TESTE DE APLICAÇÕES WEB PELA MULTIDÃO

Allan Freitas de Carvalho Girão

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador(es): Jano Moreira de Souza

Rio de Janeiro

Setembro de 2013

TESTE DE APLICAÇÕES WEB PELA MULTIDÃO

Allan Freitas de Carvalho Girão

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof. Jano Moreira de Souza, Ph.D.

_____

Prof. Geraldo Bonorino Xexéo, D.Sc.

_____

Prof.ª Vera Maria Benjamim Werneck, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2013

Girão, Allan Freitas de Carvalho

Teste de Aplicações Web pela Multidão / Allan Freitas de Carvalho Girão. – Rio de Janeiro: UFRJ/COPPE, 2013.

XIII, 143 p.: il.; 29,7 cm.

Orientador: Jano Moreira de Souza

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 120-128.

1. Crowdsourcing. 2. Teste de Software. I. Souza, Jano Moreira de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Gostaria primeiramente de agradecer aos meus pais, Ludovina e António, por estarem sempre ao meu lado, me dando o suporte necessário para minha caminhada, juntamente com a minha irmã Aline, avó Nicia, tia Glorinha e meu padrinho Bal, por estarem sempre presentes na minha vida. A minha afilhada Lara, por irradiar a minha vida com alegria.

Agradeço especialmente a minha namorada Sara, cujo apoio, incentivo e compreensão incondicionais foram de suma importância para que este trabalho pudesse ser realizado.

Ao Professor Jano, meu orientador, pela oportunidade e adequada orientação, sem as quais este trabalho não seria possível.

Ao Sérgio, meu coorientador, pelo tempo e atenção dispendidos em prol deste trabalho, além da boa vontade em lê-lo atentamente e aturar minhas indagações.

Aos professores Geraldo Bonorino Xexéo e Vera Maria Benjamim Werneck, por aceitarem fazer parte da banca de examinação, dispendendo um tempo de suas agendas atribuladas para tal.

A todos os amigos da COPPETEC, mestrado, faculdade, colégio e vizinhança, pois sem eles a vida não teria a mesma graça.

Agradeço a todos que de alguma forma colaboraram com a pesquisa e experimentos da dissertação.

Agradeço também a Deus, por tudo que tem me proporcionado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TESTE DE APLICAÇÕES WEB PELA MULTIDÃO

Allan Freitas de Carvalho Girão

Setembro/2013

Orientador: Jano Moreira de Souza

Programa: Engenharia de Sistemas e Computação

Aplicações Web se tornam cada vez mais comuns e mais propensas a modificações, visando atender demandas de um mercado que muda rapidamente de necessidade. Os usuários dessas aplicações são exigentes, querendo sempre obter qualidade, rapidez e usabilidade, devido a facilidade de simplesmente trocar de site para o concorrente. Estas situações trazem à tona a necessidade de exaustivos testes sobre essas aplicações, para que as modificações sejam adequadas e correspondam as expectativas dos usuários. *Crowdsourcing* é o ato de delegar uma tarefa a um grupo de pessoas desconhecidas. Essas tarefas podem ser dos mais variados tipos, exigindo algumas vezes criatividade e inteligência. Essas duas características são fundamentais para a criação de casos de teste, transformando a multidão em um candidato a realização dessa tarefa. Dessa forma, o objetivo do trabalho é possibilitar que a multidão gere casos de teste em aplicações web, culminando com a avaliação da capacidade tanto de criação quanto de julgamento de um caso de teste.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

WEB APPLICATION TESTING BY CROWD

Allan Freitas de Carvalho Girão

September/2013

Advisor: Jano Moreira de Souza

Department: Systems and Computer Engineering

Web applications are becoming more common and predisposed to change, seeking to meet demands of a market that hastily changes its need. The users of such applications are demanding, always wishing to obtain quality, speed and usability, due to the convenience of simply changing to the rival website. These situations bring forward the need of exhaustive tests over these applications, so that the changes are adequate and correspond to users expectations. Crowdsourcing is the act of outsourcing a task to an undefined group. These tasks can be of the most variety of kinds, requiring now and again creativity and intelligence. Those two characteristics are essential for test cases creation, transforming the crowd into a candidate to achieve that task. Therefore, the objective of this work is enabling the crowd to generate test cases for web applications, leading to an evaluation of the ability of creating and judging a test case.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

GWAP – Games With A Purpose

HIT – Human Intelligence Task

IP – Internet Protocol

ITG – Independent Test Group

MTurk – Amazon Mechanical Turk

OO – Orientado a Objetos

SM – Mapeamento Sistemático

SMS – Short Message Service

SR – Revisão Sistemática

SUT – Software Under Test

UML – Unified Modeling Language

URL – Uniform Resource Locator

WA – Web Application

1.Introdução

Com o advento e popularização da Internet, as ideias de construir uma sabedoria coletiva da multidão de SUROWIECKI (2006) e LÉVY (1998) ganhou um instrumento de grande alcance. E como cada vez mais pessoas estão acessando a Internet, maior a demanda por aplicações Web que atendam suas necessidades.

Observando esse cenário, companhias e pessoas visualizam a oportunidade de grandes negócios, buscando atender a uma determinada necessidade de um grupo de pessoas, de diferentes tamanhos. Para não dar chances à concorrência, o tempo de desenvolvimento das aplicações é cada vez menor, onde em muitos casos os testes, que consomem até 50 % do tempo e mais de 50% do custo (MYERS, 2004), são deixados em segundo plano.

Negligenciando os testes, a qualidade fica comprometida, uma vez que os testes são um conjunto de atividades realizadas com o objetivo de avaliar a qualidade e melhorar o software através da identificação de falhas (ABRAN & BOURQUE, 2004).

Para que os testes não sejam negligenciados em favor do tempo de codificação do software, formas de testes que apresentem resultados mais rápidos e menos custosos são buscadas em pesquisas acadêmicas. *Crowdsourcing* tem sido aplicado como um desses caminhos.

Alguns testes têm sido delegados a multidão, como será observado na seção 3.6, como teste de segurança, acessibilidade, aceitação, carga, internacionalização, usabilidade e funcionalidade. Os testes são feitos no nível de sistema e caixa-preta, pois compartilhar o código ou parte dele pode ser considerado prejudicial ao dono do software.

As aplicações existentes funcionam, em sua maioria, de duas formas: como uma ferramenta *Bug Tracker*, onde os testadores relatam os erros encontrados, ou como um *Recorder*, onde a ferramenta grava as ações dos usuários. Testes de regressão são conduzidos apenas quando testadores profissionais elaboram scripts de teste, por exemplo, com a ferramenta Selenium (SELENIUM, 2013).

1.1.Objetivos

Os objetivos deste trabalho são:

1. Criar um processo de teste de software web que utiliza a multidão como fonte de criação e avaliação dos casos de teste;
2. Avaliar a eficiência do processo de testes utilizando métricas apropriadas e comparando a outros modelos ou ferramentas de teste de software web disponíveis na literatura;
3. Avaliar os resultados obtidos através da multidão quanto ao julgamento dos resultados, ou seja, se os testadores são capazes de identificar quando um caso de teste apresenta um erro;
4. Elaborar uma forma da aplicação do processo que seja de fácil entendimento e uso para os testadores, sem necessidade de instalação ou configuração de nenhum componente em seus computadores.

1.2.Limitações do estudo

O título do trabalho demonstra a primeira limitação, onde os tipos de aplicações a serem testadas são apenas aplicações Web. Logo, o processo de teste de software proposto visa apenas aplicações desse gênero.

Para colocar o processo a prova, foi desenvolvida uma plataforma Web de *crowdsourcing* e um *framework* em Java, onde este último deve ser configurado em cada aplicação a ser testada. A outra opção cogitada para o lugar do *framework* era uma ferramenta *proxy*, com a vantagem de ser independente de linguagem, porém com a desvantagem que o código, ao ser retornado para o cliente, deveria ser alterado, o que acarretaria em uma possível mudança no comportamento do sistema, e.g., alteração das URLs. Como o foco é o teste, qualquer mudança poderia comprometer a fidelidade do mesmo. O *framework*, por outro lado, ao agir dentro da própria aplicação, apenas adiciona algumas funcionalidades.

A multidão considerada no trabalho é pequena, com um número entre 4 e 9 pessoas utilizando o sistema ao mesmo tempo. Por outro lado, o tempo para utilização também é reduzido, variando entre 8 e 73 minutos para cada experimento. A multidão é composta em grande maioria por pessoas da área de Tecnologia da Informação, todas com ao menos a graduação em curso.

De forma resumida, temos para a multidão e para a aplicação as limitações:

- Multidão:
  - Maioria das pessoas de TI, com no mínimo a graduação em curso;

    - Composta de 4 a 9 pessoas;
    - Testando entre 8 a 73 minutos.
- Aplicação:
    - Em Java Web.

1.3.Organização do trabalho

O Capítulo 1 mostra o escopo do problema a ser tratado, assim como as suas limitações. A organização do trabalho e os objetivos a serem alcançados pela pesquisa também são retratados nesse capítulo.

O Capítulo 2 apresenta uma revisão de literatura sobre um dos principais temas dessa dissertação: *crowdsourcing*. Através da ideia original da sabedoria das multidões, mostra como surgiu o termo e quais as características que aplicações desta natureza possuem.

O Capítulo 3 cobre o outro tema central desse trabalho, apresentando conceitos importantes no que tange aos testes de software. Por ser um campo de pesquisa muito vasto, apenas os conceitos que são de alguma forma inerentes a este trabalho são apresentados, de forma resumida. Este capítulo termina com exemplos de aplicações já existentes que aplicam o conceito de *crowdsourcing* em testes de software.

O Capítulo 4 descreve o processo de testes proposto por esse trabalho, com todas as suas características. O texto segue descrevendo a forma de aplicação do processo, ou seja, a ferramenta criada para que os objetivos desse trabalho sejam alcançados.

O Capítulo 5 traz um relato de todos os experimentos realizados, incluindo os motivos que levaram a escolha das aplicações utilizadas para teste, das ferramentas comparadas e das métricas utilizadas para comparação.

O Capítulo 6 conclui o trabalho com o relato dos objetivos atingidos e novas oportunidades a serem exploradas a partir deste trabalho. Ao final, são apresentadas as Referências Bibliográficas e os Apêndices da pesquisa.

2.Sabedoria das multidões

Em seu livro intitulado *A Sabedoria das Multidões*, SUROWIECKI (2006, p.12) argumenta que "[...] sob as circunstâncias corretas, grupos são impressionantemente inteligentes, e frequentemente são mais inteligentes que a pessoa mais inteligente em seu interior". O autor define essa sabedoria das multidões como a inteligência coletiva que existe quando as avaliações individuais imperfeitas são agregadas de forma adequada. É como se todos nós fossemos programados para sermos inteligentes coletivamente.

Antes de Surowiecki, Lévy já tratara sobre essa inteligência coletiva em uma visão bastante utópica, de acordo com BRABHAM (2008). Segundo LÉVY (1998, p.25): "Os conhecimentos vivos [...] e competências dos seres humanos estão prestes a ser reconhecidos como a fonte de todas as outras riquezas.", onde as novas ferramentas de comunicação são os instrumentos para alcançar esse objetivo.

LÉVY (1998, p.28) define a inteligência coletiva como "[...] uma inteligência distribuída por toda parte, incessantemente valorizada, coordenada em tempo real, que resulta em uma mobilização efetiva das competências.". Dessa definição, extraem-se quatro fundamentos importantes: a distribuição, a valorização, a coordenação e a mobilização das competências. O reconhecimento e o enriquecimento mútuo são os objetivos principais desse tipo de inteligência (LÉVY, 1998). BRABHAM (2008), porém, alega que LÉVY (1998) possui uma visão muito utópica da sociedade prosperando para uma inteligência coletiva, mas afirma que o conhecimento tácito e as habilidades individuais estão em processo de serem reconhecidos como a fonte primária de toda riqueza.

Uma inteligência distribuída por toda parte é o fato de que "Ninguém sabe tudo, todos sabem alguma coisa, [e] todo saber está na humanidade." (LÉVY, 1998, p.29). Ele ainda defende que a inteligência é muitas vezes desvalorizada, alegando que deveria existir uma maior preocupação da humanidade com o desperdício de inteligência. Com o ciberespaço, a possibilidade de uma coordenação inteligente em tempo real seria possível para o compartilhamento de conhecimento entre os conhecedores. Através do reconhecimento das competências dos outros, é possível desenvolver uma mobilização efetiva no conhecedor, motivando-o a compartilhar os conhecimentos e colaborar coletivamente em projetos.

Esse capítulo apresenta as principais características que permeiam a Sabedoria das Multidões, como Diversidade, Independência, Descentralização. Uma abordagem mais profunda do conceito de *Crowdsourcing* e suas características é também relatada. O capítulo se encerra com as considerações gerais acerca de como o presente trabalho utiliza essas características.

2.1.Características das multidões

Para garantir a sabedoria das massas, ela deve atender a algumas condições: diversidade, independência e descentralização, onde as duas primeiras são importantes, pois "[...] as melhores decisões coletivas são fruto de discordância e contestação, não de consenso e acordo." (SUROWIECKI, 2006, p.18). As três características são apresentadas de forma detalhadas nas próximas seções.

2.1.1.Diversidade

Quanto mais diferentes duas pessoas são, mais conhecimentos distintos elas possuem. Imagine duas pessoas que nasceram no mesmo lugar, estudaram na mesma escola, na mesma época e fizeram a mesma faculdade. Muito do conhecimento que elas têm é partilhado entre elas. É claro que cada uma tem o seu próprio conhecimento, seu próprio entendimento particular das coisas. Pense agora em duas pessoas completamente diferentes, que nasceram em países diferentes, em modelos de escola diferente, épocas diferentes, que se especializaram em áreas distintas.

Qual dos dois casos, se tivéssemos que unir o conhecimento de ambas, resultaria em um conhecimento maior? A resposta intuitiva é o segundo caso. De acordo com MARCH (1991), o conhecimento acumulado de um grupo com pessoas heterogêneas é mais amplo porque cada membro contribui com mais informações para o todo, enquanto em um grupo homogêneo cada novo membro agrega menos informações. A diversidade é essa diferença entre os membros, é a singularidade persistindo em grandes grupos (HOWE, 2009).

O Teorema da Capacidade de Superação da Diversidade[1], de PAGE (2007, p.162), diz que dada as devidas condições, "[...] um grupo de solucionadores de problemas escolhido aleatoriamente supera em desempenho um conjunto formado

[1] Tradução obtida em HOWE (2009, p.116) para: "Diversity Trumps Ability Theorem".

pelos melhores solucionadores"[2]. Seria, porém, uma aleatoriedade controlada onde as pessoas sejam capazes de realizar a tarefa, por exemplo, pedir que um grupo de transeuntes do metrô programe um software é diferente de pedir para pessoas do curso de ciência da computação fazê-lo.

BRABHAM (2007) sugere que o conceito de diversidade proposto por SUROWIECKI (2006) seja quebrado em pedaços menores: diversidade de identidade, de habilidades e de postura política.

A identidade – gênero, sexualidade, raça, nacionalidade, classe econômica, religião – de cada pessoa faz com que ela se torne única, tornando sua visão de mundo diferente das demais. O exemplo citado no início da seção, comparando o conhecimento entre duas pessoas parecidas e duas pessoas distintas, faz uso da diversidade de identidade (BRABHAM, 2007).

As habilidades de cada um auxiliam em problemas com complexidades maiores ou que tenham um escopo mais restrito. Quanto mais habilidades o grupo tiver, mais problemas de diferentes níveis e diferentes áreas de conhecimento ele será capaz de resolver (BRABHAM, 2007).

A postura política é importante quando se quer chegar a uma solução que deverá atender a vários grupos políticos diferentes (BRABHAM, 2007). Uma solução efetiva, politicamente falando, será alcançada se todas as partes puderem opinar, expondo seus pontos de vista.

O conceito de diversidade varia de questão para questão. Um problema computacional, por exemplo, não necessitaria de diversidade política, como sugeriu BRABHAM (2007), porém, uma diversidade de identidade talvez fosse desejada. Uma diversidade, nesse caso, poderia ser tida como diferentes experiências profissionais, diferentes áreas de conhecimento da computação ou diferentes faculdades cursadas pelos membros do grupo.

2.1.2.Independência

SUROWIECKI (2006, p.86) argumenta que: "Decisões coletivas tendem a ser melhores quando tomadas por pessoas de opiniões diferentes, chegando a conclusões independentes, baseadas fundamentalmente em suas informações pessoais.". O que

[2] Tradução do autor para: "[...] a randomly selected collection of problem solvers outperforms a collection of the best individual problem solvers."

torna a independência importante para a tomada de decisões é que os erros individuais não irão ser propagados, no que ele chama de cascata de informações. Além desse fato, indivíduos independentes tem uma probabilidade maior de ter novas informações, agregando maior conhecimento ao grupo como um todo.

SUROWIECKI (2006) ilustra o caso com o experimento conduzido por dois economistas, Angela Hung e Charles Plott, ao pedir que alunos retirassem bolas (escuras ou claras) de uma urna. Havia duas urnas, a urna A continha duas vezes mais bolas escuras que claras e a urna B duas vezes mais bolas claras que bolas escuras. Antes da experiência, os economistas escolhiam uma das urnas para ser usada no experimento, sem que os alunos soubessem da escolha. Em um primeiro experimento, o aluno retirava uma bola, e não era permitido revelar aos outros a cor de sua bola. Cada aluno palpitaria em voz alta qual urna estaria sendo usada. Para isso ele possuía duas informações: a cor da sua bola (informação pessoal) e os palpites anteriores (informação coletiva). Cada aluno que palpitasse certo era recompensado no final. Caso o palpite do primeiro fosse pela urna B, imagina-se que ele tenha tirado uma bola clara. Porém, caso os três primeiros tenham dito B e o próximo aluno a palpitar tenha tirado uma bola escura, seu palpite provavelmente será a urna B, iniciando uma cascata da informação. Ao mudar a regra do jogo, trocando a premiação do acerto do palpite individual para o acerto do palpite coletivo, cada um deixou de se preocupar com a informação coletiva e passou a se preocupar com a informação individual, para tornar o grupo mais inteligente.

Esse experimento pode ser concluído com a seguinte frase: “Um segredo para decisões grupais bem-sucedidas é fazer com que as pessoas prestem muito menos atenção ao que todos os outros estão dizendo” (SUROWIECKI, 2006, p.96).

2.1.3. Descentralização

Um sistema centralizado é aquele que tem um comando central, de onde surgem as ordens e decisões a serem acatadas por todos os integrantes deste sistema. Um sistema descentralizado, por sua vez, é aquele onde o poder não está em um ponto central, e as decisões importantes são tomadas pelos indivíduos, baseados em seus conhecimentos específicos e locais (SUROWIECKI, 2006).

Ainda de acordo com SUROWIECKI (2006), colocar pessoas egoístas e independentes para trabalhar de forma descentralizada em um problema resultará em

uma solução melhor que qualquer outra solução que se poderia encontrar. Isso se deve ao fato de cada indivíduo poder expor seu ponto de vista, não se deixando influenciar e agregando seu conhecimento ao coletivo.

Um ponto a favor da descentralização é que a mesma instiga a independência, a especialização e a coordenação individual das próprias atividades. Um ponto contra é que uma informação valiosa encontrada em uma parte do sistema não tem garantia de encontrar seu caminho no restante do sistema (SUROWIECKI, 2006).

Um exemplo de sucesso de um sistema descentralizado é o Linux, não possuindo uma organização formal (SUROWIECKI, 2006). Ele é feito por programadores de diversas partes do mundo, sem recompensas monetárias, apenas pelo prazer de contribuir com o crescimento e fortalecimento de algo maior, se mostrando superior a outros sistemas desenvolvidos de forma centralizada (BRABHAM, 2008).

2.2. Crowdsourcing

O termo *Crowdsourcing* foi cunhado por Jeff Howe e Mark Robinson (HOWE, 2006a), e utilizado pela primeira vez por HOWE (2006b), sendo formado da contração das palavras *Crowd*, que significa multidão, e *Outsourcing*, que pode ser traduzida como terceirização, uma forma de obter mão de obra externa. Logo, *Crowdsourcing* pode ser entendido como delegar uma tarefa a uma multidão de pessoas.

HOWE (2006a) define *crowdsourcing* como "[...] o ato de uma companhia ou instituição pegar uma função geralmente desenvolvida por um empregado e terceirizar para uma rede de pessoas indefinidas na forma de chamado aberto"[3]. Quando diz que será em forma de chamado aberto significa que não há restrições com relação a quais pessoas podem fazer parte do chamado, uma vez que qualquer pessoa que possua acesso ao meio como esse chamado foi feito poderá participar.

Com o advento da internet, os sistemas descentralizados puderam ter aplicações mais práticas (SUROWIECKI, 2006). Esse é um fator determinante para o surgimento do conceito de *Crowdsourcing*.

[3] Tradução do autor para: "[...] the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call."

Para BRABHAM (2008, p.79), *crowdsourcing* é um "[...] modelo estratégico para atrair uma multidão motivada e interessada de indivíduos capazes de prover soluções superiores em qualidade e quantidade em comparação as soluções obtidas nas formas tradicionais"[4]. O mesmo ainda argumenta que o modelo é capaz de economizar tempo e custos agregando conhecimentos.

2.2.1.Termos relacionados

Alguns termos podem ser confundidos com *crowdsourcing*, tendo uma relação estreita, e outros ainda são extensões do termo ou subáreas do assunto. Esta seção se dedica a definir e comparar, quando possível, alguns termos utilizados na literatura juntamente com o termo *crowdsourcing*. A Figura 1 ilustra a interseção dos termos *crowdsourcing*, computação humana e inteligência coletiva.

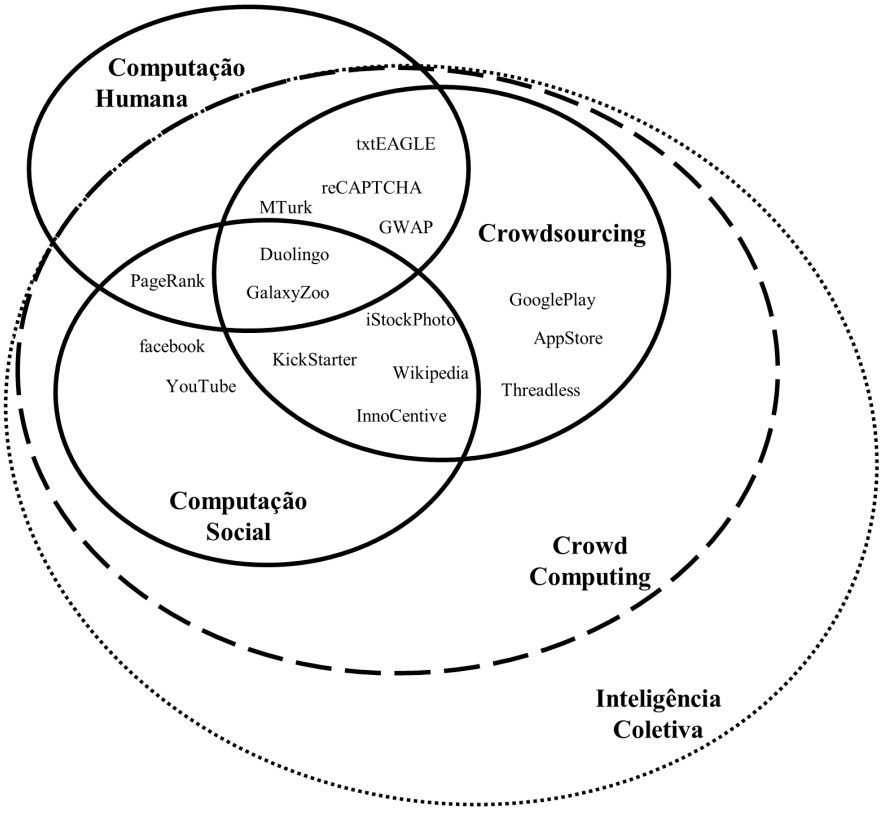

Figura 1 - Diagrama de Venn para *Crowdsourcing* e termos relacionados, baseado em QUINN & BEDERSON (2011), SCHNEIDER *et al.* (2011) e GOMES *et al.* (2012)

[4] Tradução do autor para: "[...] strategic model to attract an interested, motivated crowd of individuals capable of providing solutions superior in quality and quantity to those that even traditional forms of business can".

2.2.1.1.Computação Humana

De acordo com GEIGER *et al.*(2011), computação humana é uma área específica de aplicações de *crowdsourcing*. VON AHN (2005) em sua tese de doutorado intitulada *Human Computation* cita vários exemplos de obras grandiosas que obtiveram sucesso em sua construção porque uma multidão de pessoas colaborou para tal. O canal do Panamá (44.733 pessoas), a pirâmide de Quéops (50.000 pessoas), no Egito, e o projeto Apollo (400.000 pessoas) são alguns dos exemplos (VON AHN, 2009)

QUINN & BEDERSON (2011) explicam um pouco do histórico do termo, passando pelo seu uso em 1838 no contexto da filosofia e psicologia e mais recentemente (1950), no contexto da computação, por Alan Turing. Porém, a utilização moderna do termo é inspirada pela tese de VON AHN (2005), que o define como "[...] um paradigma para a utilização do poder de processamento humano para solucionar problemas que os computadores ainda não podem resolver"[5] (VON AHN, 2005, p.3). Por sua vez, QUINN & BEDERSON (2009, p.2) definem computação humana como "[…] sistemas de computadores e um grande número de pessoas que trabalham juntos para resolver problemas que não podem ser resolvidos nem por computadores nem por humanos sozinhos"[6] e YUEN *et al.* (2011, p.766) define como "[sistemas] que utilizam as habilidades humanas para realizar tarefas computacionais que são difíceis para os computadores processarem."[7].

De acordo com VON AHN (2009), essa nova área da ciência da computação está sendo desenvolvida para solucionar problemas que nem os humanos nem os computadores sozinhos conseguem resolver, mas que podem alcançar uma solução caso ambos trabalhem em conjunto. Para QUINN & BEDERSON (2011), as definições citadas deixam claro que os sistemas de computação humana são projetados para resolver um problema específico, através de um fluxo predeterminado de atividades, em contraste com outros tipos de sistema, onde a iniciativa e fluxo de

[5] Tradução do autor para: "[...] a paradigm for utilizing human processing power to solve problems that computers cannot yet solve"

[6] Tradução do autor para: "[...] systems of computers and large numbers of humans that work together in order to solve problems that could not be solved by either computers or humans alone"

[7] Tradução do autor para: "[...] which utilizes human abilities to perform computation tasks that are difficult for computers to process."

atividades são direcionados principalmente pela inspiração dos participantes.

2.2.1.2.Computação Social

O termo computação social traduz a ideia do uso da computação como um suporte, uma ferramenta para a interação social, mais precisamente definido por WANG *et al.* (2007) como "uma forma de facilitar computacionalmente os estudos e dinâmicas sociais, assim como a concepção e uso de tecnologias de informação e comunicação no contexto social"[8].

