formato dentre os oferecidos.

É possível fazer screenshots do vídeo que podem ser exportadas pro editor de textos Word, fazer anotações no próprio vídeo, basta abrir o arquivo salvo após a gravação e a ferramenta abre a segunda interface, figura 36, que funciona como um editor de vídeo onde é possível fazer as anotações relatando o erro, no vídeo gravado as ações do testador são destacadas. A ferramenta possui dois modos, os quais podem ser acessados por ícones de atalho diferentes: o TestAssistant Expert Recorder, figura 35, e o TestAssistant Expert Player, figura 36.

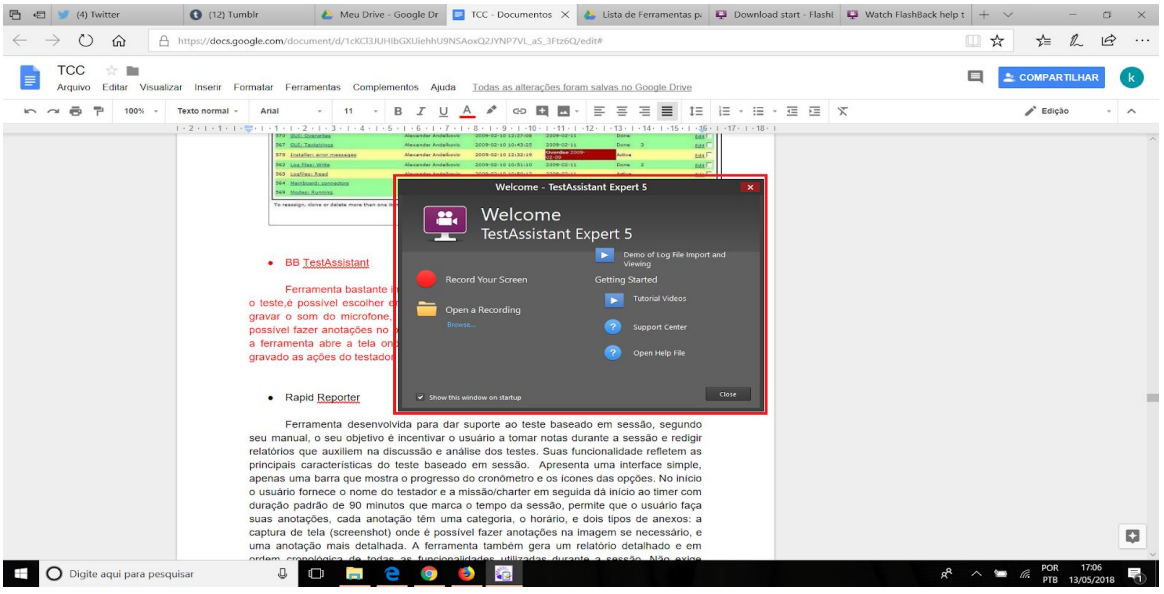

Figura 34: TestAssistant - Interface inicial, onde o usuário escolhe entre gravar a tela ou abrir uma gravação já existente.