O foco é que as pessoas tenham uma vida social online ativa, através de blogs, fóruns, email, mensageiros, redes sociais e *wikis* (GOMES *et al.*, 2012), facilitados por aplicações móveis instaladas em smartphones com disponibilidade de internet, acessíveis a qualquer hora, em qualquer lugar (GIRÃO *et al.*, 2012).

2.2.1.3.Crowd Computing

O conceito de *crowd computing* é mais abrangente que *crowdsourcing*, computação humana ou computação social. SCHNEIDER *et al.* (2011) define *crowd computing* como "[...] uma miríade de ferramentas de interação humana que permitem a utilização do espaço mental de multidões", espaço esse que é englobado pela inteligência coletiva.

2.2.1.4.Open source

Open source pode ser vista como uma filosofia de desenvolvimento de produtos, normalmente aplicada a desenvolvimento de software (BRABHAM, 2008). É uma forma rápida e de baixo custo, onde contribuintes individuais e/ou empresas cooperam no sentido de desenvolver algo que poderá ser usado por qualquer um (PERENS, 1999, BRABHAM, 2008).

Um programador sente-se confortável ao colaborar com o desenvolvimento de softwares que seguem a filosofia *Open Source* porque ele poderá fazer cópias, acessar o código-fonte e melhorar o programa como desejar (PERENS, 1999). Exemplos de softwares desenvolvidos dessa forma e largamente utilizados são o Linux, Mozilla Firefox e o servidor web Apache. Essa forma de desenvolvimento tornou-se sucesso na produção de softwares, pois o custo de realizar uma cópia do produto é muito

[8] Tradução do autor para: "Computational facilitation of social studies and human social dynamics as well as the design and use of ICT technologies that consider social context".

baixo (PERENS, 1999).

Crowdsourcing não é uma prática da filosofia Open Source, uma vez que os produtos ou soluções gerados pela multidão envolvida se torna propriedade da empresa ou grupo que abriu o chamado para aquela tarefa (BRABHAM, 2008).

2.2.1.5.Open Innovation

De acordo com CHESBROUGH (2006), *Open Innovation* muda a forma de pesquisa e inovação por parte das empresas, com fluxos de conhecimento partindo de fora para dentro, visando acelerar a inovação interna; e de dentro para fora, visando expandir o mercado com o uso da inovação, uma vez que o conhecimento está amplamente distribuído e conhecimentos externos devem ser agregados ao processo de inovação da empresa.

Sua semelhança com *crowdsourcing* reside no fato de que ambos utilizam o conhecimento distribuído, tornando a agregação desse conhecimento externo uma vantagem para as empresas (SCHENK & GUITTARD, 2011). Porém, há duas diferenças cruciais entre os dois: a primeira é que *Open Innovation* foca apenas no processo de inovação da empresa, enquanto *crowdsourcing* não tem um foco definido, podendo ser usado para qualquer fim; a segunda diferença é que *Open Innovation* descreve os fluxos de conhecimento principalmente entre empresas, enquanto *crowdsourcing* refere-se a conexões entre firmas e uma multidão de pessoas anônimas (SCHENK & GUITTARD, 2011).

2.2.2.Características

HOWE (2009, p.246) afirma que *crowdsourcing* é "[...] um termo abrangente para um grupo altamente variado de métodos que, obviamente, têm um atributo em comum: todos dependem de alguma contribuição da multidão". Nessa variedade reside a dificuldade em encontrar padrões para classificação das práticas de *crowdsourcing*. Vários autores classificaram os sistemas de *crowdsourcing,* computação humana e inteligência coletiva (HOWE, 2009, SCHENK & GUITTARD, 2011, QUINN & BEDERSON, 2011, GEIGER *et al.*, 2011), cada um a sua maneira, utilizando diferentes dimensões para avaliação. Esta seção visa condensar as características mais relevantes dos sistemas de *crowdsourcing*.

2.2.2.1. Tipos

SCHENK & GUITTARD (2011) dividem os sistemas em três tipos de tarefas realizadas pelo usuário: simples, complexas e criativas. As tarefas simples não exigem que o usuário possua um conhecimento específico, como tarefas para coleta de dados e tradução de pequenos textos. Aplicações exemplos de tarefa simples são o reCAPTCHA (ver seção 2.2.3.7) e o txtEagle (EAGLE, 2009), um sistema de tradução via SMS. As pequenas tarefas normalmente são de natureza integrativa e possuem micropagamentos. Tarefas complexas, por sua vez, necessitam de um conhecimento específico prévio por parte do usuário. Essas tarefas são normalmente seletivas. Um exemplo deste tipo é o InnoCentive (ver seção 2.2.3.1). As tarefas criativas, como o nome já diz, exigem criatividade por parte do usuário, mais especificamente criatividade artística.

HOWE (2009), por sua vez, divide as aplicações de *crowdsourcing* no que ele chama de modelo, podendo ser compreendida como a natureza da contribuição do usuário na aplicação, existindo quatro tipos não mutuamente exclusivos. Estes tipos são: a inteligência coletiva ou sabedoria das multidões, a criatividade, o voto e o *crowdfunding*. Essas categorias são amplas o suficiente para que haja variações, onde cada iniciativa nova deverá combinar e ajustar os mecanismos de *crowdsourcing* que mais se adequem aos seus objetivos.

Além desses tipos, diversos outros tipos de tarefas realizadas foram verificados na literatura (VON AHN, 2005, COHN, 2008, EAGLE, 2009, ROSS *et al.*, 2010).

Inteligência coletiva

A inteligência coletiva se baseia em um dos princípios centrais do *crowdsourcing*: grupos contêm mais conhecimento que indivíduos isolados. HOWE (2009) ainda divide este grupo em três tipos:

- Mercado de previsões ou mercado de informações: semelhante ao mercado de ações, no qual investidores compram acontecimentos futuros, como, por exemplo, quem irá ganhar o Oscar em cada categoria ou quem será eleito o novo presidente. Esse tipo valoriza os sábios, ao oferecer retorno financeiro em troca da inteligência, sendo esse o mesmo incentivo para manter os tolos fora.

- *Crowdcasting*: Delegar a solução de um problema a uma rede maior e indefinida de potenciais solucionadores, como o InnoCentive. Quanto mais pessoas e maior a diversidade melhor, pois a melhor solução não será inferiorizada pelas piores soluções.
- *Idea jams*: Sessão de *brainstorm* online que acontece ao longo de semanas, visando gerar novas ideias ou soluções para questões que ainda não apareceram.

Criatividade

Como SCHENK & GUITTARD (2011) mencionam, algumas tarefas são dependentes de habilidades específicas de cada indivíduo. Essas tarefas não precisam ser complexas nem simples, apenas exigirem do contribuinte um determinado nível de criatividade.

A criatividade é a utilização da energia criativa excedente das pessoas, geralmente envolvendo uma comunidade forte de pessoas com compromisso profundo com o trabalho e com os parceiros (HOWE, 2009). Um exemplo dessa natureza de *crowdsourcing* é o iStockphoto (ver seção 2.2.3.2). A multidão que compõe essa comunidade é composta tanto por profissionais da área como por amadores, interessados em aprimorar suas técnicas de fotografia, por exemplo, utilizando a comunidade como uma forma de aprender e ensinar.

Essa criatividade pode estar no fato de elaborar uma nova estampa de camisa (THREADLESS, 2012) ou no fato de capturar uma boa foto (ISTOCKPHOTO, 2012). A capacidade de desenvolver aplicativos diferentes para a App Store (APP STORE, 2012) ou o Google Play (GOOGLE PLAY, 2012) também é uma forma de criatividade que necessita também um conhecimento específico.

Voto

O poder de voto da multidão faz uso da opinião das pessoas integrantes da massa para organizar grandes volumes de informação. Pode ser utilizada com os dados gerados por outras formas de *crowdsourcing*, por exemplo, no iStockphoto, ao julgar se uma fotografia é boa ou não. Esse julgamento pode ser dado de forma explícita, como o esquema de votação cinco estrelas, ou de forma implícita, contabilizando os acessos a uma foto no iStockphoto.

Crowdfunding

Crowdfunding aproveita a renda coletiva para acumular recursos financeiros, permitindo que as pessoas financiem projetos em que acreditam, através de pequenas doações. Diferente dos outros três tipos, não aproveita a capacidade – seja de criar, de pensar ou de julgar – excedente das pessoas, e sim o dinheiro.

De acordo com ORDANINI *et al*. (2011), *crowdfunding* é um fenômeno que descreve o ato de uma multidão de pessoas unirem recursos para investir e patrocinar esforços que outras pessoas ou organizações iniciaram, normalmente através da internet. Por meio de pequenas e médias contribuições de uma multidão de pessoas, é possível chegar a um montante final considerável, que poderá ser usado para apoiar projetos de variados tipos, como empresas de *start-ups* ou inovações tecnológicas. Um exemplo de aplicação de *crowdfunding* é o Kickstarter (KICKSTARTER, 2012), que auxilia a financiar projetos criativos, como filmes, jogos, quadrinhos, música, teatro e outros.

A ideia de juntar pequenas frações de dinheiro para um bem maior não é nova. Na esfera da caridade esse modelo é bastante utilizado, onde os caridosos não esperam retorno, diferente de *crowdfunding*, onde o retorno é aguardado pelos contribuintes, seja ele financeiro ou não (ORDANINI *et al.*, 2011). Sua semelhança com *crowdsourcing* está no fato da multidão unir esforços em prol de um bem maior, e sua diferença é que a multidão não provê ideias ou soluções, e sim dinheiro para patrocinar iniciativas já existentes.

Processamento de linguagem natural

Humanos podem superar computadores em vários tipos de pequenas tarefas, como transcrição, interpretação e tradução de textos e áudios (EAGLE, 2009). Tarefas essas que tem ligação direta com a linguagem dos seres humanos, tanto escrita como falada, englobando os mais diferentes idiomas e alfabetos ao redor do mundo.

Um exemplos de aplicação desse gênero é o txtEagle (EAGLE, 2009), um sistema que envia mensagens de texto (SMS) para celulares pedindo para as pessoas traduzirem pequenas frases, que foi implantado no Quênia. Algumas tarefas de transcrição de áudio podem ser vistas no Amazon Mechanical Turk - Mturk (AMAZON MECHANICAL TURK, 2013). O Duolingo (DUOLINGO, 2013) é outro exemplo de aplicação desse gênero, que busca traduzir a internet ao mesmo tempo em

que ensina novos idiomas para as pessoas, esse último funcionando tanto como uma motivação e quanto como uma forma de aumentar o número de potenciais participantes.

Processamento visual

Mesmo os programas mais avançados ainda esbarram em um problema que é trivial para os humanos: o reconhecimento de imagens (VON AHN, 2005). Computadores ainda não são capazes de rotular e categorizar imagens com a mesma facilidade que os seres humanos, especialmente se o computador olhar apenas para a imagem, já que os algoritmos normalmente retiram essas informações do contexto onde as imagens estão inseridas (VON AHN, 2009). A transcrição de imagens, apesar dos algoritmos de OCR, ainda está longe da perfeição, considerando que 30% dos textos antigos não conseguem ser reconhecidos por esses programas (VON AHN, 2009).

Tarefas como estas, de categorizar, rotular, transcrever e até buscar em imagens, são inatas para os seres humanos. Por isso, *crowdsourcing* vem sendo usado com essa finalidade, em exemplos como o ESP Game (ver seção 2.2.3.5) e o reCAPTCHA. Eles usam o processamento humano de imagens para realizar tarefas quebrando-as em menores, motivando de diferentes formas, desde jogos até implicitamente.

Pesquisa e desenvolvimento

Algumas aplicações de *crowdsourcing* tem seu foco voltado para a pesquisa e o desenvolvimento, como é o caso da plataforma InnoCentive e do GalaxyZoo (ver seção 2.2.3.6), um exemplo de uma área que é conhecida como *Citizen Science*.

O primeiro busca soluções para determinado problema, que podem ser algumas vezes facilmente encontradas por pessoas de outras áreas, através de soluções já conhecidas, ou através do desenvolvimento de novas soluções e pesquisas por parte da multidão. Dessa forma, apenas uma solução – a melhor – é escolhida.

Citizen Science pode ser interpretado como a ciência realizada por cidadãos. De acordo com COHN (2008, p.193), *citizen scientists* são "[...] voluntários que participam como assistentes de campo em estudos científicos"[9], podendo auxiliar em tarefas como monitorar animais selvagens ou plantas. Normalmente são cidadãos

[9] Tradução do autor para: "[...] volunteers who participate as field assistants in scientific studies."

amadores no tipo de ciência envolvido e não analisam dados, apenas auxiliam em sua coleta (COHN, 2008). GalaxyZoo é um exemplo de aplicação que envolve cidadãos voluntários para auxiliar na pesquisa, através da extração de dados de imagens de galáxias, fornecendo informações para uma base que será analisada por especialistas da área.

Plataforma

Alguns softwares de *crowdsourcing* não possuem uma tarefa específica, servindo como plataformas para que outras pessoas ou empresas façam uso de seu sistema para a realização de tarefas pela multidão. É o caso do MTurk, onde as pessoas ou empresas conseguem colocar tarefas dos mais variados tipos, como processamento visual, de texto ou de áudio, coleta de dados para *surveys* e experimentos e outros (BUHRMESTER *et al.*, 2011, KITTUR *et al.*, 2011).

Algumas plataformas tem um segmento específico de tarefas, como é o caso do InnoCentive, onde pessoas ou empresas podem submeter suas tarefas de pesquisa a uma multidão de pessoas sem ter a necessidade de construir sua própria estrutura para isso.

2.2.2.2.Agregação

As contribuições de sistemas de *crowdsourcing* podem ter diversas finalidades e, para atingi-las, a forma como se agrega os resultados das tarefas dos usuários podem ser distintas. Essa agregação pode ser de dois tipos: integrativa ou seletiva (SCHENK & GUITTARD, 2011, GEIGER *et al.*, 2011).

A forma integrativa é muito útil na coleta de grande quantidade de informações ou dados. Nela, deve se ter cuidado com os dados inseridos, a fim de evitar redundâncias e incompatibilidades. O mérito da forma integrativa é a quantidade. Um exemplo é o iStockphoto (ISTOCKPHOTO, 2012), pois quanto mais opções de fotografias ele lhe fornecer, melhor. A forma seletiva, por sua vez, utiliza a quantidade para obter qualidade. Através da seleção da melhor dentre todas as respostas da multidão, a qualidade tem um salto. Como apenas uma solução é selecionada, é uma forma de competição entre os usuários, fazendo com que cada pessoa dê o seu melhor (SCHENK & GUITTARD, 2011). A forma seletiva age como uma competição entre os usuários, enquanto a forma integrativa não (GEIGER *et al.*, 2011)

QUINN & BEDERSON (2011) dividem a forma integrativa em seis formas diferentes de agregar o resultado:

- Coleção: cada tarefa contribui para a formação de uma ontologia ou hierarquia;
- Estatística: utiliza média simples ou mediana das contribuições;
- Melhoria iterativa: apresentar as contribuições anteriores sobre a mesma tarefa para o próximo usuário, podendo fazer com que o resultado seja mais satisfatório;
- Aprendizado ativo: coletar informações para a formação de uma base de dados de treinamento e testes para ser utilizado em aprendizagem de máquina;
- Busca: usar a multidão para buscar determinado dado em imagens ou vídeos;
- Nenhuma: muitas vezes não é preciso agregar informações, apenas a tarefa realizada é importante. Usada para pequenas tarefas em grande quantidade.

O iStockphoto é um exemplo de integração que agrega os resultados em forma de coleção, enquanto o reCAPTCHA, apesar de juntar todas as transcrições para formar um texto, se encaixa na última forma de integração - nenhuma.

2.2.2.3.Competências dos indivíduos

QUINN & BEDERSON (2011) citam que uma das características de um sistema de *crowdsourcing* é a competência humana necessária para que o indivíduo possa dar sua contribuição para o sistema, como a capacidade de ler em chinês para realizar uma tradução ou a capacidade de programar um software, havendo uma pré-seleção, um filtro dos usuários do sistema.

Essa pré-seleção de indivíduos simboliza uma restrição de potenciais contribuintes ao sistema. Apesar de SUROWIECKI (2006) afirmar que a diversidade é uma das características necessárias à multidão, essa diversidade, em determinados aspectos e sistemas, podem comprometer a qualidade dos dados (GEIGER *et al.*, 2011). Portanto, ele define quatro tipos de pré-seleção que podem ser encontrados e utilizados nesses sistemas:

- Baseado na qualificação: sistemas que exigem o mínimo de conhecimento ou habilidades específicas dos seus usuários;
- Específicos de um contexto: sistemas que querem utilizar apenas a multidão

interna da empresa, para capturar apenas as opiniões dos seus funcionários;

- Ambos: sistemas que exigem tanto a qualificação dos contribuintes quando o contexto dos mesmos. É a maior restrição possível na dimensão da pré-seleção de indivíduos;
- Nenhum: sistemas que não restringem quais os indivíduos que podem fazer parte da multidão.

Como exemplo, o iStockphoto baseia-se na qualificação, exigindo um mínimo de habilidade para fotografias. O reCAPTCHA, por exemplo, exige o conhecimento do alfabeto para identificar os caracteres, mas o conhecimento do idioma facilita o reconhecimento da palavra como um todo.

2.2.2.4.Controle de Qualidade

A qualidade do resultado de um sistema de *crowdsourcing* depende da qualidade das contribuições humanas fornecidas ao sistema. Mesmo usuários motivados podem tentar trapacear no sistema, especialmente quando se envolve dinheiro e/ou competição, ou mesmo a falta de entendimento da tarefa por parte do usuário pode provocar a inserção de dados não qualificados para o propósito, dizem QUINN & BEDERSON (2009, 2011). Eles definem algumas formas para que esse controle de qualidade seja realizado:

- Concordância forçada: sistema que usa a contribuição igual de duas os mais pessoas;
- Modelo econômico: pagar de acordo com a qualidade de cada contribuição;
- Tarefas defensivas: tarefas nas quais trapacear é mais difícil que realizar a tarefa;
- Redundância: disponibilizar uma mesma tarefa várias vezes, obtendo vários resultados e, por um esquema de votação, eleger a melhor. É útil também para saber quem são os usuários que contribuem mal.;
- Implantação de dados verdadeiros: inserir um pequeno número de tarefas para as quais os resultados já são conhecidos, a fim de identificar usuários com contribuições pobres;
- Filtro estatístico: filtrar ou agregar os dados e descobrir *outliers* para serem removidos;
- Revisão em vários níveis: um grupo da multidão realiza a tarefa e outro

grupo realiza a revisão;

- Checagem automática: alguns problemas específicos tem a verificação de uma dada solução facilmente verificada por computador, porém descobrir sua solução não.

O ESP Game utiliza a concordância forçada para que uma palavra seja aceita como uma descrição da palavra, além da redundância, ao exigir que a palavra seja aceita várias vezes até se tornar uma descrição da imagem.

2.2.2.5. Visibilidade de outras contribuições

O acesso dos indivíduos indica o nível de visibilidade que um contribuinte pode ter sobre a contribuição do outro (GEIGER *et al.*, 2011). Quatro são os tipos de visibilidade possíveis:

- Nenhuma: nenhum usuário pode visualizar a contribuição de outro;
- Visualizar: um usuário pode apenas visualizar a contribuição de alguém;
- Avaliar: o usuário pode avaliar a contribuição de outra pessoa;
- Modificar: o usuário pode alterar a contribuição de outro.

O ESP Game não permite que nenhum usuário veja a contribuição do outro, enquanto o iStockphoto permite a avaliação das fotografias. Por sua vez, a Wikipedia (WIKIPEDIA, 2013) permite a alteração da contribuição dos outros usuários.

2.2.2.6. Motivação

Para que os usuários possam contribuir com os sistemas de *crowdsourcing*, é necessário que eles se sintam motivados. Para tal existem diversas maneiras, elucidadas a seguir.

Pagamento

Recompensa financeira, provavelmente, é o modo mais fácil de conseguir com que uma multidão colabore, e também um dos que mais incentiva a trapaça, especialmente quando os usuários são anônimos (QUINN & BEDERSON, 2011).

Como exemplo de aplicação que utiliza pagamento como forma de motivação podem ser citados o MTurk e o InnoCentive. O primeiro paga a cada usuário por tarefa completa, com preços normalmente na escala de centavos. O segundo paga apenas à melhor solução obtida da multidão. Pagamentos não são necessariamente em dinheiro, podendo ser feito na forma de cartões de presente ou dinheiro virtual em

jogos (QUINN & BEDERSON, 2011).

GEIGER *et al.* (2011) ainda define que as recompensas monetárias podem ser de dois tipos:

- Fixa: a remuneração é a mesma para cada contribuição de cada indivíduo, independente do valor agregado pela sua contribuição. Aplicada apenas a sistemas integrativos;
- Dependente do sucesso: cada contribuição tem um valor específico, dependente de quão boa ela é. iStockphoto e InnoCentive são exemplos de sistemas integrativos e seletivos, respectivamente, que utilizam a remuneração através do sucesso.

Diversão

Existem muitas atividades de entretenimento na internet, funcionando como simples passatempos (QUINN & BEDERSON, 2011). Ou seja, há uma predisposição das pessoas em jogar online, mesmo que sejam jogos simples. VON AHN (2005) se aproveitou dessa ideia para inventar vários tipos de jogos nos quais as pequenas tarefas realizadas pelos usuários são transformadas em dados úteis.

Jogos Com Propósito (*Games With a Purpose* - GWAP) é um tipo de jogo onde as pessoas participam, normalmente em troca de pontuação, colaborando com um sistema de *crowdsourcing* que existe por trás. As pessoas jogam porque é divertido, como um passatempo. Desenvolver um jogo que tenha um propósito é como desenvolver um algoritmo, que pode ter sua eficiência analisada e sua correção provada, onde a dificuldade está em transformar a tarefa desejada em um jogo divertido (VON AHN, 2006, QUINN & BEDERSON, 2011). Um exemplo no qual a recompensa é diversão é o ESP Game, além do Verbosity e o Peekaboom (VON AHN, 2005).

Implícita

É possível embutir em uma tarefa comum uma tarefa computacional, porém, é uma tarefa difícil (QUINN & BEDERSON, 2011). Imagine que, ao realizar uma tarefa corriqueira, esteja-se realizando uma tarefa de um sistema de *crowdsourcing*.

Essa forma pode ser entendida como uma não recompensa, pois é transparente ao usuário, que não obteve nenhum ganho ao realizar a tarefa. Um exemplo de aplicação cuja forma de recompensa é implícita é o reCAPTCHA (QUINN &

BEDERSON, 2009).

Altruísmo

Segundo o dicionário Michaelis (ALTRUÍSMO, 1998), altruísmo significa: "Amor ao próximo; abnegação, filantropia.". Um problema interessante e importante pode obter a ajuda de pessoas que não esperam nada em troca, a recompensa delas é simplesmente saber que ajudaram e contribuíram para um bem maior (QUINN & BEDERSON, 2009).

Um sistema com esse tipo de recompensa poderá ter um número limitado de participantes, pois algumas pessoas poderão ser atraídas ou não pelo problema. O GalaxyZoo tem como motivação mais comum por parte de seus usuários a vontade de colaborar com trabalhos científicos reais (FORTSON *et al.*, 2011).

Reputação/Atenção

Uma organização de prestígio é capaz de motivar pessoas a resolver um determinado problema apenas pelo fato da conquista, sem haver recompensas financeiras (QUINN & BEDERSON, 2011).

Um exemplo de aplicação que utiliza esse tipo de recompensa é o YouTube (YOUTUBE, 2013), onde a atenção, medida pelo número de visualizações do vídeo, é um motivador para que novos vídeos sejam submetidos ao site (YUEN *et al.*, 2011).

2.2.3. Aplicações

As subseções seguintes trazem vários exemplos de sistemas *crowdsourcing* citados na literatura, dos mais variados tipos, cada qual com um propósito particular.

2.2.3.1. InnoCentive

Em 2001, a companhia farmacêutica Eli Lilly financiou o lançamento do projeto InnoCentive. Sua intenção era conectar as pessoas de fora da companhia de forma a auxiliar o desenvolvimento de seus fármacos (HOWE, 2006b). InnoCentive é um exemplo de aplicação que faz uso de tarefas complexas, onde os usuários necessitam de conhecimentos específicos para solucionar os problemas.

Para que um usuário possa fazer parte da multidão do InnoCentive, ele precisa realizar um cadastro gratuito, com informações das áreas de interesse. Os mesmos não precisam ser profissionais ou acadêmicos da área. A submissão das soluções é feita de forma simples, através de um *upload* de um arquivo baseado em um padrão, na

maioria dos casos (BRABHAM, 2008).

O projeto não ficou restrito apenas a farmacêutica: NASA, Roche, Rockefeller Foundation, Prize4Life e Procter & Gamble são exemplos de outras organizações que se beneficiam do sistema. Estas organizações (chamadas de *seekers)* optam entre três diferentes tipos de desafio, de acordo com o site oficial: *Brainstorm*, *Premium* e *Grand*. O primeiro paga valores aos usuários (*solvers*) que variam entre US 500,00 e U$ 2.000,00, e são problemas da forma "faça você mesmo". O segundo, pagando valores de U$ 2.000,00 a U$20.000,00, possui ajuda de especialistas da área. O terceiro, pagando valores normalmente maiores que U$100.000,00, são problemas totalmente personalizados. Um exemplo deste último foi da organização sem fins lucrativos Prize4Life, pagando um milhão de dólares em 2006 para acelerar o desenvolvimento de uma ferramenta capaz de medir a progressão da esclerose lateral amiotrófica (INNOCENTIVE, 2012).

O importante no projeto InnoCentive é a meritocracia, uma vez que o importante é o resultado final. Os solucionadores não tem conhecimento de para qual organização estão tentando resolver um problema, assim como a organização não tem conhecimento de quem é o autor da solução até a mesma ser aprovada (HOWE, 2009).

2.2.3.2.iStockphoto

Notadamente, com o avanço da tecnologia, o custo de câmeras digitais, *smartphones* com câmeras integradas e até câmeras profissionais faz com que cada vez mais pessoas possam ter acesso a essas tecnologias, transformando-as em potenciais fotógrafas do cotidiano. Um mercado para fotógrafos amadores – estudantes, engenheiros, cozinheiros, médicos – é a proposta do iStockphoto: transformar o passatempo das pessoas em uma forma de ganhar dinheiro (HOWE, 2006b).

Para se tornar um fotógrafo, basta preencher um formulário e submeter três fotografias para que a equipe do iStockphoto possa julgar. Independente do conteúdo, se as fotos forem tecnicamente boas, o usuário é aceito como um fotógrafo, e poderá submeter fotos ao sistema, acompanhadas de uma descrição e uma lista de palavras-chaves (BRABHAM, 2008). As buscas por fotografias podem ser feitas através das palavras-chaves, categorias, colaboradores, tipo de arquivo e faixa de preço

(ISTOCKPHOTO, 2012).

O iStockphoto surgiu como comunidade antes de ser um negócio, quando seu fundador, Bruce Livingstone, criou um site onde ele e seus amigos pudessem compartilhar fotos entre si, para não precisarem pagar para os bancos de imagens. Devido ao grande volume de fotos, para pagar a banda de internet demandada, decidiu cobrar pelas fotos, formando o modelo do negócio como é atualmente. Porém, a comunidade continua fortemente enraizada, e encontros chamados iStockalypses – promovidos pela empresa – e Minilypses – promovidos pela própria comunidade – são realizados visando a integração e a colaboração dos iStockers, como são conhecidos os usuários (HOWE, 2009).

Em julho de 2006, o iStockphoto passou a receber, além de imagens, vídeos e a vendê-los em outubro do mesmo ano. Em 2009, foi lançado o iStockaudio, com o objetivo de fazer o mesmo em relação a arquivos de áudio. É possível indicar novos consumidores de arquivos para o site, sendo recompensado em dinheiro ou créditos assim que o indicado realizar uma compra (ISTOCKPHOTO, 2012).

2.2.3.3. Threadless

Após o concurso de um extinto fórum de arte online chamado *Dreamless*, dois amigos se inspiraram para criar um concurso contínuo de *designs* de camisas (THREADLESS, 2012). Todos os designs são votados em um período de sete dias por outros usuários da comunidade, podendo ser avaliado, ao final, com uma nota variando entre 0 e 5, indicando quais as estampas deverão ser impressas e colocadas a venda no site (THREADLESS, 2012).

Como forma de recompensa, os *designers* que tiveram suas estampas de camisas selecionadas e colocadas à venda recebem U$ 2000,00 em dinheiro e U$ 500,00 em vale-presente, além de créditos de U$ 1,50 por cada foto que o usuário enviar na qual esteja utilizando uma camisa da Threadless e U$ 3,00 por indicar um amigo (HOWE, 2009). A companhia expandiu a gama de produtos a serem personalizados com design de usuários, incluindo capas para *iPhone*, mochilas e almofadas (THREADLESS, 2012). A forma de recompensa aos usuários da comunidade – como o próprio site se caracteriza – não é apenas o valor monetário, mas sim a reputação, atingida pelo reconhecimento do trabalho realizado em forma de elogios (HOWE, 2009).

2.2.3.4. Amazon Mechanical Turk

Lançado em Novembro de 2005, o MTurk é uma plataforma que permite a usuários distribuir o trabalho em microtarefas e delegá-las a uma multidão de usuários. Os usuários (*requesters*) quebram suas tarefas em HITs (*Human Intelligence Task* – Tarefas de Inteligência Humana), para que outros usuários (*workers* ou *turkers*) possam realizá-las (ROSS *et al.*, 2010).

As tarefas podem ser dos mais variados tipos, como transcrição de áudios, rotular imagens, processamento de linguagem natural e até mesmo para pesquisas com os usuários. Cada tarefa tem uma determinada duração estimada, normalmente na casa dos segundos, e cada uma tem um valor monetário associado, normalmente na casa dos centavos de dólares (ROSS *et al.*, 2010).

Diferente de outros tipos de aplicação, o MTurk não tem uma tarefa específica como base. Ele é uma plataforma genérica de *crowdsourcing*, onde qualquer pessoa pode ter qualquer tipo de tarefa realizada, além de não requerer de seus *turkers* nenhum talento especial, diferente de outros projetos como o InnoCentive e o iStockphoto (HOWE, 2006b).

2.2.3.5. ESP Game

O ESP Game é um jogo onde duas pessoas aleatoriamente escolhidas jogam ao mesmo tempo. O jogo consiste em uma tela mostrando uma mesma imagem para ambos os jogadores com o objetivo de escrever o que o outro irá escrever sobre a figura. Um jogador não possui qualquer informação sobre o outro. Assim que ambos digitarem a mesma palavra (não necessariamente ao mesmo tempo) a figura muda e o jogo continua. Ao digitarem a mesma palavra, uma concordância entre os dois é alcançada. Essa palavra torna-se então, uma descrição para aquela imagem (VON AHN & DABBISH, 2004).

Para que descrições diferentes sejam atribuídas a uma mesma imagem, a cada N acordos, essa palavra não poderá mais ser inserida, sendo mostrada pros jogadores ao lado da imagem. Uma vez que os computadores ainda não conseguem rotular imagens, esse processo se mostrou efetivo, tanto que a Google licenciou este jogo para melhorar a sua ferramenta de busca de imagens (VON AHN, 2009).

2.2.3.6.GalaxyZoo

Em 2007, o projeto GalaxyZoo foi criado a partir da exaustiva experiência de um aluno de PhD da *Oxford University*, Kevin Schawinski, em categorizar 50.000 imagens de galáxias, buscando galáxias elípticas para a criação de um banco de dados (COOK, 2011, FORTSON *et al.*, 2011). Com essa experiência, viu-se a necessidade de uma forma de classificação de galáxias que custasse menos tempo, e a resposta para tal problema foi encontrada: fazer com que a voluntários classificassem as galáxias (FORTSON *et al.*, 2011). Ao mesmo tempo, outra pesquisadora de Oxford estava planejando uma forma similar para classificar a rotação das galáxias espirais e, num encontro fortuito, souberam que os projetos se completavam, e uniram esforços para tal (FORTSON *et al.*, 2011).

O projeto possui fotos que os astrônomos ainda não categorizaram. O voluntário que deseja colaborar aprende como classificar as galáxias a partir de um tutorial. Em sua primeira versão, a partir da exibição da foto da galáxia, o voluntário tem que responder se a galáxia é elíptica ou espiral, e a direção dos seus braços (horário ou anti-horário) caso sejam do tipo espiral (FORTSON *et al.*, 2011). Na segunda versão, mais dados eram obtidos dos usuários, como a quantidade de braços (GALAXYZOO, 2012). A terceira versão, conhecida como GalaxyZoo: Hubble, tinha como objetivo entender a evolução das galáxias, com a adição de outras bases de dados muito maiores que a utilizadas nos projetos anteriores (FORTSON *et al.*, 2011). Em 2012, a quarta versão foi lançada, com imagens de uma nova câmera instalada no telescópio Hubble, com fotos ainda mais profundas do universo (GALAXYZOO, 2012).

2.2.3.7.reCAPTCHA

Criado por VON AHN *et al.* (2008), é uma forma de CAPTCHA (acrônimo para *Completely Automated Public Turing test to tell Computers and Humans Apart*) que auxilia, de forma implícita, a transcrição de livros. Duas palavras são exibidas para que o humano transcreva, uma delas o sistema sabe a transcrição e outra não. Essa última é retirada de algum livro ou figura, cujo computador não foi capaz de transcrevê-la. Dessa forma, a multidão transcreve livros de forma transparente.

2.3.Considerações Gerais

Essas características de *Crowdsourcing* são importantes para o presente trabalho. A diversidade é atingida até certo ponto, ao utilizar pessoas que não se conhecem, de origens diferentes, universidades e níveis de escolaridade diferentes, porém, a grande maioria dos usuários é da área de computação, justamente pelo interesse e por ser uma tarefa inerente a área. Entretanto, o nível de conhecimento dos usuários quanto a testes é bastante diversificado, como está descrito na seção 5.6. A independência é atingida da forma em que um usuário não sabe o que o outro fez ou como outro usuário avaliou. A descentralização, por sua vez, é obtida através da liberdade da criação dos casos de teste, com poucas restrições ao que o usuário pode fazer.

As tarefas desenvolvidas pelos usuários nesse trabalho são complexas, exigindo criatividade (não artística), com a ferramenta utilizada para os experimentos se comportando como uma plataforma. A agregação é da forma integrativa, formando uma coleção (suíte) de casos de teste. Nenhuma restrição é feita aos usuários que podem participar, porém, pode haver uma vantagem natural das pessoas com conhecimento em testes de software, ou da área de computação. A qualidade é feita através da revisão em vários níveis, onde usuários avaliam os casos de teste criados pelos outros usuários, caracterizando também a forma de visibilidade das outras contribuições. A motivação passa por vários tipos, incluindo uma recompensa, apesar de não financeira (barras de chocolate), o altruísmo, para auxiliar uma pesquisa, reputação, uma vez que uma classificação é gerada, encadeando uma disputa, e uma dose de diversão.

3.Teste de Software

Cada vez mais qualidade é exigida por parte dos softwares e, para garantir essa qualidade, teste é uma ferramenta muito utilizada. Teste consiste em uma atividade realizada com o objetivo de avaliar a qualidade do software, melhorando-a ao identificar defeitos e problemas que prejudiquem seu funcionamento (ABRAN & BOURQUE, 2004).

Para PRESSMAN (2010, p.530), teste é "[...] o processo de exercitar o software com a intenção de encontrar (e finalmente corrigir) erros"[10]. Ele ainda afirma que um bom teste é aquele que tem alta probabilidade de encontrar um erro, não é redundante e não deve ser nem muito simples nem muito complexo.

Uma famosa frase de DIJKSTRA (1970) diz que "[...] testes podem ser usados muito eficientemente para mostrar a presença de erros, mas nunca a ausência"[11].

MYERS (2004, p.6) define teste como "[...] o processo de executar um programa com a intenção de encontrar erros"[12], argumentando ainda que, por mais que sejam criativos e aparentemente completos, não podem garantir a ausência total de erros. Ele ainda diz que um erro é quando um programa não faz aquilo que o usuário final espera que ele faça.

Para ABRAN & BOURQUE (2004) e PRESSMAN (2010), teste de software é um elemento da verificação e validação, onde o primeiro é um conjunto de atividades que garante que o software realiza corretamente as funcionalidades para as quais foi especificado, e o segundo garante que o software atende as expectativas dos usuários.

Um caso de teste é uma declaração do que será testado, juntamente com as especificações de entrada para o teste e as saídas esperadas pelo sistema (SOMMERVILLE, 2008), onde o sucesso de um caso de teste está em mostrar que o programa não faz o que deveria fazer (MYERS, 2004). As entradas, muitas vezes entendidas como os dados inseridos através de um teclado, também são compostas por dados oriundos de bancos de dados ou arquivos, pelo o ambiente e pelo estado do

[10] Tradução do autor para: "[...] the process of exercising software with the intent of finding (and ultimately correcting) errors."

[11] Tradução do autor para: "[...] program testing can be used very efficiently to show the presence of bugs, but never to show their absence."

[12] Tradução do autor para: "[...] the process of executing a program with the intente of finding erros".

sistema durante a execução do teste, assim como as saídas não são apenas as exibidas no monitor, sendo compostas também por dados em arquivos, bancos de dados ou dispositivos, ou alterando o estado do sistema (COPELAND, 2004). JORGENSEN (2002) cita ainda as pré-condições para a execução de um caso de teste como um tipo de entrada do caso de teste.

MYERS (2004) estabelece 10 princípios para os testes de software:

i. A definição do resultado esperado é uma parte necessária de um caso de teste;
ii. Um programador deve evitar testar seu próprio programa;
iii. Uma organização não deve testar seus próprios programas;
iv. Inspecionar cuidadosamente os resultados de cada teste;
v. Casos de teste devem ser escritos tanto para condições de entrada inválidas e inesperadas, como estradas válidas e esperadas;
vi. Examinar um programa para verificar se ele não faz o que é proposto é apenas metade da batalha, a outra metade é ver se o programa faz o que não é proposto;
vii. Evitar casos de teste descartáveis, a menos que o programa também seja descartável;
viii. Nunca planejar um teste assumindo que nenhum erro será encontrado;
ix. A probabilidade de encontrar mais erros em uma seção de um programa é proporcional ao número de erros já encontrados na seção;
x. O teste é uma tarefa extremamente criativa e intelectualmente desafiadora.

No escopo deste trabalho, os termos software, aplicação e programa serão utilizados indistintamente, sendo diferentes do termo sistema, que será usado como um termo mais abrangente, sendo composto de software, hardware, sistema operacional, pessoas e dados.

3.1.Tipos

Para a construção bem sucedida de um software, é necessário que exista uma estratégia, composta por um planejamento e um roteiro de testes, definindo os passos a serem seguidos (PRESSMAN, 2010). Para JORGENSEN (2002), BEIZER (1990) e PFLEEGER & ATLEE (2009) existem três níveis de teste: unidade, integração e sistema. ABRAN & BOURQUE (2004) chamam esses níveis de alvos de teste. Para

(HETZEL, 1993) e COPELAND (2004), além destes três, existe ainda o teste de aceitação, o que PRESSMAN (2010) chama de teste de validação.

Além dos níveis supracitados, existe o que ABRAN & BOURQUE (2004) e PFLEEGER & ATLEE (2009) classificam como objetivos de teste. Entre eles estão os testes de funcionalidade, desempenho, aceitação e instalação, que nessa ordem compõem os passos para a realização do teste de sistema, segundo PFLEEGER & ATLEE (2009). ABRAN & BOURQUE (2004) vão mais além, citando testes de regressão, estresse, recuperação e usabilidade, não tratando esses objetivos como tipos de teste de sistema, nem impondo uma ordenação, mencionando que esses objetivos podem ser aplicados a qualquer um dos três níveis de teste (unidade, integração e sistema).

O ato de planejar todos os testes antes de qualquer desenvolvimento é um questionamento existente, o que COPELAND (2004) chama de paradigmas de teste, classificando em dois tipos, testes exploratórios ou teste com *scripts*[13].

A discussão sobre uma taxonomia de teste de softwares está fora do escopo desse trabalho, portanto, todos os tipos, níveis, estratégias ou paradigmas mencionados serão discutidos nesta seção sem distinção. Porém, ao mencionar "três níveis de teste", será em referência aos testes de unidade, integração e sistema.

3.1.1. Teste de Unidade

Uma unidade é o menor pedaço do software que um desenvolvedor consegue criar, geralmente armazenado em um único arquivo (COPELAND, 2004). Teste de unidade são aqueles que visam demonstrar erros nessas unidades do projeto, focando a lógica interna de processamento e as estruturas de controle que existem dentro do limite do componente (PRESSMAN, 2010). Para diferentes arquiteturas de software, o alvo destes testes pode variar desde pequenos subprogramas a grandes componentes compostos de unidades fortemente acopladas (ABRAN & BOURQUE, 2004).

Software convencional

Em sistemas convencionais, o foco é em pequenos blocos do software, como subprogramas, sub-rotinas ou procedimentos individualmente (MYERS, 2004). COPELAND (2004) argumenta que a unidade varia de acordo com a linguagem de

[13] Tradução para o termo *scripted testing* obtida no Glossário Padrão de Termos Utilizados em Teste de Software (2012).

programação utilizada. Por exemplo, em C, a unidade seria uma função, em C++ ou Java, linguagens orientadas a objetos, a classe seria a unidade.

Software OO

Em sistemas orientados a objetos, o foco normalmente é uma classe encapsulada, onde os métodos são testados dentro do escopo de cada classe, especialmente nas que herdam e/ou sobrescrevem métodos, onde os atributos e métodos usados são o da própria classe (PRESSMAN, 2010). JORGENSEN (2002) compara as metodologias de considerar classes ou métodos como as unidades de teste.

3.1.2.Teste de Integração

Ao unir unidades, subsistemas são formados, até que se forme o sistema como um todo. Após os testes de cada componente individual, pode-se argumentar que não existem erros no software, porém, muitos erros são associados às interfaces de comunicação entre os componentes (PRESSMAN, 2010). COPELAND (2004) cita um exemplo simples, onde uma função funciona corretamente, porém, quando é chamada por outra passando como argumento um número inteiro, uma falha é encontrada, pois a primeira função espera um número real como parâmetro. Assim como os testes de unidade, essa estratégia de teste varia de acordo com a arquitetura utilizada.

Software convencional

Para atingir o objetivo de integrar os componentes, duas abordagens podem ser utilizadas, a incremental e a não incremental. A abordagem não incremental, também conhecida como *big-bang*, integra todos os componentes ao final de toda a codificação dos mesmos (MYERS, 2004).

A incremental, por sua vez, possui três estratégias: *bottom-up*, *top-down* e sanduíche (JORGENSEN, 2002, MYERS, 2004, PFLEEGER & ATLEE, 2009, PRESSMAN, 2010). A primeira parte das unidades do nível mais baixo da hierarquia do software, codificados e testados individualmente, e segue subindo para os componentes dos níveis mais altos da hierarquia, que ainda não foram codificados. Por isso existe a necessidade da construção de *drivers*, que são pequenas rotinas que fazem chamadas aos componentes, passando um caso de teste a eles (PFLEEGER & ATLEE, 2009). A segunda utiliza a estratégia inversa, começando com o programa principal (a raiz da árvore hierárquica), depois descendo nível por nível. Como os

níveis mais baixos ainda não foram codificados, há uma necessidade de codificação de *stubs*, que são pedaços de código que simulam o comportamento do componente chamado (JORGENSEN, 2002). A terceira é uma combinação das duas primeiras, onde uma abordagem *top-down* é feita nos componentes localizados no nível mais alto da hierarquia e *bottom-up* nos níveis mais baixos (PFLEEGER & ATLEE, 2009).

As vantagens da forma não incremental são: (i) exigir menos tempo de execução da máquina e (ii) possibilitar mais atividades em paralelo, onde testes de unidades podem ser realizados ao mesmo tempo. Por outro lado, as vantagens da forma incremental são: (a) requer menos trabalho, uma vez que a forma não incremental necessita de codificação de *stubs* **e** *drivers* e a incremental apenas de *drivers* **ou** *stubs*, (b) detecção de erros mais cedo, (c) por detectar o erro mais cedo, realizar o *debug*, apontar e consertá-los se torna mais fácil, dado o número menor de módulos integrados, (d) testes incrementais tendem a ter resultados mais completos, pois o teste das unidades é repetido em cada contexto de integração (MYERS, 2004).

Software OO

Para projetos orientados a objetos, a estrutura óbvia de controle hierárquico não está presente, logo, as formas *bottom-up* e *top-down* não são aplicáveis. Sendo assim, outras duas formas de realização deste tipo de teste podem ser aplicadas: *thread-based* e *use-based* (PRESSMAN, 2010). O polimorfismo, a herança e o encapsulamento são aspectos que tornam o teste de integração de sistemas orientados a objetos mais extenso (PFLEEGER & ATLEE, 2009).

A abordagem *thread-based* conduz os testes de forma que a integração das classes é feita para atender a realização de uma entrada ou evento no software, onde cada *thread* é integrada e testada individualmente, ou seja, uma funcionalidade e testes são reexecutados após cada nova integração (testes de regressão, ver seção 3.1.6). A outra abordagem, *use-based*, começa testando as classes independentes, ou seja, classes que utilizam nenhuma ou poucas outras classes. Após essas classes, classes dependentes são testadas sucessivamente, primeiro as que utilizam apenas as classes independentes, depois as que utilizam classes dependentes já testadas, e assim sucessivamente (PRESSMAN, 2010).

3.1.3. Teste de Sistema

Diferente dos testes de unidade e de integração, que possuem como objetivo

assegurar que o código foi desenvolvido corretamente, o teste de sistema visa assegurar que o sistema realiza as ações para o qual foi especificado (PFLEEGER & ATLEE, 2009). Por esse motivo, testes de sistema não são afetados pela forma como o software é desenvolvido, orientado a objetos ou não (JORGENSEN, 2002).

Esses tipos de testes visam encontrar erros inerentes ao comportamento de todo o sistema e não apenas ao software, sendo apropriado para verificar a compatibilidade entre o sistema e os requisitos não funcionais, como segurança e desempenho, juntamente com as interações com os outros componentes do sistema: hardware, sistema operacional, pessoas e dados (ABRAN & BOURQUE, 2004).

Para alguns autores, este nível de teste abriga outros tipos de teste, por exemplo, teste de funcionalidade, usabilidade, segurança, internacionalização, localização, confiabilidade, desempenho, recuperação e configuração (BEIZER, 1990, COPELAND, 2004, PFLEEGER & ATLEE, 2009).

Para MYERS (2004), esta estratégia de teste deve ser realizada por pessoas de fora da organização que desenvolveu o software, por motivos de incompatibilidade de objetivos, quando a organização que desenvolveu o software deseja provar que o sistema funciona, enquanto o objetivo do teste é demonstrar que não funciona.

3.1.4. Teste de Aceitação

Ninguém melhor que o usuário final para avaliar se o software atende às expectativas e às necessidades para o qual foi demandado. Para tal finalidade, o teste de aceitação existe. O objetivo deste tipo de teste é encontrar incompatibilidades entre as necessidades atuais do usuário e as funções que o sistema realiza (MYERS, 2004). Para que esta verificação seja fidedigna, os testes são sempre de responsabilidade do usuário do software, sendo executado tanto ao final, com o software totalmente desenvolvido, ou durante o desenvolvimento, com a parte que está liberada para teste (BASTOS *et al.*, 2007).

Para softwares que serão utilizados por uma variedade de clientes, dois tipos de testes de aceitação são comumente efetuados: teste alfa e teste beta (PFLEEGER & ATLEE, 2009). O primeiro é realizado nas instalações do desenvolvedor, com os usuários finais sendo observados pelos próprios desenvolvedores, com os últimos registrando os erros encontrados. O segundo é realizado nas instalações do usuário final, sem a presença do desenvolvedor, e o responsável pelos relatos dos erros são os

próprios usuários (PRESSMAN, 2010).

3.1.5. Teste de Funcionalidade

Teste de funcionalidade é aquele baseado nos requisitos funcionais do sistema, buscando encontrar discrepâncias entre os requisitos e as funcionalidades executadas pelo programa (MYERS, 2004, PFLEEGER & ATLEE, 2009).

SOMMERVILLE (2008) descreve que a melhor abordagem para este tipo de teste é a baseada em cenários, onde cada cenário criado possuirá vários casos de teste relacionados. Para sistemas que possuem seus requisitos na forma de casos de uso e diagrama de sequência, essas especificações poderão ser usadas como base para a criação dos casos de teste de funcionalidade, que poderão ser executado para cada incremento em sistemas desenvolvidos de forma ágil.

Também conhecido como *thread-testing*, deve ser realizado em um ambiente cuidadosamente controlado (PFLEEGER & ATLEE, 2009), e normalmente é executado utilizando uma técnica caixa-preta (ver seção 3.2.1) (MYERS, 2004).

3.1.6. Teste de Regressão

Teste de regressão é uma estratégia de repetição de testes, realizada depois da inserção de uma nova funcionalidade ou da correção de alguma falha, visando demonstrar que o comportamento do software anteriormente funcional e testado não foi afetado, exceto quando a mudança era o objetivo (JORGENSEN, 2002, PRESSMAN, 2010). Teste de regressão pode ser executado em qualquer nível de teste de software, sendo geralmente aplicado depois de uma mudança significativa no software (MILLS, 2008).

Em outras palavras, ao inserir um novo incremento, por exemplo, a reprodução dos testes anteriores de integração é importante, assim como novos testes relacionados a nova funcionalidade, buscando saber se o erro encontrado foi no novo ou nos antigos incrementos, sendo impraticável sem algum apoio automatizado (SOMMERVILLE, 2008). Para PRESSMAN (2010), a prática manual deste tipo de teste também é possível, porém para um subconjunto dos casos de teste do software.

A importância dos testes de regressão se deve ao fato de que modificar um programa existente é um processo mais propenso a erros que a codificação de um novo (MYERS, 2004).

3.1.7.Teste com Script

Um teste com *script* é um passo-a-passo a ser seguido pelos testadores, contendo as operações e os resultados esperados, que pode ser realizado tanto manualmente como de forma automatizada, tanto a execução quanto a comparação dos resultados (KANER, 2002, 2008).

COPELAND (2004) diz que nenhuma frase resume melhor a abordagem de testes com script que a seguinte:

> "Planeje sua execução, execute seu plano."[14]

A utilização dos *scripts* traz os seguintes benefícios: (i) a possibilidade de pensar no design de cada teste, aperfeiçoando-os para os atributos importantes, (ii) a divisão do trabalho em planejamento, modelagem, codificação e execução dos casos de teste, (iii) a revisão por outros testadores, (iv) a reutilização, (v) fácil automatização e (vi) pode ser realizado por pessoas que não possuem conhecimento do sistema, devido ao alto nível de detalhamento, (vii) além de comporem um conjunto de testes, com o qual é possível saber a porcentagem dos testes realizados. (COPELAND, 2004, KANER, 2008)

Por outro lado, algumas desvantagens também podem ser citadas, como: (a) por ser inflexível, a perda das mesmas informações sempre que um mesmo *script* é executado, (b) ao desenvolver um conjunto de testes no início de um projeto, estes podem se tornar falhos devido à dependência dos requisitos, que podem não estar completos ou podem sofrer alterações ao longo do projeto, (c) o ambiente que roda o software também muda, (d) o entendimento do software é menor no início do projeto (COPELAND, 2004, KANER, 2008).

3.1.8.Teste Exploratório

BACH (2002) define teste exploratório como "[...] aprendizado, design do teste, e execução do teste simultaneamente"[15]. Ou seja, qualquer teste onde o testador modela o teste conforme o teste é executado, utilizando informações obtidas durante o processo para elaborar melhores testes. Sendo assim, pode ser considerado o oposto dos testes com *script*, que são testes pré-definidos, sendo as duas abordagens

[14] Tradução do autor para: "Plan your work, work your plan."

[15] Tradução do autor para: "[...] simultaneous learning, test design, and test execution."

completamente compatíveis, onde testes exploratórios podem surgir ao executar um *script* de teste manualmente (BACH, 2001).

É um estilo de teste de software que depende do testador, para aperfeiçoar continuamente o valor do seu trabalho, enfatizando a liberdade e responsabilidade pessoais, acontecendo em paralelo ao projeto, como atividade de apoio mútuo (KANER, 2008). A eficácia deste tipo de teste depende do conhecimento do testador, oriundo da familiaridade com a aplicação, da plataforma, dos tipos de falhas possíveis e da observação do comportamento do produto (ABRAN & BOURQUE, 2004).

Testes exploratórios são úteis quando há uma necessidade de uma rápida resposta, quando os requisitos são vagos ou inexistentes, quando uma falha é descoberta e deseja-se saber o tamanho, escopo e as variações da falha, e é um bom adicional aos testes com *script*, quando esses últimos já foram executados em exaustão e não encontram mais nenhum erro. Por outro lado, como os testes com *script* começam mais cedo – na fase dos requisitos – falhas são encontradas antes, em comparação com testes exploratórios (COPELAND, 2004). Estudos mostram que não há evidências que um dos dois tipos de teste – exploratório e com scripts – seja melhor que o outro (ITKONEN *et al.*, 2007, PRAKASH & GOPALAKRISHNAN, 2011).

3.2.Técnicas

Existem duas técnicas largamente estudadas na literatura para testar um software, onde a primeira utiliza o conhecimento do interior das operações e a segunda utiliza o conhecimento das especificações das funções do produto (BEIZER, 1990, HETZEL, 1993, JORGENSEN, 2002, COPELAND, 2004, MYERS, 2004, SOMMERVILLE, 2008, PRESSMAN, 2010). O primeiro, chamado caixa-branca, busca examinar os caminhos lógicos internos do software. O segundo, chamado caixa-preta, busca examinar aspectos fundamentais do sistema, sem se preocupar com a lógica interna. Eles não são mutuamente exclusivos, pelo contrário, são abordagens complementares, que provavelmente descobrirão classes distintas de erros (MYERS, 2004, PRESSMAN, 2010). Além dessas duas técnicas, existe uma terceira técnica híbrida chamada caixa-cinza, que mistura as duas abordagens, ou seja, utiliza as especificações e o conhecimento interno da estrutura do software.