Figura 35: TestAssistant - Expert Recorder

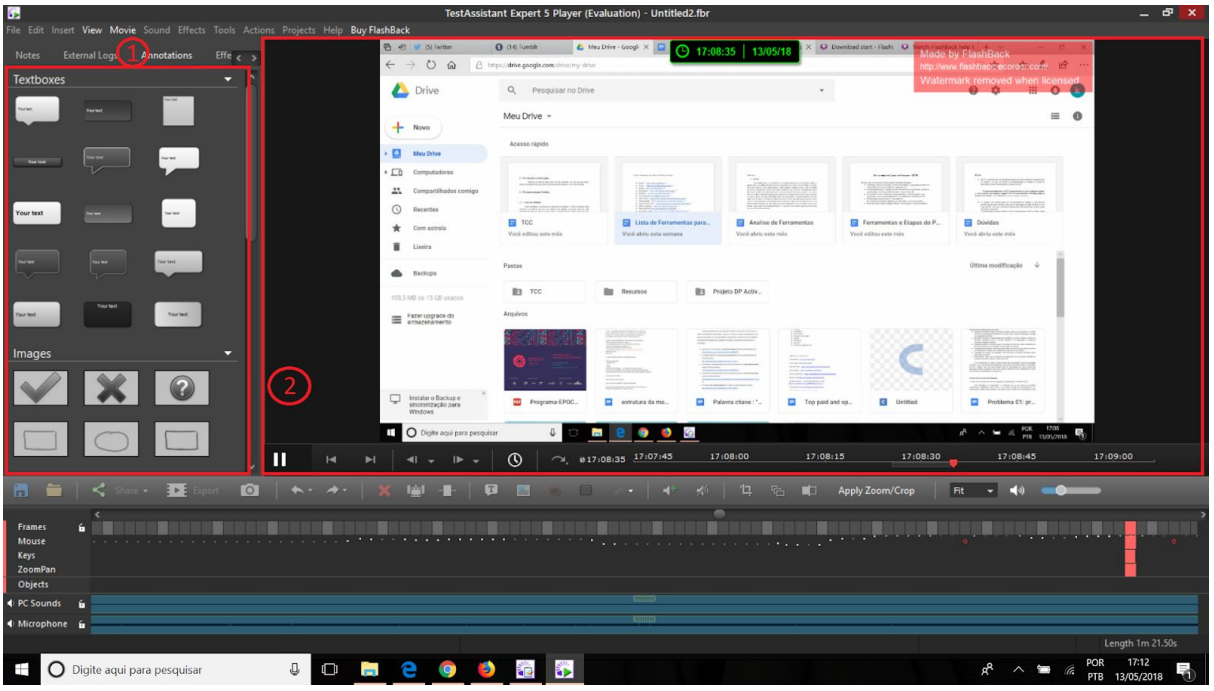

Figura 36: TestAssistant - Expert Player 1- Tipos de anotações que o usuário pode incluir no vídeo do teste, apenas clicando e arrastando o elemento. 2 - Editor e player do vídeo.

Rapid Reporter

Ferramenta desenvolvida para dar suporte ao teste baseado em sessão, segundo seu manual, o seu objetivo é incentivar o usuário a tomar notas durante a sessão e redigir relatórios que auxiliem na discussão e análise dos testes. Suas funcionalidade refletem as principais características do teste baseado em sessão. Apresenta uma interface simples, como vemos na figura 37, apenas uma barra que mostra o progresso do cronômetro e os ícones das

opções. No início o usuário fornece o nome do testador e a missão/charter em seguida dá início ao timer com duração padrão de 90 minutos que marca o tempo da sessão, permite que o usuário faça suas anotações, cada anotação têm uma categoria, o horário, e dois tipos de anexos: a captura de tela (screenshot) onde é possível fazer anotações na imagem se necessário, e uma anotação mais detalhada. A ferramenta também gera um relatório detalhado e em ordem cronológica de todas as funcionalidades utilizadas durante a sessão. Não exige instalação.

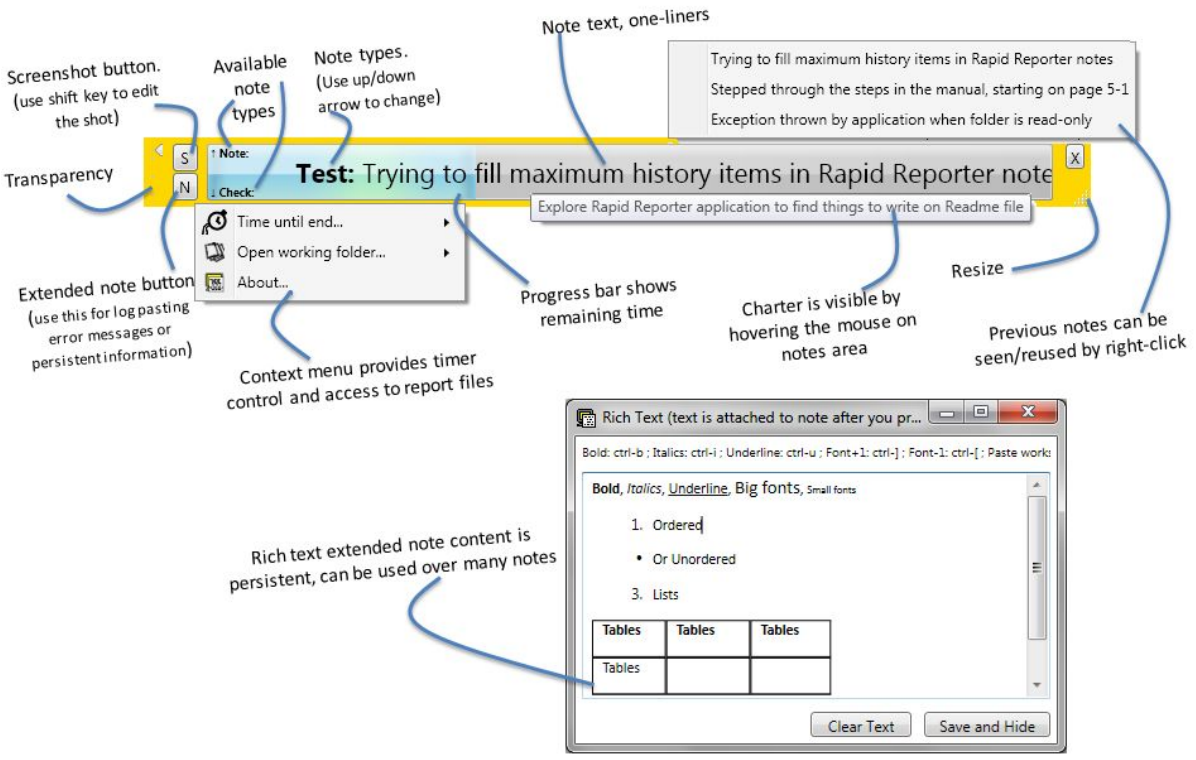

Figura 37: Rapid Reporter - Interface e Funções. Retirada do manual da ferramenta