3.2.1.Caixa-preta

Conhecido também como teste comportamental ou funcional, os testes caixa-preta tem como base a visão de que o programa como um todo pode ser considerado como uma função que mapeia valores de entrada nos valores de saída, sendo os requisitos e a especificação do software as únicas informações a serem usadas na elaboração dos casos de teste (JORGENSEN, 2002, COPELAND, 2004).

Duas vantagens podem ser enumeradas: a primeira é a independência de como o software foi desenvolvido, onde os casos de testes podem ser reaproveitados mesmo que um novo sistema seja desenvolvido em uma nova linguagem ou de outra forma (por exemplo, orientada a objetos), e a segunda é que eles podem ser executados em paralelo com o desenvolvimento, descobrindo os defeitos mais cedo.

Por outro lado, existem desvantagens na utilização deste tipo de teste, uma delas é que alguns caminhos do software podem nunca ser exercitados, outra é que os testes podem ser redundantes (JORGENSEN, 2002). Além dessas, para encontrar todos os erros, todas as possíveis combinações de entrada e saída, válidas e inválidas, teriam que ser testadas, o que quase sempre impossível, tornando-se ainda pior quando utiliza um banco de dados ou arquivo, em que as informações provenientes desses últimos também se caracterizam como entradas (COPELAND, 2004, MYERS, 2004).

Particionamento de equivalência

Uma forma de conduzir testes caixa-preta é chamada de particionamento de equivalência ou classe de equivalência, onde o domínio das entradas é dividido em classes. Assume-se que o teste de um valor representativo da classe é equivalente ao teste de qualquer outro valor da classe, e valores válidos e inválidos são utilizados (MYERS, 2004). Por exemplo, se a entrada é uma hora (0 até 23), 1 é uma entrada válida e 25 é uma entrada inválida

Análise de valor limite

Outra técnica caixa-preta é a análise de valor limite, que, como o nome traduz, utiliza os valores mais próximos aos limites do domínio das entradas. É nos valores limites que os erros muitas vezes se escondem (JORGENSEN, 2002, COPELAND, 2004). Por exemplo, se uma entrada deve ser um valor inteiro entre a e b, testam-se valores muito próximos de seus limites, como a-1, a, a+1, b-1, b e b+1.

Tabela de decisão

Tabela de decisão é uma forma de representar e analisar relações lógicas complexas, ideais para situações onde ações são tomadas baseadas em condições, que podem ser das mais variadas combinações (JORGENSEN, 2002). Cada coluna da tabela representa uma regra lógica, composta por condições e as ações que serão tomadas. Para ficar mais claro, a Tabela 1 ilustra um exemplo de uma tabela de decisão dos documentos requeridos para embarque em um voo doméstico para uma unidade federativa diferente da atual, no Brasil.

Tabela 1 - Exemplo de tabela de decisão

Condições	**Regra**			
	1	**2**	**3**	**4**
Idade?	>=12	<12	<12	<12
Acompanhado de parente até o 3º grau?	-	Sim	Não	Não
Acompanhado de adulto sem parentesco ou parentesco maior que 3º grau?	-	-	Sim	Não
Ações				
Apresentar documento de identificação com foto?	Sim	Sim	Sim	Sim
Apresentar autorização do pai, mãe ou responsável?	Não	Não	Sim	Não
Apresentar autorização judicial?	Não	Não	Não	Sim

Legenda: células com conteúdo igual a "-" significam que a condição é indiferente.

3.2.2.Caixa-branca

O objetivo desse tipo de teste, também chamado de teste estrutural, teste caixa-clara[16] ou teste caixa de vidro[17], é encontrar erros ao elaborar casos de testes que percorram todo o código, necessitando de um conhecimento da estrutura, dos caminhos internos e da codificação do software (SOMMERVILLE, 2008).

Com os testes caixa-branca é possível criar casos de teste que percorram: todos os caminhos, todas as decisões lógicas, todos os ciclos e todas as estruturas internas (PRESSMAN, 2010). A técnica pode ser aplicada a qualquer um dos três níveis de teste (unidade, integração e sistema), apesar de ser geralmente equiparada a testes de unidade, uma visão correta, porém limitada (COPELAND, 2004).

Os testes utilizados nos softwares convencionais podem ser aplicados também aos softwares OO, nos métodos das classes, porém esse esforço pode ser mais bem aproveitado se aplicado ao nível da classe (PRESSMAN, 2010).

[16] Tradução para o termo *clear-box*, obtida no Glossário Padrão de Termos Utilizados em Teste de Software (2012).

[17] Tradução para o termo *glass-box*, obtida no Glossário Padrão de Termos Utilizados em Teste de Software (2012).

Como ponto de partida para os testes caixa-branca, um grafo do fluxo de controle (*Control flow graph*) é construído a partir do código do software. Esse grafo é direcionado, onde os nós representam as instruções (ou blocos de instruções) e as arestas representam o fluxo entre elas (JORGENSEN, 2002). Blocos de instruções são um conjunto de sentenças sequenciais não interrompidos por uma decisão – ponto onde o fluxo é dividido – ou junção – ponto onde o fluxo é unificado (BEIZER, 1990). A Figura 2 (a) exibe um trecho do código fonte Java de um programa para verificar se três números podem formar um triângulo e o tipo do mesmo, enquanto a Figura 2 (b) mostra o grafo do fluxo de controle, onde a numeração dos nós corresponde a linha do código Java.

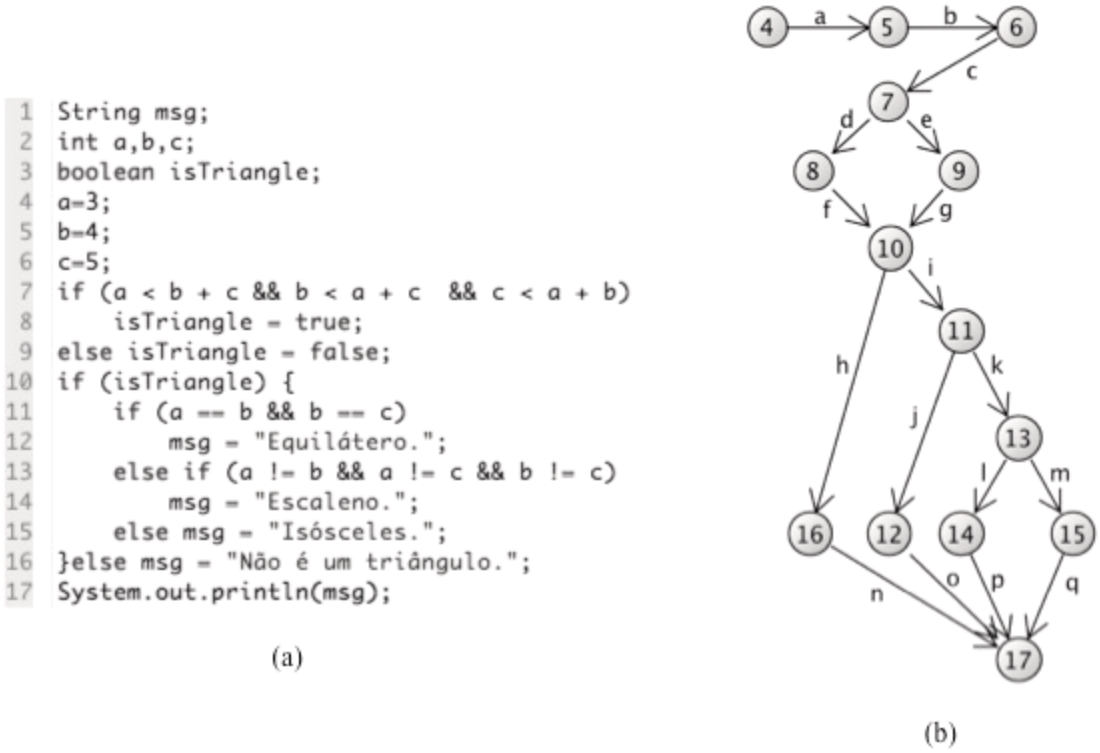

Figura 2 - Exemplo de código e grafo do fluxo de controle

Teste de Caminho (Path testing)

Teste de caminho é uma família de técnicas de testes que selecionam um conjunto de caminhos do programa a serem exercitados (BEIZER, 1990). Um caminho é uma sequência de nós a serem percorridos, entre o nó inicial e o nó final. Casos de testes são gerados a fim de exercitar cada caminho escolhido, de acordo com o critério de cobertura utilizado.

A cobertura de sentença (*statement coverage*) busca exercitar cada sentença ao

menos uma vez, o que em termos do grafo significa passar por cada nó. A complexidade ciclomática[18] do grafo define um limite superior do número de testes para garantir que todos os nós foram percorridos (PRESSMAN, 2010). Para o exemplo da Figura 2 (b), este número é V(G) = 17 - 14 + 2x1 = 5.

A cobertura de desvio (*branch coverage)*, também chamada de cobertura de decisão (*decision coverage*) estabelece que haja testes suficientes para as saídas verdadeira e falsa de cada decisão, ou seja, cada aresta deve ser percorrida ao menos uma vez. Satisfazer a cobertura de desvio geralmente satisfaz a cobertura de sentença (MYERS, 2004).

A cobertura de condição (*condition coverage*) é um pouco diferente da cobertura de desvio, buscando exercitar cada condição de uma decisão. Ao invés testes para cada saída verdadeira e falsa de cada decisão, haverá testes para cada saída verdadeira e falsa de cada condição em cada decisão. Por exemplo, o nó 7 seria testado para cada uma das três condições (COPELAND, 2004, MYERS, 2004).

Considere a Figura 3 (a). A cobertura de decisão é satisfeita com os dois casos de teste (x=verdadeiro, y=verdadeiro => V; e x=verdadeiro, y=falso => F), mas esses testes não satisfazem a cobertura de condição, pois x não é exercitado como falso. Para os testes (x=verdadeiro, y=falso => F; e x=falso, y= verdadeiro => F) a cobertura de condição é satisfeita, porém a cobertura de decisão não, pois x && y não resulta em verdadeiro. Para solucionar esse dilema, a cobertura de condição/decisão requer testes suficientes para satisfazer ambas as coberturas, nesse caso, os três citados.

Considerando que os compiladores interpretam múltiplas condições (Figura 3 - a) como condições aninhadas, um novo grafo do fluxo de controle (Figura 3 - b) pode ser redesenhado. Para a cobertura de condição múltipla (*multiple condition coverage*), todas as combinações entre as variáveis x e y deverão ser testadas, como em uma tabela verdade, ou seja, serão quatro testes: x=V, y=V; x=V, y=F; x=F, y=V; y=F, y=F. A cobertura de condição múltipla garante a cobertura de decisão, de condição e de condição/decisão (COPELAND, 2004).

[18] Complexidade ciclomática é uma propriedade de um grafo, e corresponde ao número de regiões, podendo ser obtida através da fórmula V(G) = e – n + 2p, onde *e* representa o número de arestas, *n* o número de nós e *p* o número de componentes conectados (JORGENSEN, 2002).

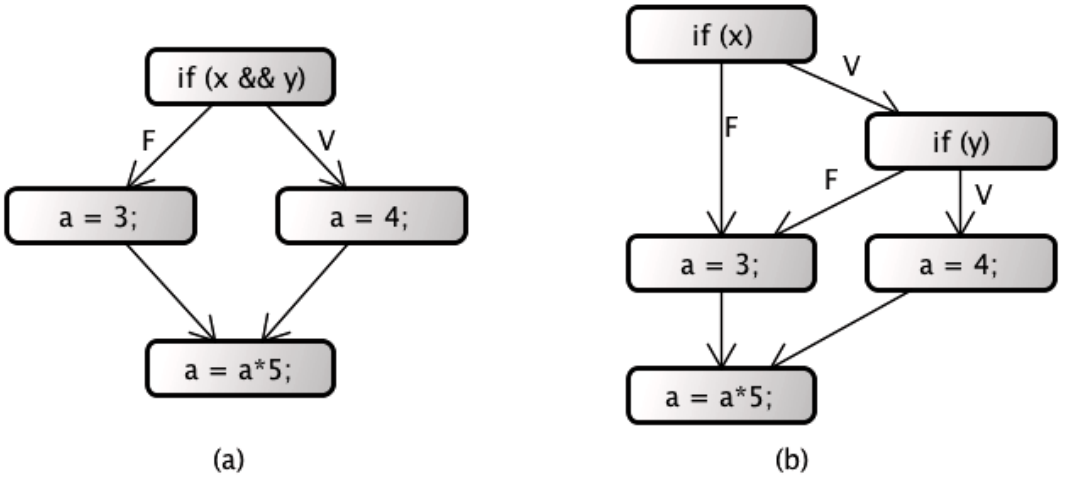

Figura 3 - Grafo do fluxo de controle em uma condição múltipla

A cobertura de caminho (*path coverage*) estipula que deve haver um teste para cada caminho possível, onde cada decisão simples dobra a quantidade de caminhos e cada laço multiplica pelo número possível de iterações, além da passagem direta pelo mesmo (BEIZER, 1990). Deve-se ressaltar que nem todos os caminhos são viáveis, por terem condições contrárias, como o caminho <4,5,6,7,9,10,11,12,17> da Figura 2, pois a passagem pelo nó 9 significa que não é um triângulo, logo, o único caminho possível a partir do nó 10 é o nó 16.

Quando um código possui um laço, o número de caminhos pode se tornar grande o suficiente para tornar a cobertura de caminho impraticável. Para resolver isso a cobertura de laço (*loop coverage*), de forma semelhante a cobertura de desvio, define que hajam testes para entrar e para passar direto pelo laço. Indo mais além, deve haver testes que considerem o número de vezes que o laço é percorrido, testando para os valores limites, como o mínimo, mínimo-1, mínimo+1, máximo, máximo-1, máximo+1 e um intermediário (BEIZER, 1990, JORGENSEN, 2002, COPELAND, 2004).

Teste de fluxo de dados (Data flow testing)

Diferentemente do que o nome sugere, teste de fluxo de dados não possuem conexão com diagramas de fluxo de dados (JORGENSEN, 2002). Assim como o teste de caminho, é um conjunto de técnicas que reside em caminhos do programa, onde as sentenças são relacionadas às variáveis, também utilizando grafos do fluxo de controle (BEIZER, 1990).

As instruções podem ser de vários tipos: definição (d), onde variáveis são definidas ou modificadas; destruição (k), onde as variáveis são destruídas; e uso (u), onde o valor da variável é usado, tanto em um cálculo (c), quanto em um predicado/condição (p).

Uma forma de selecionar as sequências de instruções a serem testadas é a *all-du-paths*, ou seja, cada caminho a partir de cada definição para cada variável até cada uso daquela definição (BEIZER, 1990).

Uma forma reduzida é a de testar todos os usos (*all-use*), onde para cada variável ao menos um caminho limpo a partir da definição até cada uso (JORGENSEN, 2002).

Outro meio ainda menos abrangente é o de testar as definições (*all-definition*), onde todas as definições de cada variável devem ser testadas ao menos uma vez, com algum uso (BEIZER, 1990).

3.2.3.Caixa-cinza

Testes caixa-preta olham para o software com uma visão do usuário, enquanto testes caixa-branca utilizam a visão do desenvolvedor, e cada uma é melhor identificando certos tipos de erros, tornando-as complementares. Incorporando elementos de ambas, como o resultado final exibido para o usuário, conhecimento técnico específico do software e do ambiente, os testes caixa-cinza são bastante eficazes nos testes das aplicações Web, por essas possuírem uma arquitetura com vários componentes (NGUYEN, 2003).

Para COPELAND (2004), a ‘caixa’ representando o código é estudada até que sua lógica seja compreendida (caixa-branca), então é fechada e testes caixa-preta mais efetivos podem ser elaborados utilizando esse conhecimento. Por outro lado, para KANSKAR & SALUJA (2012), o conhecimento do software utilizado nos testes caixa-cinza são outros que não o código fonte, por exemplo, a estrutura do banco de dados.

NGUYEN (2003) argumenta que a vantagem dessa técnica consiste em revelar falhas que não são tão fáceis nas duas outras técnicas, especialmente problemas de fluxo de informação ponta-a-ponta e de configuração de todo o sistema distribuído e sua compatibilidade.

3.3. Avaliação dos testes

Um grande número de casos de teste não implica em testes bem feitos, pois os mesmos podem ser redundantes. Por isso, é necessário estabelecer formas de mensurar a qualidade dos testes. Essa seção ilustra duas formas dentre as mais utilizadas em testes de software Web (GAROUSI *et al.*, 2013a).

3.3.1. Critério de cobertura

Como visto na seção 3.2.2, os testes caixa-branca utilizam um grafo para construir os casos de teste, e vários critérios de cobertura são utilizados. Esses critérios buscam garantir que haja casos de testes suficientes para exercitar: todos os caminhos, todas as sentenças, todas as decisões, *all-du-paths* ou outros.

Para os testes caixa-preta, esses critérios de cobertura podem fornecer um rastreamento de quais instruções, decisões e caminhos foram percorridos durante os testes. De posse desses dados, é possível identificar lacunas e redundâncias nos testes, respectivamente quando caminhos não são percorridos ou percorridos muitas vezes (JORGENSEN, 2002). A proporção do código testado também pode ser obtida, por exemplo, se 4 dos 8 caminhos possíveis foram testados, existe uma cobertura de 50% (PFLEEGER & ATLEE, 2009). Esta proporção serve como medida para avaliação dos casos de teste.

3.3.2. *Fault Seeding*

Outra técnica para estimar o número de falhas presentes em um programa é a chamada *fault seeding*. Consiste em um dos membros do grupo de teste inserir propositalmente um número conhecido de falhas no código. Em seguida, os outros membros do grupo de teste executam os testes a fim de localizar o maior número possível de falhas. A razão do número dessas falhas não localizadas pelo total de falhas inseridas fornece uma estimativa da proporção de quantas falhas reais ainda existem (PFLEEGER & ATLEE, 2009). O número total de falhas reais existentes é desconhecido, e pode ser obtido através da expressão $N = Sn/s$, onde N é o total de falhas reais, S é o total de falhas inseridas, n é o número de falhas reais encontradas e s é o número de falhas inseridas detectadas. O histórico de falhas em softwares anteriores pode funcionar como uma base para derivar as falhas.

Uma técnica que vem sendo cada vez mais estudada é o teste de mutação, que

consiste em gerar automaticamente versões mutantes do software, onde pequenos erros são inseridos no código, ao remover, alterar ou inserir operadores matemáticos ou lógicos, por exemplo (JIA & HARMAN, 2011). A técnica, porém, possui dois problemas: a dificuldade em identificar quando dois mutantes são equivalentes, e o grande custo computacional, pois os testes devem ser executados para cada versão mutante do software (MIRSHOKRAIE *et al.*, 2013). Entretanto, ANDREWS *et al.* (2006) observou que falhas inseridas manualmente são mais difíceis de serem detectadas que as falhas inseridas nos mutantes.

3.4.Ferramentas de teste

O uso de ferramentas de automação de testes pode diminuir bastante parte do trabalho penoso do processo (MYERS, 2004). Por isso e por ser uma fase trabalhosa do processo de software, as ferramentas para automação de teste estão entre as primeiras ferramentas de software a serem desenvolvidas (SOMMERVILLE, 2008).

Existem ferramentas de automação de software de diversos tipos, com vários objetivos, por exemplo, o teste da segurança e do desempenho do software. A seguir, alguns tipos de ferramentas de teste com ênfase nos testes de funcionalidade citados na literatura (NAIK & TRIPATHY, 2008, SOMMERVILLE, 2008):

i. Analisador estático: buscam identificar caminhos para os testes, baseados nos critérios descritos na seção 3.2.2, sendo dependentes da linguagem utilizada na construção do software;
ii. Analisador dinâmico ou de cobertura: medem a cobertura do código, rastreando e registrando quais caminhos foram percorridos durante a execução dos testes unitários, informando quais pedaços do código não foram testadas, ou seja, onde novos testes deverão focar;
iii. Gerenciador de testes: gerencia a execução e o acompanhamento dos testes unitários dinâmicos e os respectivos resultados, onde algumas mais avançadas conseguem comparar o resultado atual com o esperado, como o JUnit (JUNIT, 2012);
iv. Gerador de dados de teste: geram uma variação de dados a partir de banco de dados, aleatoriamente a partir de padrões ou utilizando classes de equivalência e os valores limítrofes para serem utilizados nos testes;
v. *Record/Playback*: gravam a interação do testador com a aplicação a ser

testada, para depois repetir as ações, como o Selenium (SELENIUM, 2013), sendo bastante úteis em testes de regressão. Algumas ferramentas possuem apenas a função de gravação, sendo chamadas *Recorder*;

vi. Oráculo: é um mecanismo que gera os resultados esperados a partir das entradas, verificando mais tarde se os resultados obtidos na aplicação real conferem com os esperados. Protótipos ou versões anteriores do sistema realizam o papel do oráculo;

vii. Comparador de resultados: utilizados em testes de regressão, comparam os resultados atuais com os resultados de testes anteriores. Algumas outras ferramentas tem o comparador de resultados como um módulo integrado, como o oráculo e alguns gerenciadores de testes;

viii. Gerador de relatórios: ferramenta que gera relatórios para os resultados dos testes, sendo também parte integrante de outros tipos de ferramentas;

ix. Simulador e emuladores: reproduzem o comportamento de softwares ou hardwares reais, indisponíveis no momento do teste, por exemplo, quando o hardware está sendo desenvolvido simultaneamente ao software. Também são usados por motivos de segurança, financeiros e para treinamento. Úteis em testes de desempenho e estresse;

x. *Bug Tracker*: ao contrário das outras citadas anteriormente, esse tipo de ferramenta não possui alguma forma de automação, sendo uma forma de catalogar os erros da aplicação de forma detalhada (tipo, local e severidade da falha), podendo ser público, onde qualquer usuário pode reportar erros, normalmente para softwares *open source*, ou privado, internamente em uma companhia desenvolvendo software.

3.4.1. JUnit

JUnit é um framework *open source* para testes unitários para aplicações Java, se tornando um padrão devido a sua popularidade (TAHCHIEV *et al.*, 2010). JUnit é a versão para Java da família xUnit, assim como CppUnit para C++ e FUnit para Fortran, todos inspirados em uma primeira versão, chamada SUnit, para Smalltalk.

JUnit fornece toda uma estrutura para a execução de testes: cada teste unitário deve ser independente, é capaz de detectar e relatar erros de cada teste individualmente, e é possível definir facilmente quais testes serão executados. Uma

vantagem no JUnit é que ele fornece uma variedade de métodos *assert* para serem usados na codificação do teste, como: *assertArrayEquals*, *assertEquals*, *assertFalse*, *assertNotNull*, *assertNotSame*, *assertNull*, *assertSame*, *assertThat* e *assertTrue*, onde normalmente o primeiro parâmetro é o valor esperado e o segundo é o resultado do teste. Porém, alguns *asserts* possuem outras formas de chamada com outros parâmetros, por exemplo, passando a mensagem que deverá ser lançada ao quando o teste falhar. Esses métodos são importantes por serem os responsáveis por checar os resultados dos testes. Cada classe de teste tem um ou mais testes, e cada teste deve ter seu *assert*.

Outra vantagem é sua integração com uma larga diversidade de IDEs (Integrated Development Environment), como Eclipse, NetBeans, IntelliJ e JBuilder (TAHCHIEV *et al.*, 2010). Com essa integração, é possível rodar uma classe de testes com apenas um clique. Para cada caso de teste, a classe de testes é instanciada para evitar efeitos indesejados nos testes, significando também que um teste não pode depender de outro. Integrados às IDEs, é possível visualizar o progresso dos testes ao executar, assim como o resultado de cada teste, se obteve o resultado esperado ou não.

3.4.2.Selenium

Selenium é uma ferramenta *open source* que permite a criação de testes funcionais, interagindo sobre navegadores (que suportam JavaScript) como se fosse um usuário (BURNS, 2012). Para diferentes propósitos e usuários, possui diferentes ferramentas como IDE, Web Driver, Grid e Remote Control, onde a mais adequada ou um conjunto das mais adequadas pode ser utilizada. Para as ferramentas que utilizam código, como o Web Driver, pode ser utilizado em conjunto com o JUnit, sendo o último responsável pelo gerenciamento dos testes e os *asserts*, enquanto o Selenium executa as ações.

Selenium IDE

Selenium IDE é uma das ferramentas da suíte de softwares da Selenium, sendo um *plugin* para Firefox que permite gravar as ações realizadas no navegador e capaz de reproduzi-las (*Record/Playback*). Dessa forma, o testador não precisa programar todo o teste, pois o *plugin* é capaz de fazer a gravação e exportar o código, assim como é capaz de receber o código e executá-lo. O código pode ser exportado para uma variedade de combinações, como para Web Driver ou para o Remote Control, C#

ou Java, utilizando JUnit ou TestNG[19] (ambos para o caso da linguagem ser Java) (SELENIUM, 2013).

Selenium Web Driver

Diferente do Selenium IDE, a Web Driver tenta controlar o navegador por fora do mesmo, através da API de Acessibilidade, que é capaz de acessar e controlar aplicações, e para cada navegador, a API é diferente (e.g., em JavaScript para Firefox, em C++ para Internet Explorer). O código programado pelo testador é independente do navegador a ser utilizado, sendo de responsabilidade da API transformá-lo em algo que o navegador possa executar. Um exemplo de código em Java do Web Driver, que preenche um campo de id logradouro com o valor 'Av. Horácio Macedo' é:

```
WebDriver driver = new FirefoxDriver();
WebElement rua = driver.findElement(By.id("logradouro"));
rua.sendKeys("Av. Horácio Macedo");
```

3.5.Testes de software Web

Apesar das práticas de teste em softwares convencionais serem aplicáveis às aplicações Web, algumas questões técnicas específicas dessas aplicações tem que ser consideradas, além de adaptações em relação as técnicas nos softwares convencionais (NGUYEN, 2003). Esta seção busca ressaltar as diferenças existentes entre os tipos, técnicas e afins entre aplicações convencionais e aplicações Web.

Nesse tipo de aplicação, os usuários parecem esperar mais qualidade em comparação com softwares convencionais, almejando um carregamento rápido, respostas imediatas e fácil navegação. Isso se deve a facilidade de simplesmente trocar de site, em contrapartida a instalação e configuração de um software convencional, quando softwares de qualidade razoável são mais aceitáveis (MYERS, 2004).

Em contrapartida, o desenvolvimento acelerado, consequência da tendência da indústria em querer liberar o software para o mercado o mais rápido possível, faz com que os desenvolvedores tenham menos tempo para testar (NGUYEN, 2003).

[19] TestNG é um framework de testes inspirado no JUnit, com funcionalidades novas, abrangendo desde testes unitários (classes isolados) como testes de integração (TESTNG, 2013).

3.5.1. Arquitetura

Aplicações convencionais executadas em um computador pessoal possuem todos os processos – interface de usuário, entrada de dados, lógicas internas, manipulação de dados, armazenamento em arquivos ou banco de dados e saída de dados – normalmente rodando em um mesmo local. O modelo cliente-servidor, onde os softwares Web se enquadram, por sua vez, requer ao menos uma rede de comunicação, um computador cliente e um computador servidor, onde os processos são divididos, tornando a complexidade desse último modelo exponencialmente maior que o primeiro (NGUYEN, 2003).