Test Studio Explorer

Ferramenta desenvolvida para realização de testes automatizados, para dispositivos móveis, web e desktop, sendo necessário a instalação tanto da ferramenta desktop quanto da extensão do navegador para o funcionamento correto. A figura 38 mostra a escolha do tipo de testes entre as opções web/desktop, mobile e API. Durante a configuração do teste web o usuário informa qual a url do site em que o teste será iniciado, podendo navegar livremente

para outros sites durante o teste, na figura 39 podemos ver a interface da ferramenta durante os testes. A própria ferramenta abre o navegador na página indicada e grava as ações do testador durante a execução do teste capturando a interação do testador com todos os elementos da página, (exceto os elementos do navegador como por exemplo as setas de navegação, a utilização destas setas tem a falha do testes como consequência pois a ferramenta não tem como identificar a ação do usuário), ou do aplicativo, como cliques, preenchimento de campos, etc, e as transforma no passo a passo do teste automatizado. Na figura 40 vemos a ferramenta após o término da exploração do site, podemos notar a inserção do passo a passo dessa exploração.

O Test Studio refaz o teste seguindo as instruções gravadas e fornece o resultado dessa sessão de testes, oferece também uma visualização em *storyboard* do passo a passo do teste, figura 41. Fornece suporte para alguns navegadores (Firefox, Chrome e Internet Explorer) bem como aplicativos desktop e mobile. Em relação ao teste exploratório, é possível utilizar a ferramenta para auxiliar na documentação, relatório e repetição do teste se necessário durante a fase de testes de regressão e dessa forma ajuda a superar algumas adversidades encontradas nos testes exploratórios.

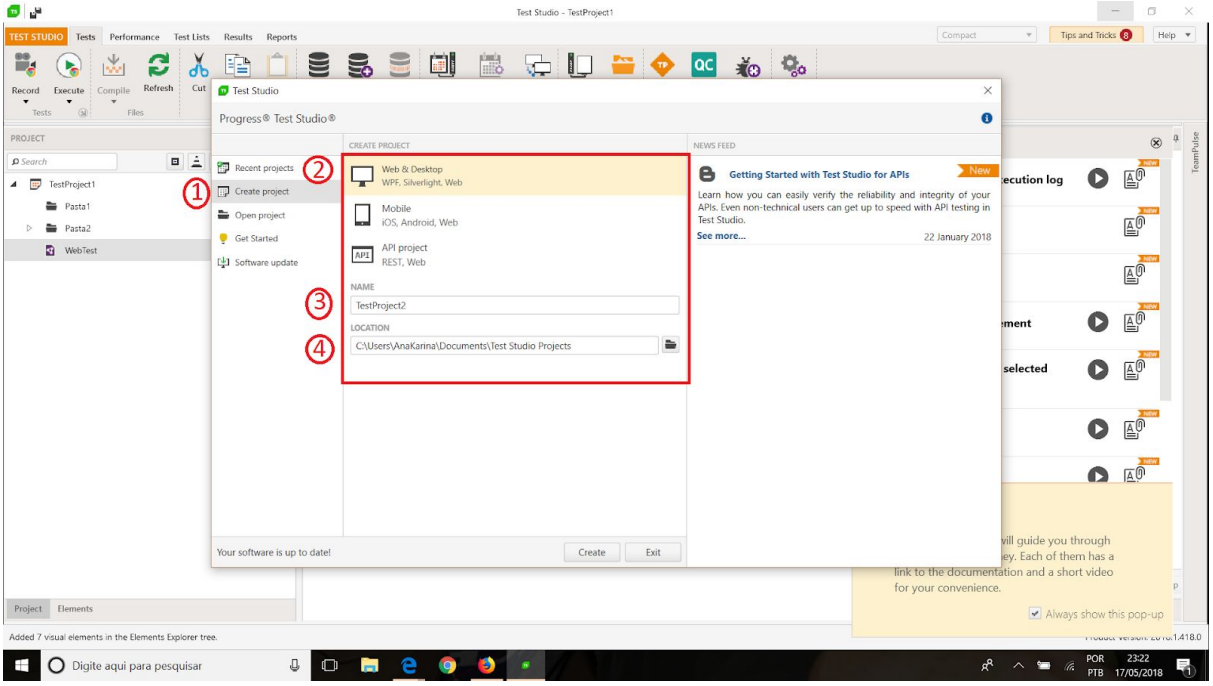

Figura 38: Test Studio - 1 Criação do projeto; 2 Escolha do tipo de testes: Web/Desktop, Mobile, API; 3 - Nome do projeto; 4 - Localização.

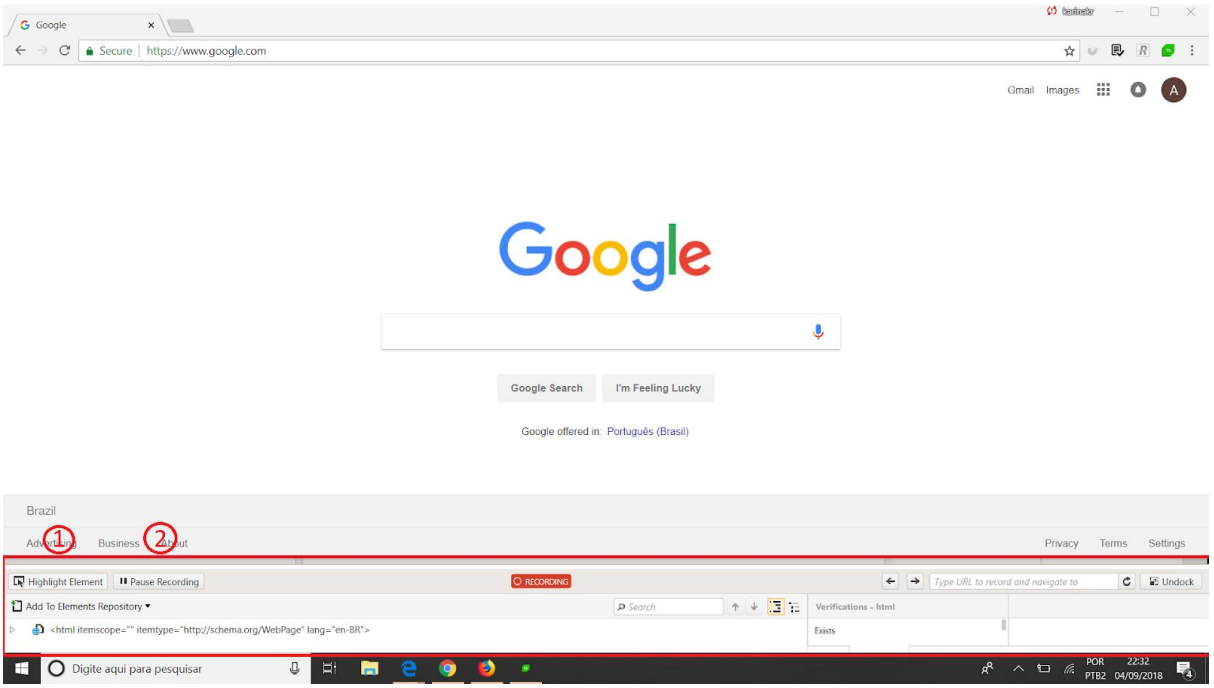

Figura 39: Test Studio - Interface durante os testes. Navegador durante os testes, onde a área ressaltada é a interface do aplicativo. 1- Realçar elemento, 2 - Pausar gravação do teste