Testar aplicações desse tipo também é um desafio. Para testar adequadamente o sistema, toda a arquitetura que será utilizada em produção deverá ser duplicada. Isso significa a replicação das mesmas camadas, normalmente as camadas de apresentação, de negócio e de dados, como mostra a Figura 4 (MYERS, 2004). A figura ilustra uma arquitetura utilizada em aplicações Web, especialmente aquelas que aplicam o modelo MVC. Porém, uma grande variedade de arquiteturas é possível em sistemas desse gênero, transformando a tarefa de testar um software desse tipo em um desafio, tarefa essa que tem sido algumas vezes negligenciada por conta das pressões que o mercado impõe (DI LUCCA & FASOLINO, 2006a).

Cada uma dessas camadas possui tipos de erros inerentes a elas. Para a camada de apresentação, os erros podem ser de conteúdo (fonte, cor, gramática), de arquitetura (links quebrados) ou devido ao ambiente do usuário (versões incompatíveis de navegador ou sistema operacional). A camada de negócio foca os erros na lógica do sistema, sendo muito similar aos testes realizados nas aplicações convencionais, além de tentar detectar erros de aquisição de dados e de processamento de transações. A camada de dados pode possuir erros de tempo de resposta (transações e manipulações de dados), de integridade de dados e de tolerância a falhas e recuperação (MYERS, 2004).

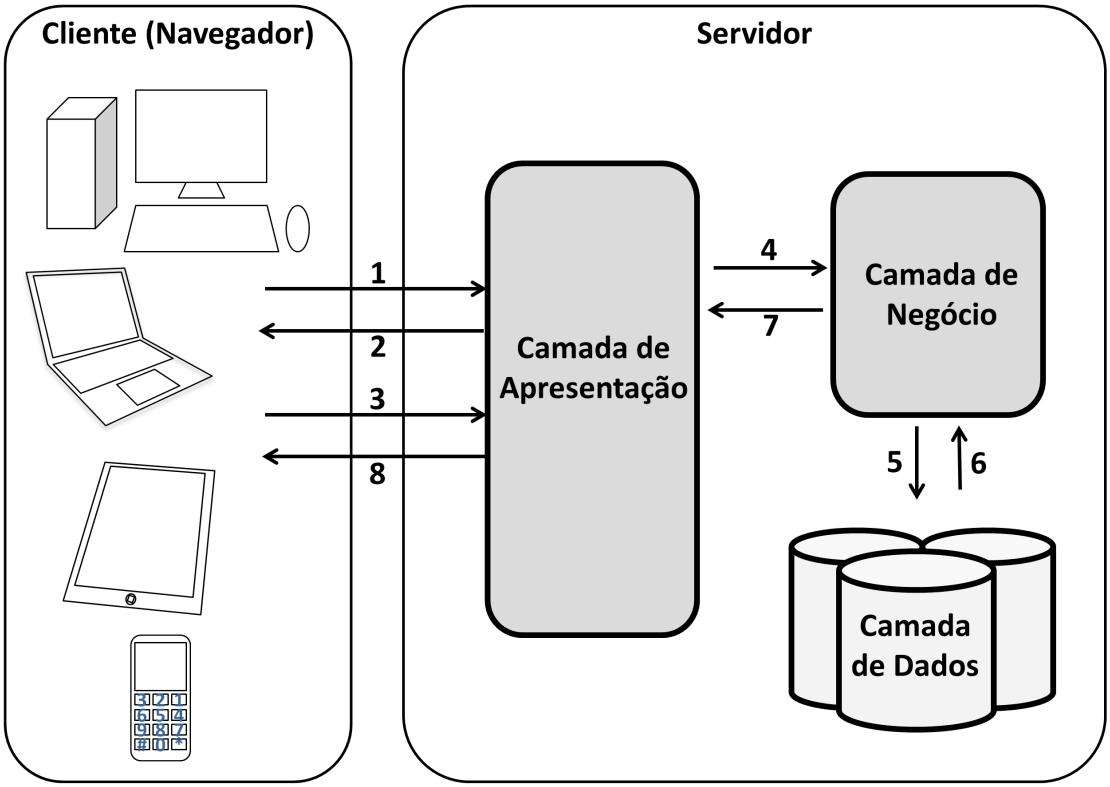

Figura 4 - Chamada à aplicação Web, adaptado de (MYERS, 2004, MILLS, 2008)

3.5.2.Níveis

Os níveis de testes mencionados na seção 3.1 também são aplicados a testes de software Web, porém, alguns conceitos devem ser adaptados para tal. Páginas normalmente são consideradas como as unidades, tanto páginas estáticas (arquivos html) ou dinâmicas (jsp, por exemplo). Testes de integração consideram um conjunto de páginas relacionadas, para investigar como elas trabalham juntas. Testes de sistema não variam muito em relação aos softwares tradicionais, podendo achar erros como links errados (DI LUCCA & FASOLINO, 2006a).

3.5.3.Objetivos

Alguns dos desafios presentes no desenvolvimento de software para web são: (i) segurança, uma vez que usuários de todo mundo poderão ter acesso a ele, inclusive pessoas mal–intencionadas, que queiram roubar dados do sistema, e (ii) a variedade de usuários, onde cada um possui habilidades específicas e utilizam diferentes navegadores, sistemas operacionais, aparelhos, velocidades de navegação e são de

lugares diferentes, com outro idioma e outro fuso horário (MYERS, 2004).

O objetivo de testar um software é fazer com que o mesmo vença esses desafios, através da melhoria da sua qualidade, que é um conceito complexo que abrange muitas dimensões (PRESSMAN & LOWE, 2008, EATON & MEMON, 2009). Essas dimensões são examinadas pelos testes individualmente ou em conjunto, e, portanto, não constituem fases e, sim, objetivos de teste (LAM, 2001). Eis alguns dos objetivos mencionados pelos autores, com uma breve descrição do que cada teste significa:

- Acessibilidade: busca verificar que o acesso ao conteúdo do software Web é possível mesmo com configurações de hardware ou software (bloqueio de *scripts*, por exemplo) reduzidas no lado do cliente, e também para pessoas com deficiências, como deficiência visual (DI LUCCA & FASOLINO, 2006b);
- Carga: avalia o desempenho de uma aplicação quando a mesma está submetida a um nível de carga predefinido, incluindo uma configuração mínima e um nível de atividade muito alto, simulando múltiplos acessos simultâneos de usuários (DI LUCCA & FASOLINO, 2006a);
- Compatibilidade: visa determinar se a aplicação se comporta como esperado em diversas configurações de hardware e software, tanto do lado do cliente como do lado do servidor (DI LUCCA & FASOLINO, 2006b, PRESSMAN & LOWE, 2008);
- Conteúdo: o conteúdo de um site também precisa ser avaliado sintaticamente (erros de pontuação e gramática), semanticamente (informações corretas e consistentes) e legalmente, onde os direitos autorais de imagens e textos devem ser respeitados (LAM, 2001, PRESSMAN & LOWE, 2008);
- Desempenho: visa medir o desempenho do sistema, em termos de tempo de resposta, tempo de execução, rendimento, taxa de tráfego, disponibilidade do serviço e utilização de recursos. Um exemplo de desempenho esperado é o tempo de resposta ser menor que 1 segundo em 90% dos casos (DI LUCCA & FASOLINO, 2006a, NAIK & TRIPATHY, 2008);
- Estresse: procura avaliar o software além da capacidade dele, verificando o comportamento do mesmo nessas circunstâncias, se o mesmo quebra ou consegue se recuperar. A diferença entre o teste de estresse e o teste de

desempenho ou de carga se dá pelo primeiro trabalhar em condições além do limite do software, enquanto os últimos simulam um uso regular da aplicação (DI LUCCA & FASOLINO, 2006b);

- Funcionalidade: trata de verificar se as funcionalidades do sistema executam aquilo para o qual foram projetadas de forma correta, exibindo o resultado esperado pelo usuário (PRESSMAN & LOWE, 2008). Ver 3.1.5;
- Internacionalização: com o potencial de qualquer usuário do mundo poder acessar a aplicação, caso mais de uma língua seja oferecida como opção, as traduções devem ser verificadas, se não há erros, se não há conteúdo ofensivo para o outro país e se valores e datas são convertidos e exibidos de maneira correta (LAM, 2001);
- Interoperabilidade: busca exercitar a comunicação com outros sistemas e/ou bancos de dados (PRESSMAN & LOWE, 2008);
- Navegabilidade: testes que exercitam a navegação do site, para que links errados, impróprios ou antigos sejam capturados e corrigidos (PRESSMAN & LOWE, 2008);
- Segurança: como qualquer usuário tem acesso à aplicação Web, testes de segurança buscam provar que o software é a prova de acessos e usos indevidos e de usuários não autorizados, que podem se aproveitar de falhas tanto no código como no ambiente em que o mesmo está hospedado (DI LUCCA & FASOLINO, 2006a);
- Usabilidade: testa a facilidade com que o usuário consegue realizar as tarefas, facilidade esta que é prejudicada por páginas pouco intuitivas, com uma navegação ruim e sem ajuda (LAM, 2001).

A troca de um software web por outro é o simples ato de acessar outra URL. Devido a essa facilidade e a concorrência cada vez maior entre as aplicações, uma falha em qualquer um dos objetivos pode acarretar em um prejuízo grande, com uma baixa no número de usuários, que pode levar ao fim do software.

3.5.4.Modelos de teste

Modelos de teste de funcionalidade de aplicações Web estão presentes na literatura, como formas de melhor testar um software desse gênero. Esta seção mostra algumas das formas presentes, em particular as voltadas para testes de funcionalidade

e navegabilidade, que são os focos deste trabalho.

3.5.4.1. *User-session based*

User-session based é uma técnica de teste caixa preta, desenvolvida por ELBAUM *et al*.(2003, 2005), que faz uso dos dados de navegação dos usuários e os dados inseridos pelos usuários em formulários em uma sessão, onde uma sessão é a sequência de URLs originadas de um mesmo IP, constituindo também um caso de teste. O foco dessa técnica são testes funcionais, e os dados são obtidos durante a execução dos testes beta, não utilizando dados dos usuários no ambiente de produção, usuários esses que poderiam questionar a privacidade das informações.

Cada requisição ao servidor de aplicação é armazenada, na forma da URL e dos parâmetros que, de acordo com o autor, podem ser obtidos configurando o servidor web Apache para armazenar em log, adicionando códigos javascript às páginas ou adicionando filtros de servlets Java para interceptar requisições e respostas dinamicamente. De posse desses dados, quatro abordagens de teste podem ser realizadas: a primeira repete a sessão do usuário completa; a segunda mistura duas requisições de usuários, selecionando um ponto de parada em uma sessão e procurando esse mesmo ponto (URL) para continuar o teste com dados de outra sessão; a terceira é a repetição em paralelo de várias sessões; a quarta é a mistura de uma sessão com requisições tipicamente problemáticas, como as do navegador (voltar, avançar e recarregar).

Um complicador desta técnica é o estado da aplicação quando o teste é executado, uma vez que os resultados das operações são influenciados por esse estado (banco de dados, por exemplo). A primeira abordagem não possui esse problema, desde que o estado inicial seja conhecido e replicado antes da reexecução. Por outro lado, as outras abordagens são afetadas pelo estado da aplicação. Uma alternativa a esse problema é guardar o estado da aplicação periodicamente. Outra é ignorar esse fator, fazendo com que o teste não seja replicado de forma fidedigna, mas ainda seja um teste útil, apesar de não possuir mais um parâmetro de comparação para designar o sucesso ou falha do teste.

3.5.4.2. Baseado em Modelo UML

ReWeb e *TestWeb* são duas ferramentas complementares para teste de aplicações web desenvolvidas por RICCA & TONELLA (2001). Elas se baseiam na

criação de um modelo UML da aplicação, onde podem existir páginas estáticas, páginas dinâmicas, frames, formulários e links. Através desse modelo criado, é possível identificar erros como links quebrados. Esse modelo é semiautomático, pois a construção dos modelos é feita automaticamente, porém os casos de teste necessitam das entradas e saídas fornecidas pelos testadores.

A aplicação *ReWeb* é dividida em três módulos: o primeiro é a varredura, que baixa todas as páginas de uma aplicação a partir de uma URL inicial, que resulta sempre em um html, independente de ser dinâmica ou estática, criando um modelo UML do software; o segundo é a análise utilizando o modelo UML para extrair informações da aplicação, como os caminhos, para que técnicas caixa-branca sejam aplicados; o terceiro é o visualizador do modelo, suportando consultas específicas para facilitar o processo.

O *TestWeb* possui dois módulos: o gerador e o executor de casos de teste. O primeiro gera casos de teste a partir das análises do modelo, dos critérios escolhidos pelo testador e das entradas fornecidas. Em posse dos casos de teste, o módulo de execução roda os testes, guardando as páginas de saída para a análise do testador. O testador abre as páginas resultantes e verifica o sucesso ou a falha do teste. Para testes de regressão, esse último tipo de intervenção não é mais necessário, sendo a primeira execução funcionando como um oráculo (ver seção 3.4).

3.6.Testes utilizando a multidão

Ao desenvolver uma aplicação, o grupo de desenvolvimento tem um interesse em demonstrar que o software desenvolvido funciona, ou seja, está livre de falhas. Ao pedir que esses mesmos desenvolvedores testem o programa, ou seja, tentem desconstrui-lo, acontece um conflito de interesses (PRESSMAN, 2010).

MYERS (2004) sugere que uma organização não deve testar seus próprios programas, onde um grupo diferente do que construiu o software fica encarregado de testar a aplicação, resolvendo o conflito existente. A esse grupo é dado no nome de ITG (*Independent Test Group* – Grupo de Teste Independente) (PRESSMAN, 2010).

3.6.1.A multidão

Um dos princípios de MYERS (2004) menciona a criatividade e o desafio intelectual que existe em testar softwares. A criatividade e a inteligência estão

presentes em vários projetos de *crowdsourcing* (SCHENK & GUITTARD, 2011, THREADLESS, 2012, INNOCENTIVE, 2012, GALAXYZOO, 2012, ISTOCKPHOTO, 2012), tornando a multidão um potente criador de testes. Além disso, a multidão é independente da equipe de desenvolvimento do software, funcionando como um ITG.

3.6.2.Aplicações

Algumas empresas notaram o potencial do *crowdsourcing* para a realização de testes em software (CROWDTEST, 2012, MOB4HIRE, 2013, UTEST, 2012), utilizando testadores de diversas partes para testar softwares web, móveis ou desktop. Na literatura, também foram encontrados trabalhos relacionados ao teste utilizando a multidão (YU *et al.*, 2009, NEBELING *et al.*, 2012). A pesquisa para encontrar esses estudos está relatada no Apêndice 1. As seções seguintes ilustram algumas delas, mostrando como funcionam e quem são os testadores envolvidos.

3.6.2.1.Call-For-Testing

Call-For-Testing - CFT (YU *et al.*, 2009) é um modelo de teste para aplicações web utilizando a multidão, aplicável a testes de aceitação, uma vez que encontrar usuários finais apropriados é uma tarefa árdua, dada a possibilidade de qualquer pessoa no globo poder ser usuário da aplicação.

Os desenvolvedores da aplicação, baseados nos requisitos, tem a função de criar anotações sobre os elementos da página: quais serão validados e respectivos tipo de validação e quais elementos se relacionam com outros e de que forma. Através da exibição dessas anotações, os testadores na multidão geram os casos de testes correspondentes, através da gravação de seus eventos, como cliques em links ou inserção de data.

Além disso, há um componente responsável por varrer toda a aplicação, recuperar as páginas e gerar requisitos de teste para os links, entradas, eventos, relações entre elementos e caminhos de links. Após a varredura, os requisitos extraídos são comparados com os dados dos casos de teste gerados. Com isso, é possível obter a cobertura em termos de elementos, relacionamento entre elementos e caminhos percorridos, concluindo se os requisitos de teste foram adequadamente testados.

3.6.2.2.CrowdStudy

CrowdStudy (NEBELING *et al.*, 2012) é uma ferramentas para avaliar a usabilidade de aplicações web utilizando a multidão, permitindo assim o teste sob diversas condições diferentes, como aparelho, sistema operacional e navegador. O recrutamento dos testadores é feito de duas formas, a primeira é implícita, ou seja, os próprios usuários da aplicação desempenham esse papel, e a segunda é através da integração com o MTurk.

A configuração é realizada de forma tão simples quanto a inclusão de uma biblioteca JavaScript, e injeta componentes adicionais nas aplicações para rastrear as atividades do lado do cliente e registrar dados do lado do servidor. Do lado do cliente, é capaz de capturar informações como quais tamanhos de fonte e orientação o usuário prefere ao ler em um smartphone ou tablet, da mesma forma que captura os dados de toque na tela, conseguindo saber como interagem com o site, onde erram os cliques e onde aproximam a tela, dados importantes para avaliação da usabilidade.

Tarefas são distribuídas aos testadores do MTurk, através de descrições textuais, podendo ter especificações como qualificação do usuário, especificações dos aparelhos utilizados pelo mesmo ou tarefas que devem ser feitas previamente. As tarefas podem envolver um ou mais componentes dos softwares, além da navegação entre páginas.

3.6.2.3.uTest

A ferramenta uTest, utiliza a multidão para testes de aplicações web, desktop e mobile, além de especialistas em testes da própria companhia. É o maior mercado de testadores do software do mundo, contando com mais de 70.000 testadores distribuídos em mais de 190 países (UTEST, 2012).

A aplicação fornece cinco tipos de testes:

- Funcionalidade: a aplicação que será testada possui o controle para especificar os requisitos dos testes, para selecionar os testadores e para revisar os erros e os casos de teste. Através de testes exploratórios, consegue armazenar os erros devido à integração com ferramentas *bug-tracking*, como o *Bugzilla*. A automação de testes é atingida através da multidão de testadores programando *scripts* em diversas ferramentas, como o Selenium;
- Carga: testes de carga podem ser realizados por ferramentas de simulação,

por vários testadores simultaneamente ou por uma combinação dos dois, onde os testadores realizam testes funcionais com o sistema carregado através dos simuladores;

- Segurança: testadores irão submeter os tipos de ataques mais comuns, como *Cross-Site Scripting* (XSS), *SQL Injection*, *Denial of Service* (DoS) e outros;
- Internacionalização: como a linguagem, cultura, moeda e padrões são diferentes em cada país, há uma necessidade de se testar a tradução, onde os testadores inspecionam o conteúdo das páginas;
- Usabilidade: a usabilidade de um site pode arruiná-lo, por isso, testes desse tipo são importantes, onde testadores navegam pelo site buscando descobrir falhas nesse sentido.

O cadastro de um testador é extenso, e engloba os idiomas, os tipos de aplicação (web, móvel ou desktop), os tipos de teste que ele está apto a realizar, junto com a experiência em cada área (automação em diversas linguagens, execução manual de caso de teste, teste de carga utilizando Selenium ou Rational), sistemas operacionais, navegadores e dispositivos móveis utilizados.

3.6.2.4.Mob4Hire

Semelhante ao uTest (UTEST, 2012), Mob4Hire (MOB4HIRE, 2013) possui uma grande quantidade de testadores (60.000), distribuídos em mais de 150 países, voltada para aplicações móveis. O objetivo da ferramenta é auxiliar os aplicativos a atingir uma qualificação de quatro estrelas ou mais nas lojas online (Google Play ou App Store). Nessa plataforma, pessoas com diferentes habilidades são capazes de testar, desde as que apenas utilizam os aparelhos no cotidiano quanto os profissionais de teste. Algumas grandes empresas de aparelhos móveis possuem parceria com a ferramenta, como a Motorola[20], Samsung[21] e a BlackBerry[22].

3.6.2.5.Crowdtest

Crowdtest é uma ferramenta brasileira de testes, que pode ser utilizada para testar aplicações web, móvel ou desktop, sendo utilizada por empresas como a Natura e Buscapé (CROWDTEST, 2012).

Cada aplicação a ser testada possui um regulamento, onde poderão ser

[20] http://www.motorola.com.br/consumers/home

[21] http://www.samsung.com/br/

[22] http://br.blackberry.com/

especificados os sistemas operacionais e navegadores que deverão ser utilizados, no caso de aplicações web. O registro de uma falha segue um guia, onde são exigidos os preenchimentos de um título, uma descrição do erro, os passos necessários para reproduzir a falha, os resultados esperados, a funcionalidade da aplicação onde o erro foi encontrado, o navegador, o sistema operacional, arquivos anexos (*print screens* ou videos) e o tipo de falha, que pode ser: (i) de interface, que inclui problemas de apresentação ou cores, (ii) funcional, falha que pode ser contornada, como um botão de limpar que não limpa, (iii) impeditiva, uma falha que impede o uso do sistema, ocasionando erro ou travamento do sistema, (iv) de segurança da aplicação, (v) de texto, incluindo a gramática e a ortografia ou uma (vi) sugestão de melhoria. Os valores para cada tipo de falha são diferentes, variando de projeto para projeto.

No escopo de testes de aplicações web, o acesso ao site a ser testado é feito externamente, e um usuário e senha é requisitado pelo navegador, para que possa ser feita a visualização da aplicação. O testador tem total liberdade de navegar pelo site, buscando qualquer tipo de erro, relatando assim que encontrar. O testador recebe uma recompensa por bug inédito encontrado, que varia de acordo com o tipo. Os testadores são selecionados de acordo com o perfil do público alvo da aplicação, fazendo com que o usuário se preocupe não só com o funcionamento, mas também se a aplicação atende suas necessidades (CROWDTEST, 2012). Para isso, no cadastro de um testador, informações como idiomas, áreas de conhecimento (construção civil, saúde, indústria, comércio), certificações de teste, experiência em testes (exploratórios, automação), sistemas operacionais e dispositivos móveis são campos a serem preenchidos.

3.6.2.6.Comparativo

A seleção das cinco ferramentas mencionadas nas seções anteriores foi devido a uma revisão sistemática de literatura (KITCHENHAM & CHARTERS, 2007) sobre a utilização de *crowdsourcing* em testes de software. Por se tratar de um processo detalhado, os pormenores estão contidos no Apêndice 1, em conjunto com uma explicação do que é uma revisão sistemática de literatura, e uma justificativa para sua escolha.

A partir dos dados extraídos de cada uma das ferramentas de teste, um comparativo foi elaborado, como ilustra a Tabela 2. A coluna referente aos tipos de

ferramenta possui o valor Pesquisa em algumas aplicações, significando que não é uma ferramenta, e sim um questionário aplicado aos testadores. O valor "-" significa que o dado não está disponível.

É possível notar que há uma variedade razoável nos tipos de testes aplicáveis pelas ferramentas, porém apenas uma fornece alguma forma de automação de testes de regressão (uTest através dos scripts). Quanto a capacitação dos testadores, para testes de funcionalidade e usabilidade praticamente não é exigido um conhecimento de testes, exceto pela aplicação uTest quando utiliza a multidão para programar scripts.

Tabela 2 - Características das ferramentas de teste de software com *crowdsourcing*

Ferramenta	Tipos de aplicação testáveis	Tipos de teste disponível	Tipo da ferramenta	Motivação da multidão	Quem pode testar
Call-For-Testing	Web	Aceitação	*Recorder* e *Bug Tracker*	-	-
CrowdStudy	Web	Usabilidade	*Recorder*	-Implícito (usuários da aplicação) -Dinheiro (usuários do MTurk)	Usuários da aplicação e usuários do MTurk
CrowdTest	-Tradicional -Móvel -Web	-Funcionalidade -Segurança -Internacionalização -Interface	*Bug Tracker*	Dinheiro, pagando por bug	Qualquer um
Mob4Hire	Móvel	-Aceitação -Funcionalidade -Usabilidade	Pesquisa	Dinheiro, pagando por cada teste do usuário	Qualquer um
uTest	-Tradicional -Móvel -Web	-Funcionalidade -Carga -Internacionalização -Segurança -Usabilidade	-*Bug Tracker* (funcionalidade, carga, segurança e internacionalização) -Simuladores (carga e segurança) -*Scripts* (funcionalidade, utilizando o Selenium) -Pesquisa (usabilidade)	Dinheiro, pagando por bug	Qualquer um (funcionalidade, internacionalização e usabilidade) Profissionais de teste (carga, segurança, usabilidade e funcionalidade)

3.7.Considerações Gerais

Esse trabalho foca em testes de sistema, ao estabelecer um processo para testes utilizando multidão. Porém, ao utilizar pessoas externas, elas acabam incorporando o papel de usuário, tornando o processo adequado a testes de aceitação. Como o código das aplicações a serem testadas são proprietários, os testes utilizaram a técnica caixa-preta, sem que os testadores conheçam o código. Isso possibilita também a utilização de pessoas que não entendam de programação.

O objetivo do processo é fazer com que a multidão assuma o papel de quatro tipos diferentes de ferramentas de teste: (i) Gerador de dados de teste, ao entrar com dados oriundos de suas cabeças; (ii) *Record/Playback*, ao interagir com o sistema de forma a criar um caso de teste que seja possível reproduzir; (iii) Oráculo, gerando resultados esperados e conferindo com outros resultados; e (iv) *Bug Tracker*, ao relatarem os erros encontrados.

Dentre os objetivos, podem ser citados três quanto a parte relacionada a programação e três quanto a apresentação. Os três primeiros são a funcionalidade, objetivo principal da proposta, navegabilidade e compatibilidade pelo lado do cliente, uma vez que cada pessoa da multidão utiliza configurações diferentes de hardware e software. Para a apresentação, objetivos secundários da proposta, são a internacionalização, o conteúdo e a usabilidade, possível apenas por utilizar pessoas nos testes.

4.Proposta

Como mencionado algumas vezes, aplicações Web necessitam de formas de teste que atendam a rapidez das demandas de mudança nos softwares. A multidão é uma maneira eficiente de conseguir mão-de-obra diferenciada, criativa e qualificada, como mostram as aplicações da seção 2.2.3 e o sucesso que fazem. A criatividade e a inteligência, qualidades necessárias para a realização de testes de software de acordo com o décimo princípio de MYERS (2004), vem sendo largamente explorada em aplicações que utilizam a multidão como fonte dessas duas qualidades. Por essa razão, existem softwares e pesquisas sobre a utilização da multidão em testes de aplicações web, mobile e desktop, como mostrado no capítulo 3, em especial na seção 3.6.

4.1.Motivação

Testar aplicações Web utilizando navegadores reais aumenta a precisão e o realismo dos testes, quando comparado com testes que simplesmente iniciam requisições HTTP e não analisam nem renderizam o HTML e JavaScript, por exemplo (SACKS, 2012). Reproduzir um erro gerado em um ambiente (por exemplo, sistema operacional e navegador) em outro pode ser difícil ou até mesmo impossível (PRESSMAN & LOWE, 2008).

Além disso, outro ponto importante é que a automação de testes não pode substituir testes manuais, uma vez que a máquina não consegue imitar algumas características importantes dos seres humanos, como a criatividade, a variabilidade e a observação, não detectando alguns problemas facilmente detectados por humanos (NAIK & TRIPATHY, 2008).

O objetivo principal deste trabalho é criar um processo de teste de aplicações Web que utiliza a multidão, considerando suas ações e seus julgamentos dos testes. O processo tem como foco os testes de funcionalidade utilizando a multidão, definindo passos necessários para a utilização da multidão no processo. Antes de abordar o processo em si, algumas definições precisam ser feitas, para que fique mais claro o entendimento.

4.2.Definições

Dois tipos de papéis estão envolvidos: o responsável pela aplicação e o do testador, parte integrante da multidão. O responsável pela aplicação é aquele contratado da empresa que está desenvolvendo o software, ou mesmo os próprios desenvolvedores da aplicação. Por sua vez, o testador da multidão é qualquer pessoa ligada à rede que possua vontade e disponibilidade em auxiliar nos testes da aplicação

A definição de caso de teste no processo proposto é uma adaptação da definição utilizada por vários autores: um caso de teste é uma sequência de URLs e dos dados preenchidos pelos usuários nos formulário HTML, passados na forma de parâmetro em cada uma das requisições (ELBAUM *et al.*, 2003, 2005, HAJIABADI & KAHANI, 2011, ALSHAHWAN & HARMAN, 2012). No processo a definição vai além, um caso de teste é uma sequência de ações/eventos executados por um testador obedecendo a um caso de teste base. Caso de teste base e ações/eventos são outros dois conceitos a serem explicados.

Caso de teste base é um conceito de nível superior ao caso de teste. Ele é a união de um requisito com um roteiro, ao qual o caso de teste atende. Esse roteiro é algo que pode ser bastante abrangente ou bastante específico, dependendo de como o responsável pela aplicação o define. A partir dessas informações, o testador da multidão pode criar casos de testes adequados.

Uma ação ou evento é todo tipo de interação que o testador realiza com a aplicação a ser testada. Um clique do mouse, uma tecla pressionada, um foco em um determinado campo e uma submissão de um formulário são exemplos de ações que o usuário pode executar. Ações do navegador, como voltar, avançar e recarregar não são considerados eventos no contexto do processo.

4.3.Etapas

Dada as definições anteriores, torna-se possível a explicação do processo em si. Ele pode ser dividido em duas fases: a fase de definição e a fase de realização. A primeira é feita pelos responsáveis, enquanto a segunda pelos testadores da multidão. A Figura 5 ilustra essas fases e etapas. Os detalhes de cada etapa são explicados na sequência.

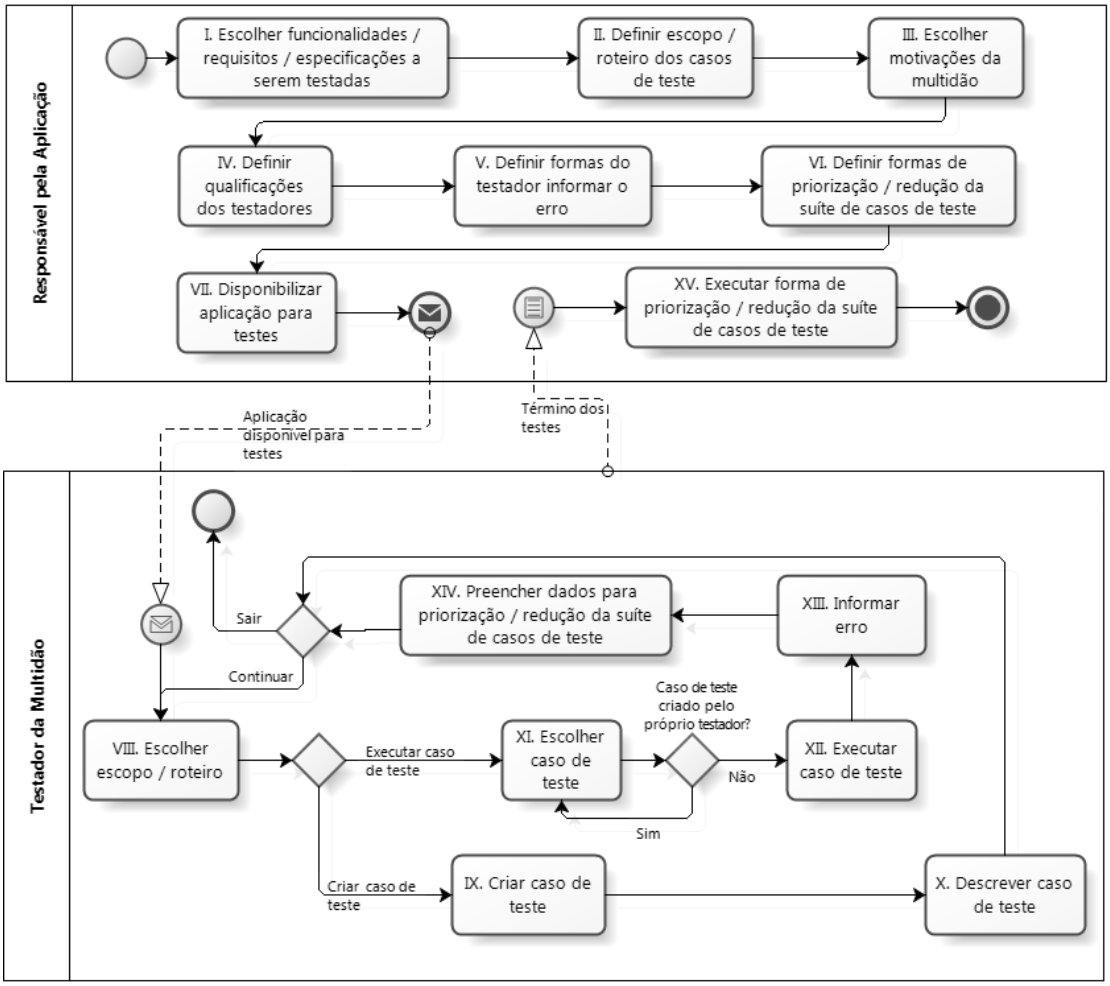

Figura 5 - Etapas e passos do processo

As etapas definidas no processo são bastante flexíveis, deixando a maioria das definições a cargo de quem for implantá-lo. Um ponto rígido do processo é a visibilidade da contribuição dos outros, uma das características das aplicações de *crowdsourcing*, melhor explicada na seção 2.2.2.5. Ela é da forma de avaliação, onde um testador consegue visualizar, executar e avaliar, porém não modificar a contribuição de outro. Isso funciona também como um controle, onde um testador não poderá avaliar as próprias contribuições.

As etapas do responsável pela são executadas apenas uma vez, em uma ordem linear, conforme o processo (exceto a etapa XV). Entretanto, as etapas da multidão, por serem atribuídas a muitas pessoas utilizando em paralelo, não são executadas apenas uma vez, havendo concorrência entre as execuções de diferentes testadores.

I - Escolher funcionalidades / requisitos / especificações a serem testadas

Algumas vezes a aplicação ainda não está completamente pronta, e uma boa prática é que os testes comecem sempre mais cedo, sendo mais fácil consertá-los nas fases iniciais. Além disso, por alguma circunstância, pode não ser exigido que a aplicação inteira seja testada, apenas a parte que atingirá a maioria das pessoas.

Os motivos citados justificam a primeira etapa do processo: definir quais as funcionalidades, requisitos e/ou especificações serão testadas. Somente a partir da escolha, que pode ser a aplicação inteira, poderá ser melhor definido como os casos de teste são desejados, através de um roteiro, que pode ser visto na etapa seguinte.

II – Definir escopo / roteiro dos casos de teste

Dada a escolha de uma funcionalidade, o testador da multidão não conhece a aplicação que irá testar. Sem um roteiro, a única base que ele possui é o entendimento próprio da aplicação que estará testando. Por isso, um roteiro deve ser definido para cada funcionalidade especificada na etapa anterior. Esse roteiro pode ter como base um caso de uso, ou formas mais específicas, como um diagrama de sequência ou um diagrama de atividades. Pode também ser um roteiro menos específico, tratando as funcionalidades de forma mais abrangente, deixando o testador com maior liberdade para cadastrar um caso de teste, criando um caso de teste exploratório.

Essa etapa, junto à etapa anterior, compõem o que foi denominado de caso de teste base, ou seja, a fonte dos dados que serão disponibilizados para que os testadores possam criar novos casos de teste.

III – Escolher motivações da multidão

A definição da motivação especifica quais serão as formas de recompensa dada aos testadores da multidão ao colaborar, que podem ser dadas apenas aos melhores de um ranking ou mesmo pela criação de um caso de teste. Qualquer forma citada na seção 2.2.2.6 pode ser escolhida.

IV – Definir qualificações dos testadores

A definição da qualificação consiste em especificar quais conhecimentos uma pessoa deverá possuir para se tornar um testador da aplicação. Os tipos de

qualificação exigidos em projetos de *crowdsourcing* podem ser vistos na seção 2.2.2.3, e a que melhor se enquadre as necessidades deve ser escolhida.

V – Definir formas do testador informar o erro

Um bom caso de teste é aquele que tem grande probabilidade de encontrar um erro. Porém, os testes podem resultar em erro ou não, e o desejado ao testar, é que eles sejam encontrados. Por isso, ao menos uma forma que permita aos testadores informar o erro deve estar disponível, transformando a multidão em um Oráculo da aplicação. O número de casos de testes gerados pela multidão pode ser muito grande, tornando impraticável a execução de um a um pelo responsável da aplicação.

O ponto crucial é como o testador avalia se um caso de teste que ele executou resultou em erro ou não. A presente pesquisa mostra que o fator de maior relevância para esse julgamento é a descrição fornecida pelo testador que criou o caso de teste, mais precisamente a etapa X do processo. O processo propõe que quem deve informar o erro são os testadores ao executar o caso de teste (etapa XII) e não os criadores na descrição do caso de teste (etapa X).

Existem duas formas imediatas para que o testador revele a presença de um erro. A primeira, ao final da execução de um caso de teste, ele escolhe se o resultado obtido é o resultado esperado por ele, de acordo com o roteiro, e a segunda é disponibilizar algum lugar para que ele descreva o erro, caso tenha sido encontrado, funcionando como um *Bug Tracker*.

VI – Definir formas de priorização / redução da suíte de casos de teste

Como mencionado na etapa anterior, o número de casos de teste pode ser muito elevado e, além disso, a qualidade dos testes pode ser baixa, devido a diversidade da multidão. Por isso, uma forma de redução da suíte de casos de teste deve ser pensada. A redução deve ser eficiente, de forma que o número de casos de testes reduza, porém a eficiência deles não seja comprometida.

Apesar disso, mesmo com uma redução, o número de casos de teste pode ainda ser grande, demorando muito tempo para serem executados como testes de regressão. Nesse ponto entra a priorização dos casos de teste. Tanto a redução quanto a priorização são processos importantes, e definir a forma como isso será feito é uma

tarefa complicada.

No presente trabalho, ao conduzir experimentos que atendem ao processo, uma forma de avaliação foi designada aos testadores, onde eles atribuíam de 1 a 5 estrelas para cada caso de teste. Porém, os resultados obtidos não foram claros para saber se essa é uma boa forma ou não.

VII – Disponibilizar aplicação para testes

Após todas as definições das etapas anteriores, a aplicação deve ser disponibilizada para que os testadores possam realizar seu papel. Ela deve ser colocada em um ambiente semelhante ao de produção, replicando toda a arquitetura e estrutura, para que testes fidedignos possam ser realizados. A forma como a multidão de testadores será rastreada pode variar, desde a utilização dos logs dos servidores, como inserindo scripts na página ou filtros na aplicação (ELBAUM *et al.*, 2003, 2005).

VIII – Escolher escopo / roteiro

Após todas as etapas elaboradas pelos responsáveis pela aplicação, entram em ação os testadores da multidão. A primeira etapa é escolher a funcionalidade e o roteiro a seguir, definidos nas etapas I e II. Um testador escolhe um caso de teste base por vez e, a partir dele, poderá cadastrar um caso de teste, como será explicado na etapa seguinte, ou executar um caso de teste, explicado posteriormente.

O testador da multidão deve se sentir confortável com o escopo / roteiro, entendendo-o bem, extraindo o objetivo que deve alcançar ao criar os casos de teste, assim como para compreender o objetivo do testador que criou um caso de teste. A partir dessa etapa, o testador escolhe o que fazer, criar (etapa IX) ou escolher (etapa XI) um caso de teste.

IX – Criar caso de teste

Feita a escolha do caso de teste base a seguir, o testador está apto a cadastrar um novo caso de teste. Alguns roteiros podem ser rígidos o suficiente para servir de *scripts* de caso de teste, quando o testador poderá variar apenas os dados inseridos, funcionando como uma fonte de dados para os testes. De acordo com PRESSMAN & LOWE (2008), testes funcionais são, geralmente, direcionados a entradas de formulários HTML, para aplicações Web.

Porém, alguns outros roteiros podem ser flexíveis o suficiente para que o testador se sinta livre para navegar a vontade na aplicação, criando um caso de teste mais próximo do exploratório. Entretanto, um cuidado deve ser tomado quanto a criação de casos de testes muito longos. Percorrer toda a aplicação em um único caso de teste pode parecer uma boa ideia, mas os experimentos mostraram que os outros testadores, ao executar casos de testes muito grandes, reclamavam, e acabavam por não colaborar.

Nessa etapa, durante a criação do caso de teste é que funciona o rastreamento das ações da multidão, onde cada ação é gravada, através de logs, Javascript ou filtros. Essa forma é importante para a futura reprodução dos casos de teste por outros testadores.

X – Descrever caso de teste

Ao término da criação de um caso de teste, o testador deve ser convidado a descrever o caso de teste que ele criou. Dessa forma, o testador que irá reproduzir o caso de teste criado por ele tem as informações do roteiro que ele seguiu e os objetivos que o levaram a realizar aquelas ações. A descrição é o fator mais relevante para que outro testador consiga dizer se houve ou não erro na execução do caso de teste, de acordo com a forma definida na etapa V. Essa informação foi retirada dos experimentos realizados nesse trabalho.

Outro dado importante extraído dos experimentos é que a descrição não deve conter o erro, caso haja, encontrado pelo autor do caso de teste. Como a descrição é importante, caso o erro esteja contido nela, os outros testadores, ao se depararem com o erro, podem achar que ele é esperado (de acordo com a descrição ele é), resultando em um julgamento equivocado.

XI – Escolher caso de teste

Após ler, escolher e compreender o caso de teste base, ele deverá escolher um dos casos de teste criados, para execução. Para escolher, ele deve ler cuidadosamente a descrição fornecida pelo criador do caso de teste, por esse ser o fator de maior relevância para informar a presença ou não do erro. Caso o testador escolha testar um caso de teste criado por ele, não será permitida sua execução, sendo necessária a escolha de outro caso de teste.

XII – Executar caso de teste

Essa etapa funciona como uma verificação da etapa IX, onde um testador executa um caso de teste. O testador que cria o caso de teste (criador) não poderá executar um caso de teste próprio, sendo essa tarefa designada a outro testador (executor). Ao selecionar um caso de teste para testar, o executor deve ter a exata noção das ações que o outro executou, como uma repetição de um *script*. Quanto mais fiel a execução, melhor o entendimento que o testador que está executando terá.

Não há um limite imposto ao número de vezes que um testador pode executar um caso de teste, e nem um limite quanto ao número de casos de teste que o testador pode executar. Com tempo e paciência suficiente, o testador poderá executar todos os casos de teste cadastrados, exceto os criados por ele, se assim for desejado. Porém, as ferramentas seguindo esse processo poderão impor os limites desejados, caso seja interessante.

XIII – Informar erro

Após a execução, o testador deverá informar se ele percebeu a presença ou não de um erro, levando em conta o roteiro do caso de teste base, a descrição do caso de teste e sua percepção de como a aplicação funciona. Essa etapa é fundamental, pois é a partir dela que o responsável pela aplicação saberá quais casos de teste encontraram erros, sem precisar reproduzir um a um.

XIV – Preencher dados para priorização / redução da suíte de casos de teste

Além de informar o erro, é possível delegar à multidão a tarefa de reduzir e priorizar os casos de teste. Uma das formas utilizadas nesse trabalho foi disponibilizar para a multidão um esquema de avaliação 5 estrelas, para que o testador diga quão bom ele julga o caso de teste. Dessa forma, os mais bem avaliados podem ser considerados os melhores casos de teste, e serem executados pelo responsável pela aplicação, para investigar algo a mais que ele deseje.

XV – Executar forma de priorização / redução da suíte de casos de teste

A priorização pode acontecer durante as etapas em que a multidão está colaborando, como na etapa anterior, ou após a multidão interagir com a aplicação. Nesse caso, formas clássicas (SAMPATH *et al.*, 2008) que não envolvam a multidão deverão ser utilizadas para priorização, sendo dirigidas pelos responsáveis da aplicação. Essa etapa é executada após toda a fase de testes por parte da multidão acabar, de posse de todos os casos de testes criados.

4.4.Prevenção de problemas

A diversidade da multidão pode causar dois problemas: quantidade exorbitante e baixa qualidade dos casos de teste, como mencionado anteriormente. O processo de redução e priorização dos casos de testes possui como objetivo maior a prevenção de ambos os problemas.

Como mencionado por QUINN & BEDERSON (2009, 2011), quando há uma competição ou dinheiro envolvido como motivação, alguns usuários podem tentar trapacear. No processo, caso a recompensa seja pelo número de casos de teste criados, testadores podem trapacear ao criar muitos de baixa qualidade. Por isso, esse tipo de recompensa é desencorajado, podendo ser substituída por recompensas proporcionais a qualificação dada por outros testadores, excluindo os de menor importância.

Outro problema existente ao se tratar de testes de regressão é que podem exigir muito tempo para sua reexecução, caso o número de casos de teste seja grande. Como a multidão se torna responsável pela criação dos casos de teste, a quantidade gerada pode ser um número extremamente grande. A avaliação da importância dos casos de teste por meio de notas ou algum outro critério subjetivo busca eleger um grupo de casos de testes importante para uma reexecução.

4.5.Cálculo da importância dos testes

A etapa 14 menciona uma forma de redução e priorização dos casos de teste utilizando a multidão, atribuindo 1 a 5 estrelas. Uma forma de delegar essa tarefa a multidão é cada testador julgar a importância de um caso de teste, não sendo possível atribuir um valor de relevância a um caso de teste elaborado por ele mesmo. Como

resultado final para o caso de teste, uma média das avaliações é feita, e consequentemente uma ordem é gerada. Porém, antes de extrair a média, algumas considerações e ajustes devem ser feitos.

Atribuir um número de estrelas é um modo subjetivo de avaliar, apesar de um número de estrelas ser um valor inteiro. A subjetividade reside em cada usuário ter sua própria percepção do que significa 1 ou 5 estrelas. Considere dois usuários, o primeiro costuma atribuir 2 ou 3 estrelas (média de 2,5) e o segundo costuma atribuir 4 ou 5 estrelas (média de 4,5). Para um determinado teste x, ambos atribuíram 5. Para um sistema simples, as duas notas 5 tem o mesmo peso. Porém, a diferença da importância que o primeiro usuário deu para o teste em comparação ao segundo usuário é grande. Algo no teste x tem uma importância diferente para o primeiro usuário, fazendo com que o avaliasse com importância 5.

Devido a esses fatores, a importância de cada teste pode ser calculada de outra maneira, como o valor relativo. Esse valor é calculado a partir da diferença da importância com a média das importâncias do respectivo usuário. Considere o conjunto U de n usuários (u_1, u_2, ..., u_n), e o conjunto T de m testes (t_1, t_2, ..., t_m), onde a_{ij} é a avaliação do usuário u_i para o teste t_j, com 0 representando que não foi avaliado. Cada usuário tem sua média definida por m_i, onde são contados apenas os testes avaliados pelo usuário. Após o cálculo da média para cada usuário, uma nota real da avaliação do usuário i para o teste j será representada por n_{ij}. Este valor é calculado da seguinte forma: caso $a_{ij}>m_i$, então $n_{ij}=(a_{ij}-m_i)^2$, caso contrário $n_{ij}=-(a_{ij}-m_i)^2$. O exemplo seguinte deixa mais claro os cálculos e a diferença deste modo, aqui denominado relativo, para a simples utilização da média, denominado absoluto.

Tabela 3 - Exemplo do cálculo de relevância de um teste

	t_1	**t_2**	**t_3**	**t_4**	
Forma absoluta					
u_1	$a_{11}=2$	$a_{12}=0$	$a_{13}=3$	$a_{14}=4$	$m_1=3$
u_2	$a_{21}=5$	$a_{22}=1$	$a_{23}=1$	$a_{24}=2$	$m_2=2,25$
u_3	$a_{31}=0$	$a_{32}=5$	$a_{33}=4$	$a_{34}=5$	$m_3=4,67$
Média	**3,5**	**3**	**2,67**	**3,67**	
Forma relativa					
u_1	$n_{11}=-1$	$n_{12}=0$	$n_{13}=0$	$n_{14}=1$	
u_2	$n_{21}=7,56$	$n_{22}=-1,56$	$n_{23}=-1,56$	$n_{24}=-0,062$	
u_3	$n_{31}=0$	$n_{32}=0,11$	$n_{33}=-0,44$	$n_{34}=0,11$	
Média	**3,28**	**-0,73**	**-0,67**	**0,35**	

Como é possível observar, caso a média absoluta das notas fosse considerada,

a ordem de prioridade dos testes a serem executados seria t4, t1, t2 e t3. Da forma relativa, a ordem foi modificada, resultando em t1, t4, t3 e t2. Isso se deve ao peso da nota 5 em a_{21} em um usuário de média m_2=2.25, enquanto a nota 5 em a_{34} tem um peso menor por ser de um usuário de média m_3=4,67. Vale ressaltar que o valor n_{13}=0 deve ser levado em consideração na divisão, pois é resultado da diferença entre a nota e a média, que são iguais. Caso apenas um teste tivesse que ser escolhido, esse resultado mudaria completamente de uma abordagem para a outra. Outras formas de calcular a importância de um caso de teste também podem ser utilizadas no processo.

4.6.Ferramenta

Para demonstrar que o processo é funcional e aplicável, uma plataforma de testes pela multidão foi criada, aplicando o processo proposto ao definir algumas etapas. A ferramenta possui a forma de uma plataforma, onde várias aplicações Web podem ser cadastradas com o intuito de serem testadas pela multidão. A ferramenta recebeu o nome CAT, acrônimo para *Crowdsourcing Application Testing*.

Esta ferramenta é inspirada em outros modelos e ferramentas, como o *User-session based* de ELBAUM *et al.* (2003, 2005), na aplicação uTest (UTEST, 2012), e nos tipos de ferramentas de automação de captura e repetição, tendo como base as ações dos usuários a partir de uma sequência de URLs definidas pelos responsáveis pela aplicação.

4.6.1.Tecnologias

A plataforma foi desenvolvida utilizando a linguagem Java em sua versão 6, utilizando o conceito de MVC. Para a camada de controle e visão, foi utilizado o framework Java Server Faces 2.0 (JSF). Para a camada de modelo, foi utilizado o framework JPA, através do Hibernate 3. O banco de dados utilizado é o PostgreSQL 8.4. O container Java utilizado foi o Tomcat 7.0.

As aplicações candidatas a serem testadas, necessitam atender a um requisito básico: ser desenvolvida em Java Web. Esse requisito é necessário devido a facilidade que o mesmo possui ao permitir filtros que interceptem todas as requisições e respostas, como mencionado por ELBAUM *et al.* (2005). Para aplicações que não utilizam banco de dados, será necessária a criação de um banco, e a comunicação será realizada através do framework JPA, permitindo assim a escolha de uma variedade de

banco de dados. Para os que já utilizam banco de dados com ou sem JPA, poderá ser criado um esquema no próprio banco, ou outro banco de dados, onde serão persistidos os dados das requisições e respostas. A escolha do JPA é devido a sua ampla utilização, simples configuração através de um arquivo de persistência e, como mencionado, não ser limitado a nenhum banco de dados específico.

Caso atendam os requisitos, as aplicações precisam ser configuradas antes de ser cadastradas. A configuração depende de três passos: *plugin*, configuração do banco e configuração do filtro. O *plugin* fornecido pela plataforma é uma biblioteca no formato jar, que deve ser adicionado a aplicação. O jar é responsável pela troca de informações entre a plataforma e a aplicação testada. Porém, só o *plugin* não é suficiente, sendo necessária a configuração do filtro de requisições e respostas, no arquivo descritor da aplicação (web.xml). Esse filtro aponta para uma classe do *plugin*, que insere código Javascript em todas as páginas da aplicação depois de configurada. Para armazenar alguns dados dos testes, um banco deverá ser configurado, ou o mesmo banco da aplicação poderá ser utilizado. A comunicação com o banco é feita através do arquivo de comunicação do JPA (persistence.xml), porém o *plugin* necessita de um arquivo de propriedades fornecendo a unidade de persistência do JPA. Ao cadastrar uma aplicação, um teste é realizado para saber se as configurações estão corretas.

Todas as etapas do processo são realizadas dentro da plataforma, exceto a primeira e a última, realizadas pelo responsável pela aplicação. A partir da configuração, a plataforma consegue acessar a aplicação e mostrar a aplicação agindo como um navegador dentro do navegador do usuário. Esse efeito é atingido através do uso de *iframe*, um componente HTML que acessa páginas de outros domínios e exibe para o usuário. Dessa forma, a aplicação a ser testada e previamente configurada consegue se comunicar com a plataforma, fornecendo os dados necessários.

O processo é orientado a ações e, para atender esse ponto, a tecnologia utilizada foi a linguagem de script JavaScript. Essa linguagem é fortemente acoplada com páginas Web, e possui tratadores de eventos nativos. O código inserido em cada página é justamente para capturar e tratar todos os eventos do usuário, enviando-os para a plataforma. Essa comunicação entre a plataforma e a aplicação testada é feita através de PostMessages, uma forma de comunicação em JavaScript para HTML5, onde duas páginas hospedadas em domínios diferentes podem se comunicar. O

bloqueio entre mensagens de domínios diferentes é uma forma de segurança da tecnologia, e o PostMessages consegue filtrar mensagens recebidas através do domínio, bloqueando páginas indesejadas.