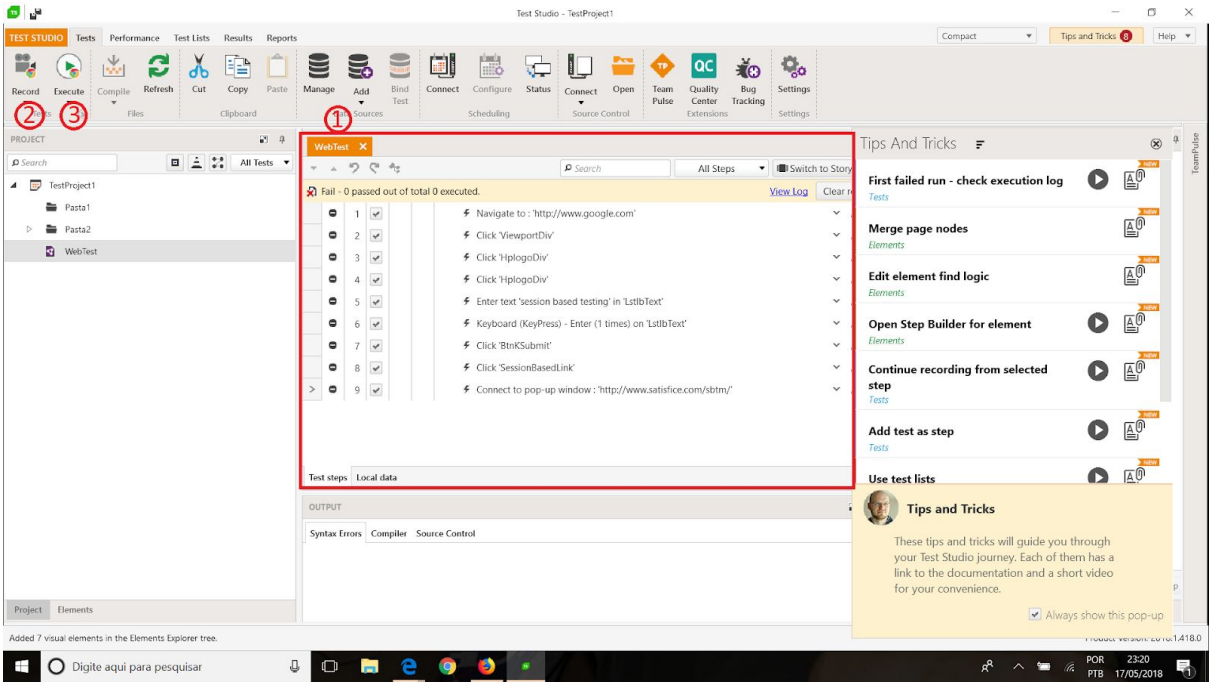

Figura 40: Test Studio 1- Após a gravação do passo a passo do teste; 2 - Gravar teste, 3 - Executar teste gravado.

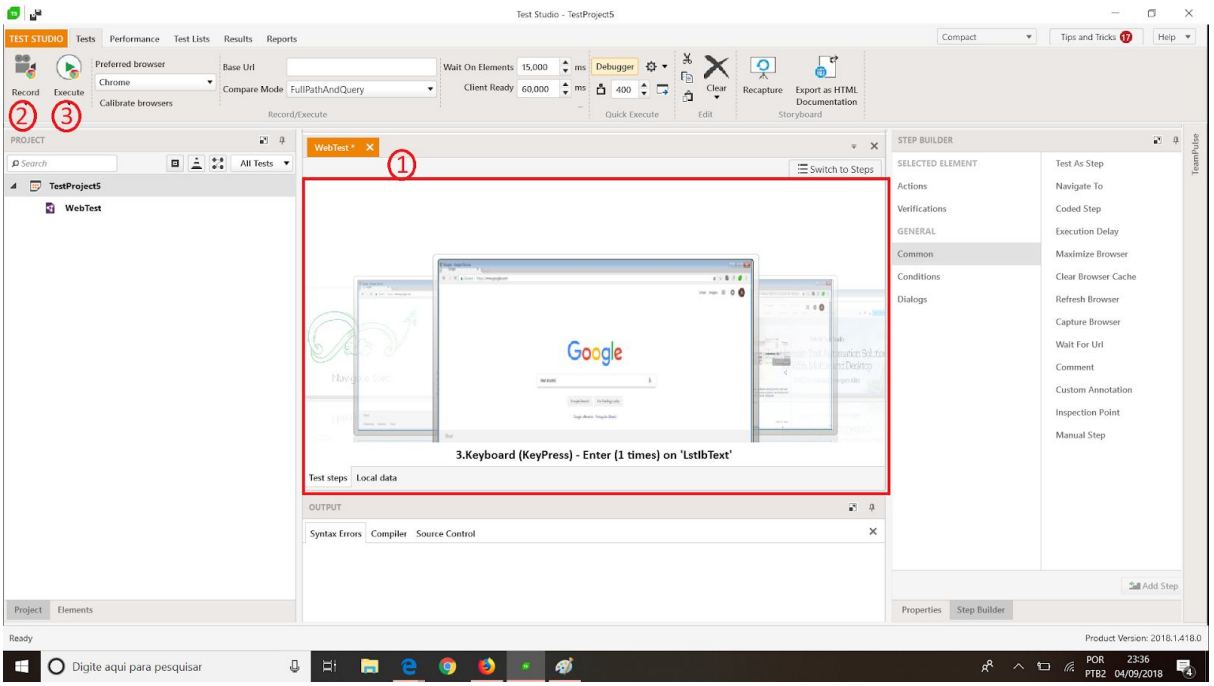

Figura 41: Test Studio - 1 - Storyboard, onde o usuário tem acesso ao passo a passo do teste em formato de storyboard, 2 - Gravar teste, 3 - Executar teste gravado

Exploratory Testing Chrome Extension

Extensão para Chrome para testes em aplicações web, ferramenta desenvolvida para auxiliar nos testes exploratórios. Apresenta uma interface simples e intuitiva, figura 42, onde o usuário é capaz de reportar bugs, figura 43, fazer anotações, anotar idéias/ oportunidades, questões clicando no botão correspondente e preenchendo a caixa de diálogo aberta logo abaixo, como também também fazer screenshots dos bugs encontrados. A ferramenta produz um relatório simples de entender que traz todas as anotações feitas pelo usuário, informações sobre a sessão: data/ hora, sistema operacional, versão do navegador, como vemos na figura 45. É possível ainda salvar e importar a sessão, bem como exportar sessões em arquivos CVS e HTML, acessar a URL do erro através de links fornecidos pela ferramenta ou realizar um reset na sessão, mostrado em detalhe na figura 44.

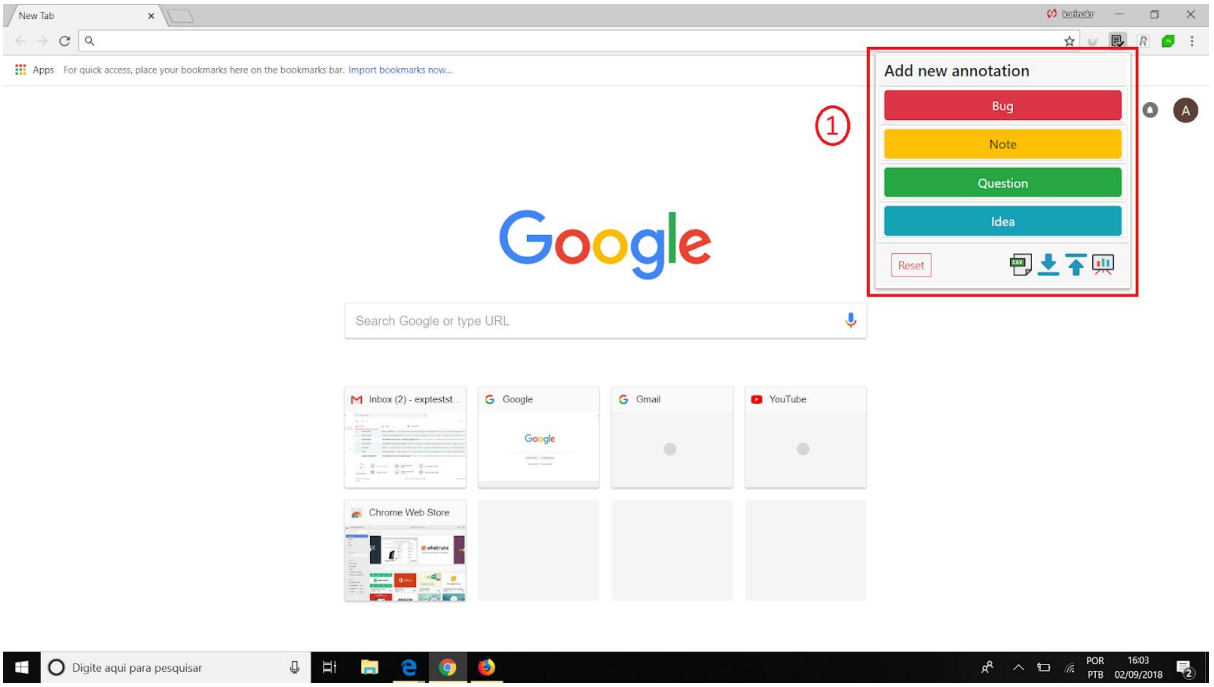

Figura 42: Exploratory Testing Chrome Extension - Interface inicial.