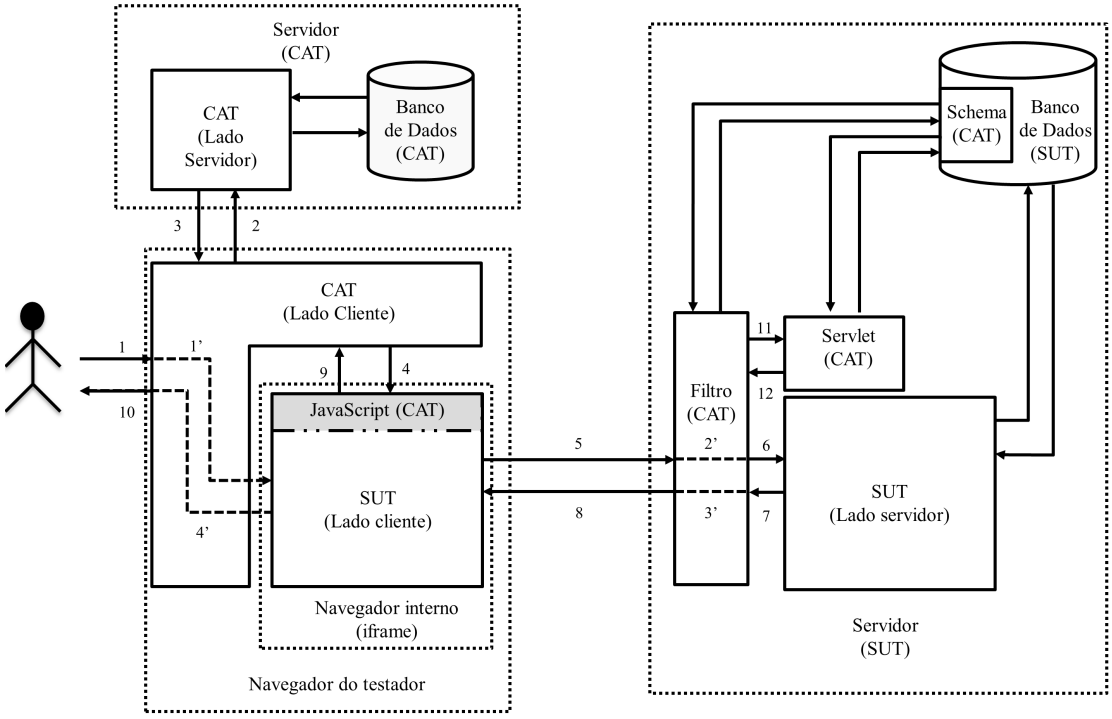

Figura 6 - Arquitetura e funcionamento da ferramenta CAT

A Figura 6 ilustra os principais componentes da plataforma CAT, juntamente com os componentes da aplicação sendo testada (SUT), suas interações e fluxos de chamada. Os fluxos com os bancos de dados não foram enumerados por motivo de simplificação. Os fluxos originais, ou seja, os que seriam percorridos caso não houvesse a plataforma como intermediária entre o testador e a aplicação estão representados pelas setas (ou continuação de setas) tracejadas, enumeradas de 1' a 4', correspondente ao fluxo 3, 4, 7 e 8 da Figura 4. A plataforma em si, como aplicação, utiliza o fluxo 1, 2, 3 e 10, para o usuário (testador) realizar o login, por exemplo.

O seguinte fluxo é executado quando o testador for realizar um teste, obedecendo a numeração da Figura 6:

1. Seleciona a opção de testar na plataforma;
2. CAT (Servidor) busca a URL da aplicação a ser testada;
3. CAT (Servidor) retorna para CAT (Cliente) a URL da SUT;
4. CAT (Cliente) utiliza a URL em seu navegador interno (iframe);
5. O iframe acessa a URL, sendo interceptado pelo Filtro (CAT);

6. O Filtro guarda os dados da requisição do usuário, exceto quando a requisição tem como destino o Servlet (CAT);
7. O SUT retorna uma resposta a requisição, que também é interceptada pelo filtro. O filtro guarda os dados da resposta, e adiciona código JavaScript, que se comunicará com a aplicação CAT (Cliente);
8. O código do SUT acrescido do código JavaScript é enviado para o iframe;
9. Ao terminar de carregar a página no iframe, o código JavaScript sinaliza ações a serem tomadas, como o monitoramento das ações;
10. O iframe é exibido para o usuário.

Após a execução desse fluxo, o testador pode usar o SUT como se estivesse na aplicação pura, enquanto o CAT funciona apenas como uma capa, capturando todos os eventos que o testador está realizando. Todos esses eventos são guardados para futuras reproduções do teste. Quando uma nova requisição da aplicação é realizada (item 5 do fluxo), os passos seguintes também são realizados.

A ideia de utilizar um navegador interno não é nova. Por exemplo, a aplicação DynaRIA (AMALFITANO *et al.*, 2013) utiliza um navegador baseado no Mozilla para realizar rastrear as sessões do usuário. Outra ferramenta que também utiliza dessa ideia é a WebMole (LE BRETON *et al.*, 2013), porém, por se tratar de uma aplicação Web, também utiliza um iframe para tal. Iframe é uma tag HTML que exibe um site dentro de outro, da forma como ele seria exibido se fosse acessado diretamente pelo navegador. As vantagens de utilizar o iframe é que ele é compatível com a maioria dos navegadores, e o testador pode escolher qual ele prefere, oferecendo um leque maior de captura de erros, dado que muitos erros são de incompatibilidade de navegadores. Outra vantagem é não precisar instalar nenhum driver, como o Selenium. Dessa forma, o testador não precisa configurar nada, apenas acessar a plataforma.

4.6.2.Etapas

A plataforma foi construída buscando atender ao processo, buscando atender a cada etapa, onde as escolhas que foram realizadas nos pontos flexíveis serão justificadas, enquanto as implementações dos pontos rígidos apenas descritas. As próximas seções detalham cada uma das etapas do processo e suas respectivas formas de implantação através da ferramenta.

I - Escolher funcionalidades / requisitos / especificações a serem testadas

Essa etapa não é realizada dentro da plataforma, ficando a cargo dos responsáveis pela aplicação pensar quais seriam as funcionalidades desejadas para serem testadas pela multidão. A escolha depende de como está a aplicação, se está finalizada ou não, ou se deseja consolidar determinada parte apenas da aplicação.

II – Definir escopo / roteiro dos casos de teste

Após definir as funcionalidades, a ferramenta permite cadastrar um caso de teste base. É composta por uma tela onde o responsável pela aplicação consegue acessá-la, percorrer alguns passos iniciais, que serão reproduzidos automaticamente para os testadores, e finalmente percorrer as funcionalidades que deseja que o testador teste, preenchendo valores nos campos apenas para percorrer o caminho necessário. Os testadores percorrerão esses mesmos caminhos. Ao término do cadastro do caso de teste base, o roteiro deve ser preenchido, que será exibido para o testador antes do cadastro dos casos de teste.

III – Escolher motivações da multidão

Duas formas fortemente ligadas motivam a multidão ao uso da ferramenta: recompensa e competição. Os testadores possuem uma pontuação dentro de cada aplicação a ser testada, gerando uma classificação que é sempre exibida aos testadores, motivando-os a fazer mais para que seus pontos aumentem. Além disso, uma recompensa é dada, que pode ser para o primeiro lugar, ou para os n primeiros colocados, podendo variar. A ferramenta permite preencher a recompensa, para que os testadores saibam quais são.

Para pontuar, existem três formas: (i) cadastro de caso de teste, (ii) avaliação se o resultado obtido é o resultado esperado e (iii) avaliação da importância do caso de teste. Para o cadastro de caso de teste, uma pontuação fixa pode ser definida, com o testador ganhando os pontos para cada caso de teste criado. Outra forma de fazer, para evitar que um testador crie muitos casos de teste, é definir uma pontuação que obedeça a uma função logarítmica, ou seja, o ganho com novos casos de teste é menor. Uma terceira maneira ainda é computar a importância dos casos de teste dados pelos outros avaliadores.

Para a avaliação do resultado obtido, a pontuação é um valor fixo, atribuído apenas quando o voto é o mesmo da multidão. Porém, o voto da multidão não é conhecido, para não influenciar, obedecendo ao princípio da independência do *crowdsourcing*. A avaliação da importância pontua da mesma forma, porém, por se tratar de um valor numérico, a média da avaliação da multidão é considerada, sendo a pontuação atribuída para os testadores que avaliaram próximos da média.

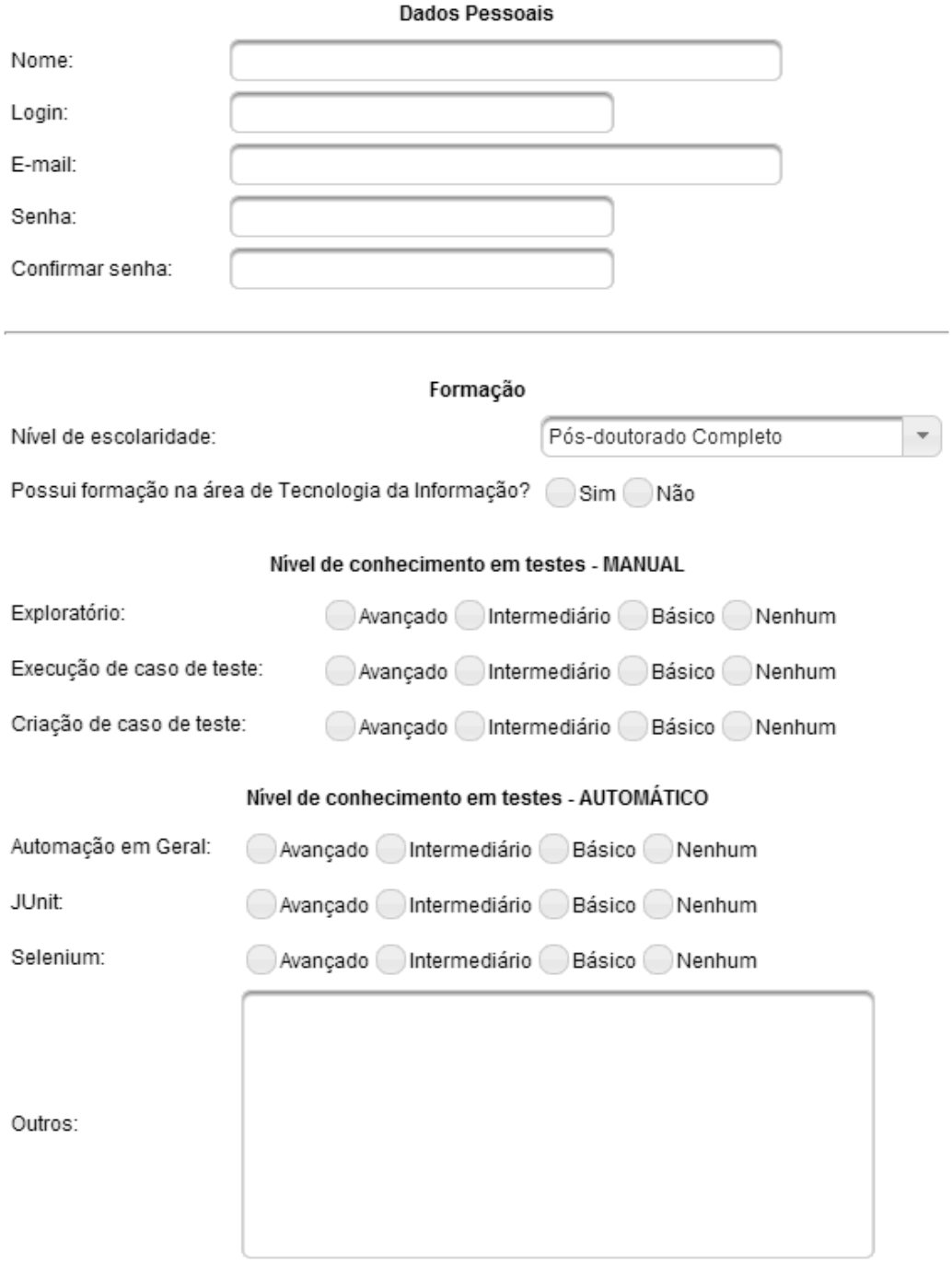

Figura 7 - Formulário de cadastro de usuário da ferramenta CAT

IV – Definir qualificações dos testadores

Para o cadastro de um usuário, um pequeno formulário de competências foi desenvolvido. A Figura 7 exibe o formulário de competências a ser preenchido no ato do cadastro, juntamente com dados pessoais como nome, login, e-mail e senha.

Apesar do preenchimento obrigatório do formulário de competências, a plataforma não exige nenhum requisito especial é para que ele seja um testador de alguma aplicação. O formulário possui o papel de avaliar se as pessoas que são de TI e/ou possuem conhecimento em automação de testes tem desempenho melhor que as outras.

V – Definir formas do testador informar o erro

A ferramenta não fornece formas para que o testador, ao criar um caso de teste, informe o erro. Por outro lado, para o testador que está reproduzindo o caso de teste, a ferramenta disponibiliza duas formas. A primeira é escolhendo se o resultado obtido com a reprodução do caso de teste é o resultado esperado, de acordo com o roteiro definido e a descrição do caso de teste. A segunda é um campo textual, onde pode informar livremente quais erros foram encontrados, uma espécie de *Bug Tracker*. A plataforma não possui outra forma de informar o erro.

VI – Definir formas de priorização / redução da suíte de casos de teste

A ferramenta possui apenas uma forma de priorização e redução da suíte de casos de teste, que é através da multidão. Ao término da execução de cada caso de teste, o testador informará o erro ou não, como definido na etapa anterior, e também informará, caso ache conveniente, um valor de importância para o caso de teste, no esquema de 1 a 5 estrelas. Esse valor será usado para classificar os melhores casos de teste, como uma forma de priorização. A plataforma não possui outra forma de priorização.

VII – Disponibilizar aplicação para testes

Após os cadastros dos casos de teste base pelo responsável pela aplicação e a definição das recompensas, a aplicação estará disponível para cadastro e execução dos casos de teste. Essa disponibilidade é determinada por um tempo específico e, durante

esse tempo, pode ser livre, onde os testadores podem cadastrar e/ou executar os casos de teste livremente, ou fechada, onde determinada fatia do tempo a aplicação estará disponível apenas para cadastro dos casos de teste e a fatia restante apenas para execução dos mesmos. A escolha fica a critério do responsável pela aplicação.

VIII – Escolher escopo / roteiro

Uma vez disponibilizada a aplicação, o testador da multidão pode cadastrar os casos de teste ou executá-los. Para isso, deve escolher um dos casos de teste base disponíveis. A plataforma exibe o roteiro do caso de teste base para que ele leia e siga o roteiro. A etapa seguinte pode ser a criação de um caso de teste ou a escolha de um caso de teste para executar. A tela de criação de caso de teste possui um botão para que ele possa checar sempre o roteiro, para que não fuja do escopo do mesmo, da mesma forma que a tela de execução do caso de teste.

IX – Criar caso de teste

Ao selecionar o caso de teste base a partir de uma listagem, seguida da seleção da opção para criar um caso de teste, a plataforma mostrará a aplicação e executará os passos iniciais definidos pelo responsável pela multidão, navegando até a primeira tela que o testador precisará interagir.

O testador tentará percorrer exatamente o mesmo caminho que o testador responsável percorreu quando criou o caso de teste base. A cada requisição, um componente chamado comparador de URL é executado, justamente para verificar se a URL do caso de teste é a mesma da base. Caso a URL não seja a mesma, o caso de teste é finalizado, caso seja, o caso de teste continua, até que a última URL seja atingida, finalizando o teste.

Todas as ações são rastreadas pela plataforma: cliques, teclas digitadas, valores dos campos trocados, rolagens horizontais e verticais, submissão de formulários, foco e perda de foco dos campos. Cada uma dessas ações é armazenada para as futuras execuções. Além disso, o navegador e sistema operacional que o testador está utilizando no momento da criação do caso de teste são armazenados, visando comparações.

X – Descrever caso de teste

Ao término do cadastro do caso de teste, a plataforma exibe um campo para

que o testador preencha uma descrição, o objetivo do caso de teste. Nesse campo é desencorajado mencionar os erros percebidos por ele, como será mais bem explicado através dos experimentos. Essa etapa é bastante importante, pois é através da descrição que os outros testadores irão avaliar se houve ou não erro.

XI – Escolher caso de teste

Após escolher o escopo (etapa VIII), o testador tem a opção de criar um caso de teste ou escolher um caso de teste para executar. Caso a segunda opção seja escolhida, uma listagem de casos de teste cadastrados que obedecem àquele escopo é exibida. A listagem possui a descrição do caso de teste fornecida pelo criador, fator que deve ser levado em conta na escolha. A ferramenta não exibe na listagem os casos de teste criados pelo próprio testador, para que ele não possa executá-los.

XII – Executar caso de teste

A partir dos casos de teste criados, um testador poderá executar casos de testes de outros testadores, para que possa avaliar. A execução é mostrada da mesma forma que os cadastros de caso de teste, através de um *iframe*. Porém, o testador agora não poderá interagir com a aplicação, poderá apenas assistir a aplicação sendo manipulada automaticamente pela plataforma, reproduzindo todos os eventos gravados, mostrando onde cliques foram realizados e campos de texto tendo seus valores preenchidos.

A execução não é em tempo real, pois poderia levar um tempo grande, uma vez que o criador pode ter ficado parado algum tempo em determinada tela. As ações têm intervalos fixos entre elas, e a velocidade desse intervalo é escolhida pelo testador que estiver executando, da forma que melhor lhe convier, para que possa observar cuidadosamente as ações.

XIII – Informar erro

Após a execução do caso de teste, o testador informa se o resultado obtido ao término da execução do caso de teste era o resultado esperado por ele, em uma percepção pessoal, levando em conta a descrição do caso de teste e o roteiro do caso de teste base. Além disso, caso tenha verificado a presença de um erro, a descrição do mesmo é possível através de um campo textual livre.

Esses dados não são disponibilizados para nenhum testador, para que não haja influência no resultado final. Os usuários que criaram os casos de teste ou avaliaram

também não são exibidos, visando dificultar qualquer fraude. O nome do usuário apenas aparece no ranking exibido como forma competição. Porém, nada se pode fazer quanto a comunicação externa a plataforma, podendo influenciar no resultado caso haja uma combinação.

XIV – Preencher dados para priorização / redução da suíte de casos de teste

Ao mesmo tempo que pode-se informar o erro, há a disponibilidade da avaliar a importância do caso de teste. Como mencionado anteriormente, o modelo 5 estrelas é disponibilizado, por ser facilmente convertido em um inteiro. Ao término do período de testes, duas classificações dos casos de teste são montadas: a primeira para a forma absoluta, levando em conta apenas a média obtida nas avaliações dos casos de teste, e a segunda para a forma relativa, recalculando as notas que cada testador forneceu através da média das avaliações de cada um.

XV – Executar forma de priorização / redução da suíte de casos de teste

A forma de priorização e redução que independe da multidão não é incluída na ferramenta CAT, ficando a cargo do responsável pela aplicação fazê-lo externamente.

4.7.Evolução

O processo e a ferramenta foram sofrendo modificações ao longo do processo experimental, se adequando e melhorando, para que os resultados fossem mais satisfatórios, assim como a usabilidade por parte dos usuários. A versão exibida no presente capítulo é a versão final do mesmo, após as evoluções. Evoluções essas que estão descritas no capítulo seguinte, junto aos próprios experimentos. Os experimentos estão descritos em ordem cronológica de execução, de forma que a evolução fique mais clara.

A Tabela 4 mostra cada etapa do processo e como cada uma dessas etapas foi desenvolvida na ferramenta CAT. As primeira e última etapas, como mencionado anteriormente, não são contempladas pela ferramenta. As demais estão resumidas na tabela.

Tabela 4: Etapas do processo e técnicas correspondentes na ferramenta CAT

Etapa do processo	Técnica na ferramenta CAT
I - Escolher funcionalidades / requisitos / especificações a serem testadas	Não suportado pela ferramenta.
II – Definir escopo / roteiro dos casos de teste	Através de um iframe contendo a aplicação, o responsável percorre o caminho do roteiro a ser seguido. Após percorrer, um campo textual deve ser preenchido com a explicação de como devem ser os casos de teste sobre o respectivo caso de teste base.
III – Escolher motivações da multidão	Competição, com pontos sendo atribuídos para criação e avaliação dos casos de teste, contabilizados para a formação de um *ranking*; Pagamento, através de uma recompensa informada em um campo textual.
IV – Definir qualificações dos testadores	Formulário de cadastro dos usuários, onde determinados conhecimentos em teste podem ser informados.
V – Definir formas do testador informar o erro	Para o testador criador: não há; Para o testador executor: campo textual para descrição do erro e declarar se o resultado obtido é o resultado esperado.
VI – Definir formas de priorização / redução da suíte de casos de teste	Modelo 5 estrelas para o testador executor.
VII – Disponibilizar aplicação para testes	Selecionando o tempo que a aplicação ficará disponível, e se esse tempo é dividido entre as fases de criação e avaliação dos testes ou não.
VIII – Escolher escopo / roteiro	Escolha através de uma listagem de roteiros.
IX – Criar caso de teste	Interação com a aplicação através de um iframe, onde todas as ações do testador são gravadas via JavaScript.
X – Descrever caso de teste	Campo textual para descrição do caso de teste, onde um erro, caso tenha ocorrido, não deve ser informado.
XI – Escolher caso de teste	Escolha através de uma listagem dos casos de teste contendo a descrição dos mesmos.
XII – Executar caso de teste	Reprodução das ações gravadas via JavaScript, dentro de um iframe contendo a aplicação, como se um usuário real estivesse manipulando a aplicação.
XIII – Informar erro	Opção para selecionar se o resultado obtido foi o resultado esperado; Campo textual para descrição do erro.
XIV – Preencher dados para priorização / redução da suíte de casos de teste	Modelo 5 estrelas para avaliação do testador executor.
XV – Executar forma de priorização / redução da suíte de casos de teste	Não suportado pela ferramenta.

5.Experimentos

A partir da ferramenta criada, é possível comparar a eficiência do processo, comparando-os com outras técnicas. A seção seguinte mostra um estudo completo sobre testes de aplicação Web desde o ano 2000, ano do primeiro artigo sobre o assunto. Através deste estudo foram escolhidas métricas, aplicações e técnicas para comparações, apresentadas nas seções subsequentes. Feito isso, os experimentos em si são relatados.

5.1.Fundamentos

GAROUSI *et al.* (2013a) apresenta um estudo sobre testes de funcionalidade em aplicações Web, desde o ano 2000, ano da primeira publicação sobre o tema, até agosto do ano de 2011. Nesse estudo, é realizado um mapeamento sistemático (SM), um método para revisar, classificar e estruturar artigos relacionados a um campo específico da pesquisa, nesse caso, teste de aplicações Web. Apesar de toda uma sistematização do processo, da mesma forma que é feita em uma revisão sistemática, algumas diferenças existem entre eles, mais especificamente nos objetivos.

Mapeamentos sistemáticos geralmente possuem mais questões, e questões mais abrangentes, classificando os estudos de forma suficientemente detalhada, para que as perguntas possam ser respondidas. Revisões sistemáticas, por sua vez, possuem questões mais específicas, incluindo uma análise mais profunda de um tema, com sínteses descritivas, enquanto SMs possuem totalização e gráfico como formas de síntese (KITCHENHAM & CHARTERS, 2007).

O estudo apresenta duas questões, na qual uma busca o espaço de pesquisa na literatura em testes de software Web e outra as tendências e dados demográficos das publicações. A primeira é dividida em 16 questões menores e a segunda em 5 outras questões, cada qual possuindo um domínio específico, e suas respostas estão disponibilizadas em um repositório online (GAROUSI *et al.*, 2013b). Exemplos de questões e respostas apresentadas são: quais ferramentas apresentadas nos estudos estão disponíveis para download, quais aplicações foram utilizadas para testes (SUT – *Software Under Test*), tipos de artefatos gerados (casos de teste, entradas de teste, resultados esperados), o foco do teste (servidor ou cliente), nível do teste (unidade, integração ou sistema) e o tipo de métrica utilizada (cobertura, por exemplo). As

questões foram respondidas para cada um dos 79 artigos selecionados, baseadas apenas nas informações contidas nos mesmos, ou seja, não houve uma busca online por cada ferramenta apresentada para saber se estava disponível.

Uma das vantagens das formas sistemáticas é a possível reprodução e continuação. Assim, foi possível realizar a pesquisa para contemplar os anos seguintes, buscando-se estudos a partir de setembro de 2011 até estudos disponíveis em 18 de junho de 2013, data da pesquisa. Os passos do estudo original foram seguidos, e maiores detalhes podem ser encontrados no Apêndice 2.

Para realizar o experimento do presente estudo, um SUT e uma ferramenta utilizada em outros estudos foram selecionados para comparação. A escolha da ferramenta para comparação considerou apenas aquelas disponíveis online, da mesma forma, a escolha do SUT foi limitada apenas aqueles desenvolvidos em Java, devido à limitação da plataforma.

5.1.1.SUTs

Segundo GAROUSI *et al.* (2013a) em seu SM, dentre os artigos, 176 nomes de SUTs únicos foram identificados enquanto 210 SUTs foram utilizados, indicando que alguns SUT foram utilizados em mais de um artigo. Na sequência do estudo, relatado no Apêndice 2, 65 nomes de SUTs únicos foram identificados em 95 SUTs utilizados. No resultado agregado, em 305 SUTs utilizados, 216 SUTs diferentes foram usados, havendo SUTs utilizados em ambos os estudos. Observando o repositório fornecido por eles, agregado ao resultado da continuação da pesquisa, pôde-se elencar as aplicações que tiveram mais aparições, qual tecnologias elas utilizam e se as mesmas estão disponíveis para download. Os dados obtidos são apresentados na Tabela 5, para aplicações que tiveram pelo menos 5 aparições.

Tabela 5 - SUTs mais mencionados na literatura, baseado nos dados de GAROUSI *et al.* (2013b) e no estudo complementar

Nome do SUT	# de aparições (original + cont.)	Tecnologia (Servidor)	Disponível para download	Observação
Bookstore	15+6=21	Java	Não	Open source, citada como disponível em http://www.gotocode.com.
CPM	8+2=10	Java	Não	Aplicação proprietária para gestão de cursos e alunos.
TuduList	4+6=10	Java	Sim	Open source, disponível em http://www.julien-dubois.com/tudu-lists/
FaqForge	3+5=8	PHP	Sim	Open source, disponível em http://sourceforge.net/projects/faqforge/
Schoolmate	4+4=8	PHP	Sim	Open source, disponível em http://schoolmate.sourceforge.net/
Webchess	4+4=8	PHP	Sim	Open source, disponível em http://webchess.sourceforge.net/
Masplas	6+1=7	Java	Não	Aplicação proprietária para gestão de workshop.
TimeClock	2+4=6	PHP	Sim	Open source, disponível em http://timeclock.sourceforge.net/
The Organizer	3+2=5	Java	Sim	Aplicação exemplo de livro, disponível em http://www.apress.com/downloadable/download/sample/sample_id/839/

A escolha natural de um SUT para comparação seria a aplicação Bookstore, porém ela não está mais disponível[23]. Considerando as aplicações em Java disponíveis para download, TuduList foi a que teve maior presença nos estudos, sendo uma das aplicações escolhidas para serem testadas. A outra aplicação escolhida foi a The Organizer, por atender o requisito de ser em Java e também estar disponível.

5.1.2.Ferramenta para comparação

Dos 79 artigos, 41 apresentavam alguma ferramenta (51,9%), um ponto positivo do estudo, porém apenas 6 dos 41 (14,6%) mencionavam que suas ferramentas estavam disponíveis para download, um ponto negativo (GAROUSI *et al.*, 2013a), sendo duas delas a mesma ferramenta. No SM original, uma ferramenta é

[23] O site http://www.gotocode.com não está disponível na data de 25 de julho de 2013.

dita disponível para download ou não baseado apenas nos dados contidos no estudo. Observando essa limitação, uma pesquisa no Google[24] foi conduzida, combinando os nomes da ferramenta, dos autores e das universidades, resultando em 2 novas ferramentas disponíveis para download, ou seja, na verdade 8 (19,5%) dos 41 estavam disponíveis para download. Na continuação do SM, dos 43 selecionados, 23 apresentavam alguma ferramenta (53,5%), sendo 9 (39,1%) disponíveis para download, sendo uma delas não trazendo este dado no estudo. Das 9, uma ferramenta aparece duas vezes, a mesma que também aparece duas vezes no SM original (Crawljax), além de outra também presente no SM original (DynaRIA). Totalizando 17 aparições, uma ferramenta aparece 4 vezes, outra duas vezes e as outras apenas uma vez, totalizando 13 ferramentas diferentes disponíveis para download.