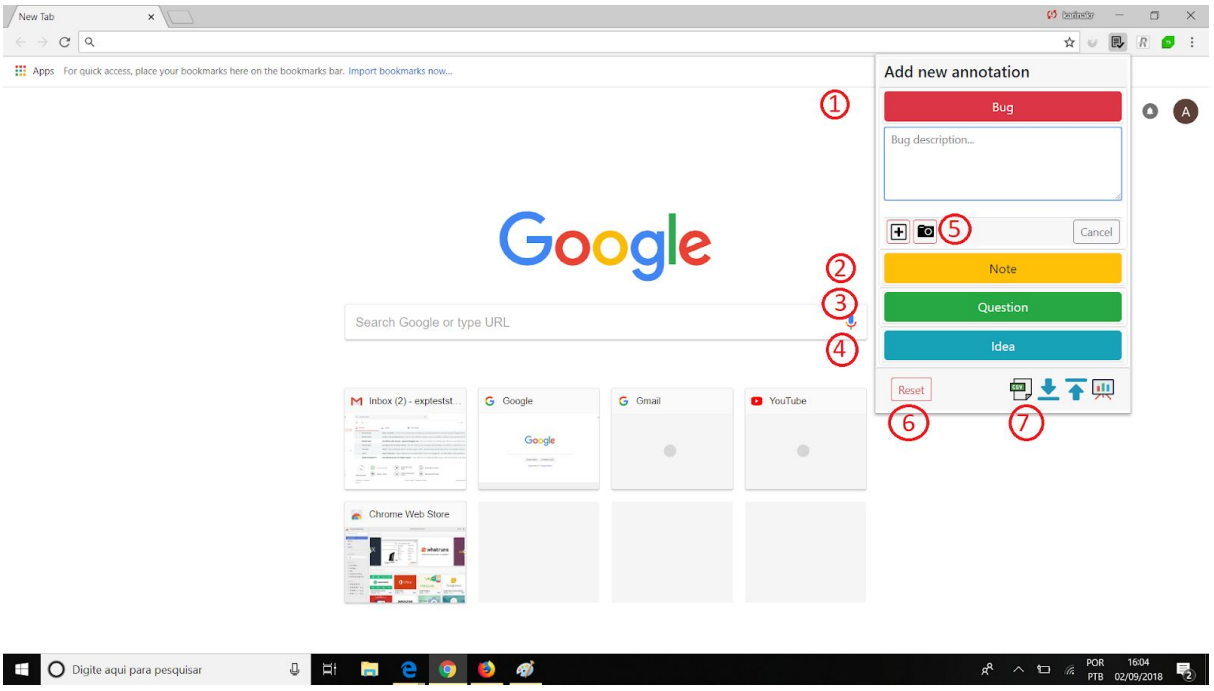

Figura 43: Exploratory Testing Chrome Extension - Reporte de bugs: 1 - Descrição do bug, 2- Anotações, 3 - Questionamentos, 4 - Ideias, 5- Screenshot, 6 - Reset da sessão, 7 - Exportar arquivos.

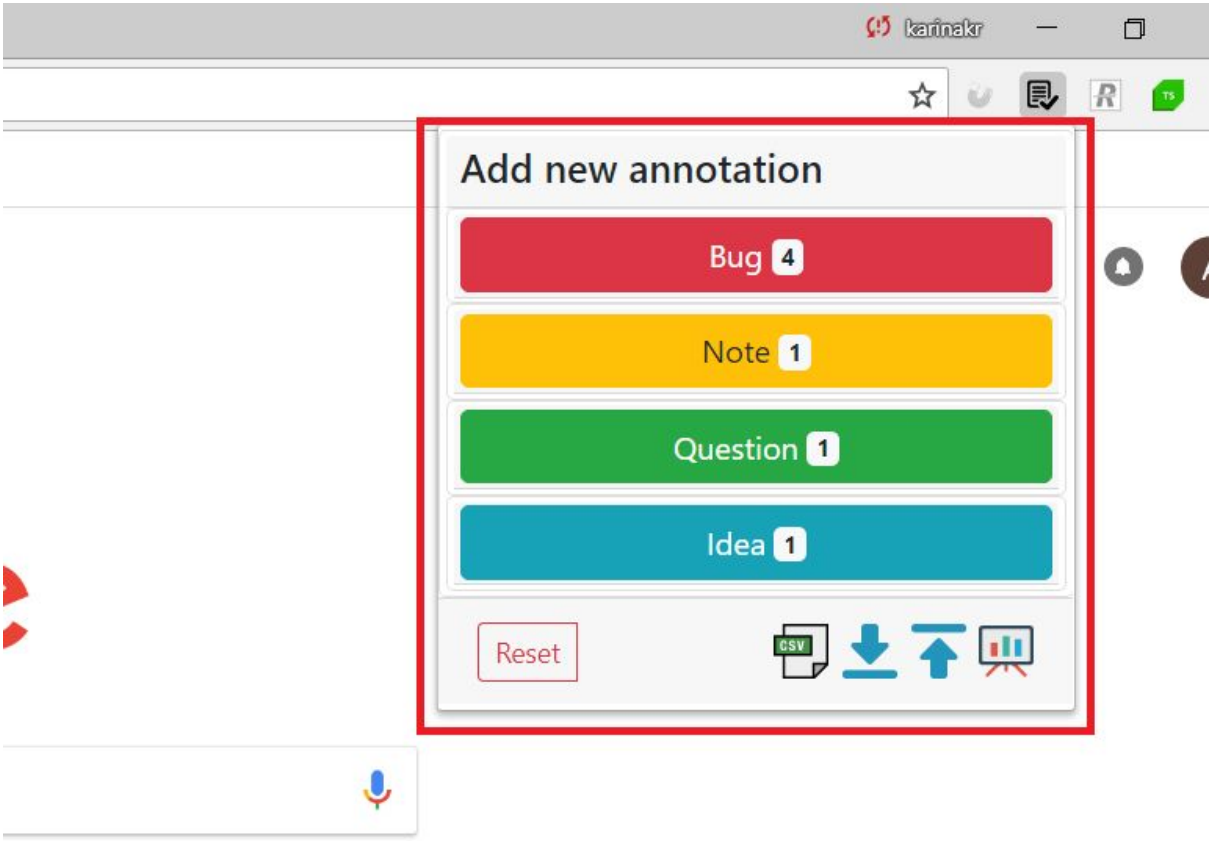

Figura 44: Exploratory Testing Chrome Extension - Detalhe da Interface durante/após a sessão, os números à direita de cada tipo de anotação indica a quantidade anotações feitas até o momento.

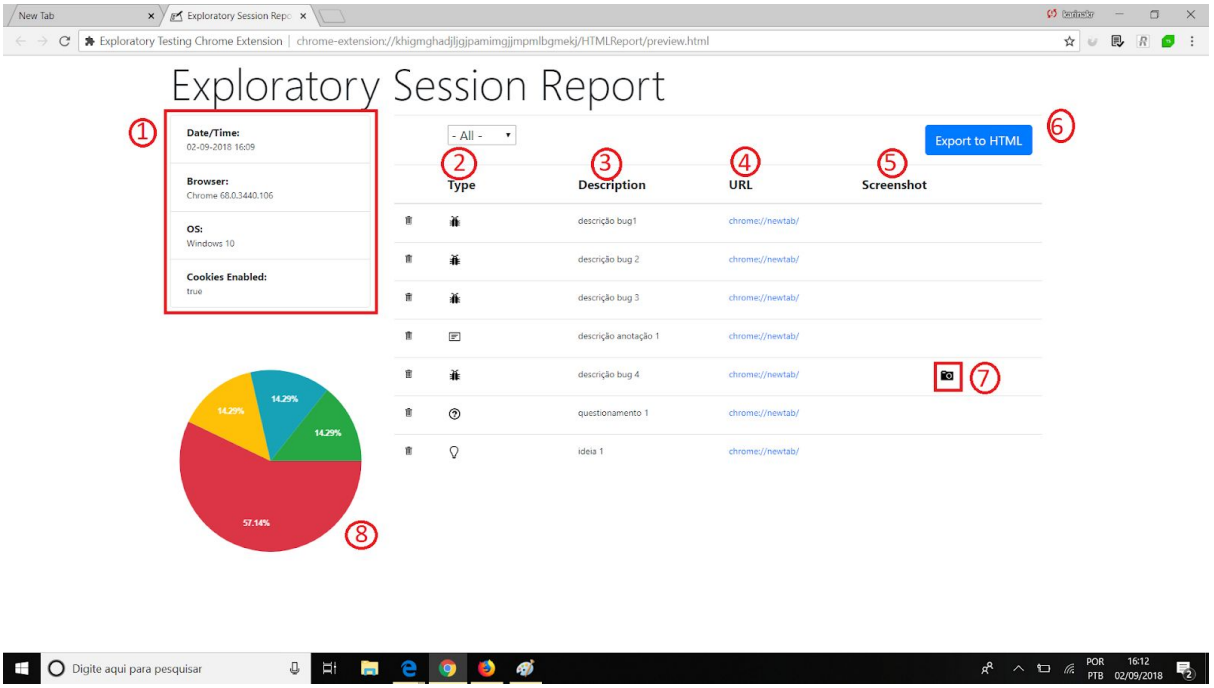

Figura 45: Exploratory Testing Chrome Extension - Relatório de teste que contém informações como: 1 - Informações sobre a sessão e o ambiente, 2 - Tipo de anotações contidas no relatório que podem ser identificadas pelo ícone, 3 - Descrição da anotação, 4 - URL do erro, 5 - Lista de screenshots, no exemplo apenas uma das anotações possui uma screenshot, 6 - Exportar para HTML, 7 - Símbolo que indica a existência de screenshot anexada à anotação.

As ferramentas analisadas podem ser relacionadas as abordagens estudadas da forma vista na tabela 6:

	SBTM	Pairs	TET	Tours	R-BET
TESTSTUFF	X	X	X	X	
RAPISE					X
TESTPAD	X	X	X	X	
BUG MAGNET	X	X	X	X	
TESTRAIL	X	X	X	X	
WINK	X	X	X	X	
JIRA + JIRA CAPTURE*	X	X	X	X	
BB ASSISTANT	X	X	X	X	
RAPID REPORTER	X	X	X	X	
TEST STUDIO EXPLORER					X
EXPLORATORY TESTING CHROME EXTENSION	X	X	X	X	

Tabela 6 - Relação entre ferramentas e abordagens estudadas.

Apêndice B - Link Para as Ferramentas

- Testuff http://www.testuff.com/
- Rapise https://www.inflectra.com/Rapise/
- Testpad https://ontestpad.com/
- Bug Magnet https://gojko.github.io/bugmagnet/
- TestRail http://www.gurock.com/testrail/
- Wink: http://www.debugmode.com/wink/
- JIRA Capture https://www.atlassian.com/software/jira/capture
  zephyr-capture-for-jira (extensão do chrome)
  https://chrome.google.com/webstore/detail/zephyr-capture-for-jira/mmmjimhmoodbiejkjgcecaoibmochpnj
- BBTest Assistant - http://www.bbtestassistant.com/
- Rapid Reporter: http://testing.gershon.info/reporter
- Test studio explorer http://www.telerik.com/download/teststudio
- Exploratory Testing Chrome Extension
  https://chrome.google.com/webstore/detail/exploratory-testing-chrom/khigmghadjljgjpamimgjjmpmlbgmekj?hl=en