A Tabela 6 mostra os artigos que disponibilizaram suas ferramentas para download, juntamente com o nome da ferramenta ou do conjunto de ferramentas, se é voltada para testes de aplicação Ajax e a URL para download.

A aplicação Crawljax foi utilizada em quatro estudos, sendo em dois deles citada também a ferramenta ATUSA, que, de acordo com o site, é parte integrante da ferramenta Crawljax. Outras três ferramentas são, na verdade, *plugins* para a ferramenta Crawljax: AutoFLox, JSART e Mutandis. O conjunto de ferramentas CreRIA, CrawlRIA, TestRIA e DynaRIA é citado em um trabalho, sendo a última citada em outro estudo, onde as ferramentas são utilizadas em determinada ordem, ou seja, o resultado de uma é a entrada para outra.

Todas as ferramentas tiveram tentativas de serem utilizadas, porém a maioria falhou. As ferramentas Artemis e AutoDBT não conseguiram ser instaladas no Ubuntu, como recomendado no site. WebTestingExplorer e WebVizOr apresentaram erros ao serem executadas. A falta de documentação de algumas ferramentas foi determinante para que as algumas não conseguissem ser utilizadas, como é o caso de: reAJAX, CreRIA e DynaRIA. WebMole foi configurada corretamente, porém descartada por não apresentar o resultado esperado para o SUT de exemplo que a acompanha. Por sua vez, Rubicon funciona apenas para aplicações desenvolvidas em Ruby on Rails, que possuem seu comportamento descrito em RSpec, enquanto WIT busca testar Portlets, componentes independentes que podem ser acoplados a uma página, interagindo ou não entre si (XIONG *et al.*, 2005).

[24] http://www.google.com/

Tabela 6 - Dados das ferramentas de teste disponíveis para download

Referência	Ferramenta	Ajax	URL
AMALFITANO *et al.* (2010, 2013)	CreRIA, CrawlRIA, TestRIA, DynaRIA	Sim	http://wpage.unina.it/ptramont/downloads.htm
ARTZI *et al.* (2011)	Artemis	Sim	http://brics.dk/artemis/
LE BRETON *et al.* (2013)	WebMole	Sim	http://github.com/WebMole
MARCHETTO & TONELLA (2011)	reAJAX	Sim	http://selab.fbk.eu/marchetto/tools/ajax/reAJAX
MCMASTER & YUAN (2012)	WebTestingExplorer	Sim	http://www.webtestingexplorer.org/
MESBAH & VAN DEURSEN (2009), ROEST *et al.* (2010), MESBAH *et al.* (2012a, 2012b)	Crawljax, ATUSA	Sim	http://crawljax.com/
MIRSHOKRAIE *et al.* (2013)	Mutandis	Não	https://github.com/saltlab/mutandis/
MIRSHOKRAIE & MESBAH (2012)	JSART	Não	http://salt.ece.ubc.ca/content/jsart/
NEAR & JACKSON (2012)	Rubicon	Não	http://people.csail.mit.edu/jnear/rubicon
OCARIZA *et al.* (2012)	AutoFLox	Sim	http://ece.ubc.ca/~frolino/projects/autoflox/
RAN *et al.*(2009)	AutoDBT	Não	http://digital.cs.usu.edu/~cdyreson/pub/AutoDBT/
SPRENKLE *et al.* (2008)	WebVizOr	Não	http://www.eecis.udel.edu/~hiper/webvizor
XIONG *et al.* (2005)	WIT	Não	http://sourceforge.net/projects/wit-ict/files/wit-ict/

Crawljax, por sua vez, conseguiu ser executada com sucesso e vários *plugins* podem ser utilizados, tanto os presentes em outros estudos quanto outros que poderão ser criados especificamente para a comparação. O foco de Crawljax é em aplicações Ajax, diferente da ferramenta proposta que não faz distinção entre o foco. Apesar disso, foi a única ferramenta consolidada a ser encontrada na literatura, e por esse motivo foi escolhida para comparação.

Como o nome da própria ferramenta sugere, baseia-se na técnica de *crawling* para aplicações Ajax. Através de uma URL inicial, busca na página todo possível elemento capaz de disparar uma nova requisição, seja ela síncrona ou assíncrona. Após disparar o evento, a modificação da página, seja ela como um todo (carregando outra página) ou não (carregando apenas novos elementos), é computada. As modificações são analisadas e podem gerar um novo estado não visitado anteriormente ou concluir que o estado já fora visitado. De posse dos estados e

eventos, um diagrama de estados é construído, representando a aplicação como um todo. Formulários são preenchidos com valores aleatórios, porém pequenos trechos de código podem fornecer os dados de entrada para um determinado formulário, por exemplo, formulário de login.

A ferramenta está na versão 3.2[25], que possui poucas diferenças em relação as versões 3.1 e 3.0. Porém, as diferenças para as versões 2.2 e 2.1 são maiores, especialmente no que diz respeito à codificação dos *plugins*. A versão a ser utilizada nos experimentos é a versão 2.2, por possuir mais *plugins* disponíveis. A ferramenta é baseada no Selenium Web Driver, e para os testes realizados nesse trabalho, foi utilizado o ChromeDriver.

Um dos *plugins* utilizado é o CrawlOverview[26], responsável por gerar um relatório de visão global da varredura, montando o diagrama de estados, onde cada estado é uma fotografia da página no instante em que o estado foi identificado, exibindo uma borda ao redor dos elementos passíveis de eventos. A borda pode ser de três cores: verde, representando que a ação gera um novo estado; ciano, representando que a ação volta ao estado anterior, e amarelo, representando que a ação não modifica o estado. Este *plugin* foi desenvolvido pela própria equipe que desenvolve o Crawljax, tendo sido incorporado a partir da versão 3.0.

Outro *plugin* usado é o TestCaseGenerator[27], responsável por transformar a varredura em testes unitários JUnit, também desenvolvido pela equipe Crawljax, não disponível para a versão 3.0 ou superior. A partir dos eventos identificados e dos estados, cria uma classe Java com vários métodos de teste unitários. Este *plugin* possui um Oráculo dentro dele, que compara o estado atual do teste com o estado esperado (estado quando foi gerada a classe). Porém, esse Oráculo necessita ser melhorado, pois se trata da comparação textual do código fonte HTML da página, limitando seu uso a certos tipos de aplicação, como será explicado a frente.

5.1.3.Métricas

Para comparar e avaliar as técnicas e ferramentas de teste, algumas métricas são habitualmente usadas. Dentre as mais utilizadas estão a comparação manual, a

[25] Versão mais atual em 25 de julho de 2012, lançada em 20 de julho de 2013.

[26] Disponível em https://github.com/crawljax/crawljax/tree/master/plugins/crawloverview-plugin

[27] Disponível em https://github.com/crawljax/testcasegenerator-plugin

cobertura e a *fault seeding*, como mostra o gráfico da Figura 8. Como o nome traduz, a comparação manual não é automática, portanto, não será utilizada nos experimentos. As métricas utilizadas serão a cobertura e a *fault seeding*, explicadas nas seções 3.3.1 e 3.3.2 respectivamente.

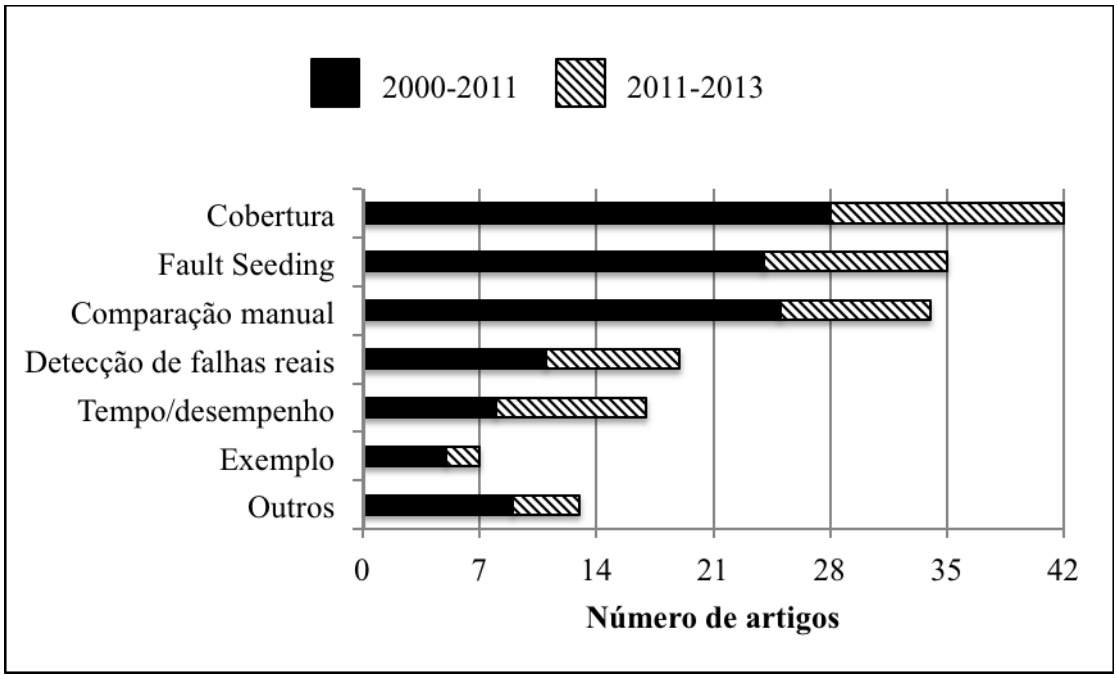

Figura 8 - Distribuição dos métodos de avaliação de testes para aplicações Web, baseado nos dados de GAROUSI *et al.* (2013b) e na continuação do estudo

Para cada uma das aplicações, as falhas artificiais inseridas serão mostradas. Para a cobertura, todas as aplicações irão utilizar uma ferramenta chamada Cobertura (DOLINER, 2013). Esta ferramenta identifica a quantidade de vezes que cada linha do código Java foi executada, assim como a quantidade de vezes que cada condição foi executada para cada saída (verdadeira e falsa), ou seja, a ferramenta provê os dados da cobertura de sentença e da cobertura de condição, explicadas na seção 3.2.2. Para verificar a cobertura, a ferramenta instrumenta cada classe Java após serem compiladas para *bytecode*, possibilitando seu uso em aplicações Web. A partir de um arquivo gerado, é possível gerar relatório em XML e HTML.

5.2.Aplicações

Três aplicações foram utilizadas para os testes, sendo uma elaborada pelo autor, outra *open source* utilizada em outros artigos, e a terceira uma aplicação elaborada como parte de um livro, também utilizada em outros trabalhos. Cada uma

das três aplicações foi testada pelas duas ferramentas, para que pudesse haver uma comparação dos métodos.

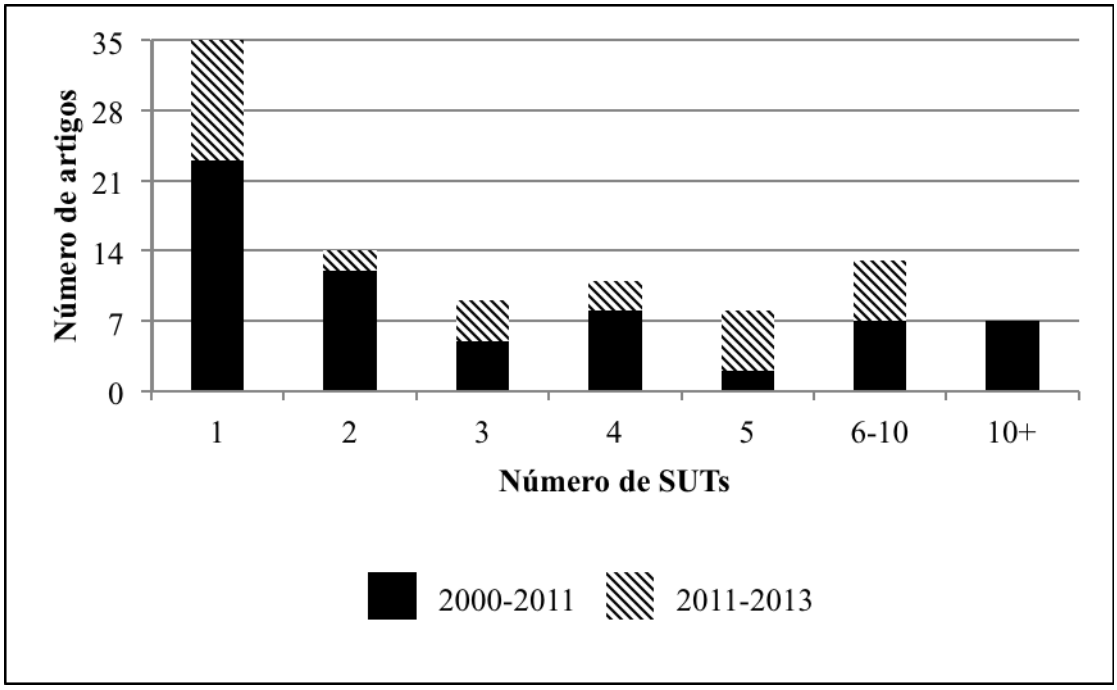

Figura 9 - Número de SUTs utilizados para testes em cada artigo, baseado nos dados de GAROUSEI *et al.* (2013b) e na continuação do estudo

A média de SUTs por estudo é de 4,3, possuindo mediana igual a 2 e desvio padrão igual a 6,3. Dos 97 estudos que apresentaram SUTs, 58 deles utilizaram de 1 a 3 SUTS, correspondente a 59,8%. Três foi considerado um número razoável de SUTs para aplicar a técnica, e cada um deles são descritos nas seções seguintes.

5.2.1.Triângulo

A primeira aplicação testada é uma aplicação simples, elaborada pelo próprio autor, contendo apenas uma página JSP e um *Servlet*. A página contem três campos de texto e um botão, e o *Servlet* verifica se os três campos formam os lados de um triângulo, dizendo ainda qual o tipo do mesmo. Nenhuma validação JavaScript é utilizada, nem limites de tamanho do campo no código HTML.

A única classe presente possui 54 linhas de código, ignorando as linhas brancas e as que contêm apenas comentários, porém contabilizando as linhas que possuem apenas chaves, distribuídas em 3 métodos não estáticos. Esses números foram obtidos através da utilização de um *plugin* para Eclipse denominado Eclipse Metrics (ECLIPSE METRICS, 2010), também utilizado para medição das outras

aplicações.

5.2.2.TuduList

A aplicação TuduList (DUBOIS, 2013) é uma aplicação Web para gerenciar listas de tarefas, com opções de compartilhamento entre usuários. Seu código é desenvolvido em Java e disponibilizado online, porém a aplicação também é distribuída na forma de um WAR (Web ARchive), ou seja, pronta para rodar em qualquer servidor Java, apesar de seu desenvolvimento ser focado para funcionar no *container* Tomcat versão 6.0. Utiliza o framework Hibernate para conexão com o banco, uma implementação do framework JPA, porém, o desenvolvimento é focado nos bancos MySQL e HSQLDB. Utiliza o framework DWR[28] para comunicação Ajax do código JavaScript com o código Java, utilizando o framework Spring.

Figura 10 - Tela principal da aplicação TuduList

Dentro dos 18 pacotes, há 60 classes e 11 interfaces, contabilizando ao todo 3258 linhas de código e 346 métodos, sendo 1 estático. Possui 11 JSPs representando páginas, e 7 JSPs como componentes das páginas, por exemplo, cabeçalho e menu. A contagem das JSPs foi feita manualmente, enquanto a das classes foi feita através do Eclipse Metrics.

A aplicação possui uma tela inicial de login, onde há também uma opção para cadastro. Ao logar, a tela principal é apresentada, como ilustrada na Figura 10. Na página, é possível criar, editar, apagar e compartilhar listas, realizar cópias e restaurá-

[28] Direct Web Remoting - http://directwebremoting.org/

las, adicionar tarefas às listagens, apagar e editar as tarefas, além de marcá-las como completas, tudo feito na mesma página HTML, através de requisições Ajax.

5.2.3. The Organizer

A aplicação The Organizer foi desenvolvida como parte integrante do livro de ZAMMETTI (2006), mais precisamente no capítulo 8. A aplicação é um gerenciador de informações pessoal, possibilitando incluir anotações, gerenciar tarefas, armazenar contatos e agendar compromissos, tudo baseado em Ajax, utilizando a biblioteca Prototype. Seu código é em Java, e o banco de dados é o HSQLDB, porém foi modificado para que utilizasse o banco de dados PostgreSQL.

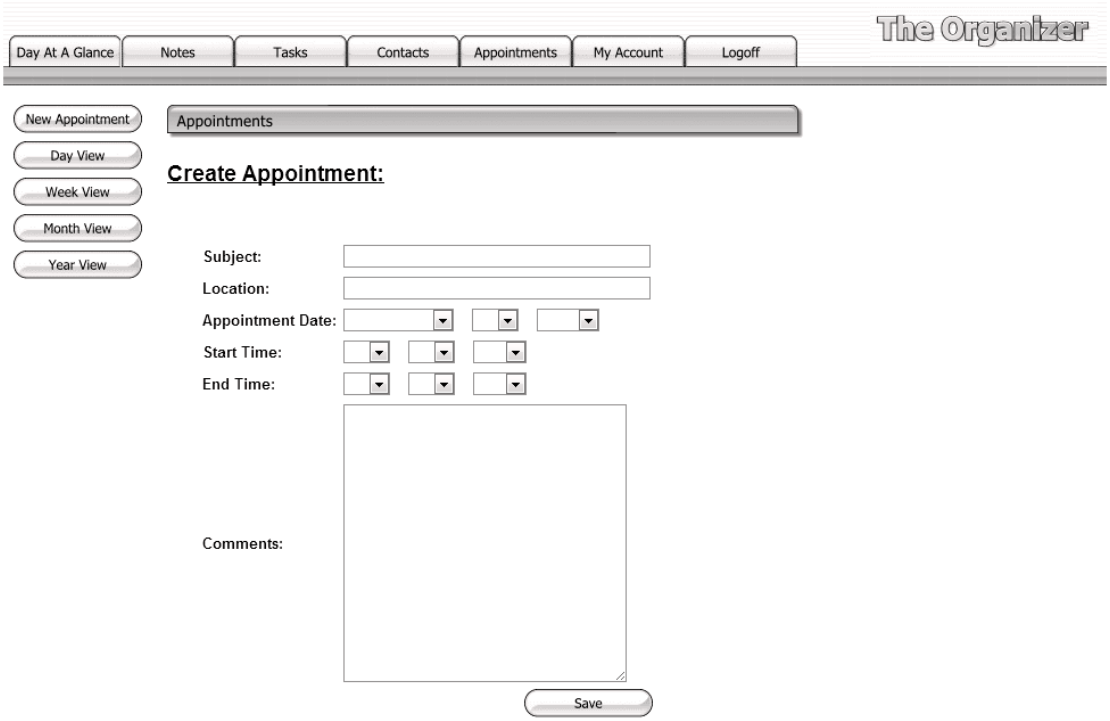

Figura 11 - Tela de criação de compromisso da aplicação The Organizer

Composto por 5 pacotes, com 22 classes e nenhuma interface, em um total de 2927 linhas de código e 404 métodos, sendo 5 estáticos. Possui 14 JSPs representando páginas, e 1 representando o menu. A contagem das JSPs foi feita manualmente, enquanto a das classes foi feita através do Eclipse Metrics.

Ao iniciar, a aplicação exibe uma tela de login, com opção para cadastro de novos usuários. Ao logar, uma tela com o resumo do dia é exibida. Organizada em abas, é possível visualizar separadamente os compromissos, tarefas, anotações e contatos. A Figura 11 ilustra a tela de cadastro de compromissos da aplicação.

5.3.Experimentos com Triângulo

Por ser uma aplicação simples, Triângulo terá outros testes como parâmetros de comparação. Foram criados casos de teste caixa-branca para percorrer todos os caminhos (*path coverage*) e testes caixa-preta. O código da aplicação pode ser verificado na Figura 12, acompanhado do grafo do fluxo de controle, onde o número de cada nó corresponde ao número contido no comentário de algumas linhas.

Apenas um tipo de erro foi inserido no código, presente nas linhas 33, 34 e 35. Ao inserir valores não inteiros, por exemplo, “2,3” ou “a”, uma exceção do tipo *NumberFormatException* é lançada e não tratada.

```
1 package br.ufrj.cos.girao.triangulo;
2
3 import java.io.IOException;
4 import javax.servlet.RequestDispatcher;
5 import javax.servlet.ServletException;
6 import javax.servlet.annotation.WebServlet;
7 import javax.servlet.http.HttpServlet;
8 import javax.servlet.http.HttpServletRequest;
9 import javax.servlet.http.HttpServletResponse;
10
11 @WebServlet("/VerificarTriangulo")
12 public class VerificarTriangulo extends HttpServlet {
13     private static final long serialVersionUID = 1L;
14
15     public VerificarTriangulo() {
16         super();
17     }
18
19     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
20         String _a = request.getParameter("a");
21         String _b = request.getParameter("b");
22         String _c = request.getParameter("c");
23
24         request.setAttribute("msg", triangle(_a,_b,_c));
25         RequestDispatcher rd = getServletContext().getRequestDispatcher("/index.jsp");
26         rd.forward(request, response);
27     }
28
29     public String triangle(String _a, String _b, String _c){
30         String msg = "";
31         int a,b,c;
32         boolean isTriangle;
33         a = Integer.parseInt(_a);//4
34         b = Integer.parseInt(_b);//5
35         c = Integer.parseInt(_c);//6
36         if (a < b + c && b < a + c && c < a + b){//7
37             isTriangle = true;//8
38         }else{//9
39             isTriangle = false;//9
40         }
41         if (isTriangle){//10
42             String tipo = "";
43             if (a == b && b == c){//11
44                 tipo = "equilatero";//12
45             }else if (a != b && a != c && b != c){//13
46                 tipo = "escaleno";//14
47             }else{ //15
48                 tipo = "isosceles";//15
49             }
50             msg = _a+", "+_b+", "+_c+" e um triangulo "+tipo+"!";
51         }
52         else{
53             msg = _a+", "+_b+", "+_c+" nao e um triangulo.";//16
54         }
55         return msg;//17
56     }
57 }
```

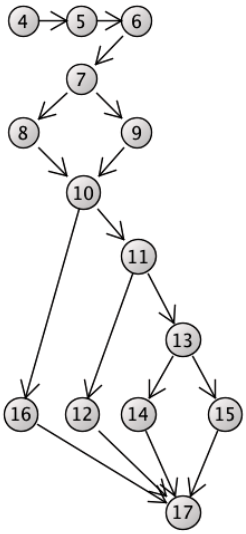

Figura 12 - Código fonte do servlet VerificarTriângulo e grafo do fluxo de controle do método triangle

5.3.1. *Path coverage*

Observando o grafo da Figura 12, são identificados 8 caminhos diferentes a serem percorridos na abordagem *path coverage*, listados na Tabela 7. Metade dos caminhos é inatingível (1, 6, 7, 8), pela lógica do algoritmo, e outra metade é possível (2, 3, 4, 5), um para cada tipo de triângulo e um para lados que não formam um triângulo.

Quatro casos de teste foram criados utilizando JUnit e Selenium, baseados nos caminhos do código. O único erro presente não foi identificado, porém a cobertura de sentença do código foi de 100%, e a cobertura de condição foi de 77%, de acordo com a ferramenta Cobertura. A razão dos 77% é que os nós 7 e 13 obtiveram 66% cada, pois apenas as primeiras condições (“a<b+c” e “a!=b”, respectivamente) foram testadas para valores verdadeiro e falso, sendo as outras apenas para valores verdadeiros.

Tabela 7 - *Path coverage* para o método triangle

Caminho	#	Entrada			Saída esperada	Saída obtida
4,5,6,7,8,10,16,17	1	-			-	-
4,5,6,7,8,10,11,12,17	2	3	3	3	Equilátero	Equilátero
4,5,6,7,8,10,11,13,14,17	3	3	4	5	Escaleno	Escaleno
4,5,6,7,8,10,11,13,15,17	4	3	3	4	Isósceles	Isósceles
4-5-6-7-9-10-16-17	5	3	1	1	Não é um triângulo	Não é um triângulo
4-5-6-7-9-10-11-12-17	6	-			-	-
4,5,6,7,9,10,11,13,14,17	7	-			-	-
4,5,6,7,9,10,11,13,15,17	8	-			-	-

5.3.2. Caixa-preta

Para os testes de caixa-preta, casos de teste foram inspirados nas sugestões de MYERS (2004), exibidas na Tabela 8. Os 16 casos foram criados utilizando JUnit e Selenium da mesma forma que os testes *path coverage*.

O único erro presente foi identificado por dois casos de testes, 15 e 16, onde o primeiro preencheu com valores não inteiros e o segundo deixou um dos campos em branco. O mesmo erro também seria identificado caso uma *string* qualquer fosse inserida, e.g., “alfa”. A cobertura de sentença e de condição foram ambas de 100%.

Tabela 8 - Casos de teste para um programa de verificação de um triângulo, baseado em MYERS (2004)

Descrição	#	Entrada			Saída esperada
Triângulo escaleno válido	1	3	4	5	Escaleno
Triângulo equilátero válido	2	3	3	3	Equilátero
Triângulo isósceles válido	3	3	3	4	Isósceles
Permutações de um triângulo isósceles	Igual ao 3				
	4	3	4	3	Isósceles
	5	4	3	3	Isósceles
Um dos lados é 0	6	0	3	3	Não é um triângulo
Um dos lados é negativo	7	-1	3	3	Não é um triângulo
Soma de dois lados é igual ao terceiro	8	1	2	3	Não é um triângulo
Três testes para as permutações do teste acima	Igual ao 8				
	9	3	2	1	Não é um triângulo
	10	1	3	2	Não é um triângulo
Três lados positivos, onde a soma de dois lados é menor que o terceiro	11	1	2	4	Não é um triângulo
Três testes para as permutações do teste acima	Igual ao 11				
	12	4	2	1	Não é um triângulo
	13	1	4	2	Não é um triângulo
Todos os lados iguais a 0	14	0	0	0	Não é um triângulo
Com números não inteiros	15	2.3	3.1	2.7	Não é um triângulo
Com número errado de parâmetros (apenas dois lados)	16	2	3	-	Não é um triângulo

5.3.3.Crawljax

Para os testes com a ferramenta Crawljax, os *plugins* CrawlOverview e TestCaseGenerator foram utilizados. A ferramenta, por padrão, aciona todas as âncoras e botões do HTML, e todos os campos são preenchidos com valores aleatórios. Como a aplicação é de verificação do tipo de triângulo, valores aleatórios causariam sempre e apenas o erro, não verificando os tipos de triângulo.

Para resolver o problema das entradas, um código para gerar entradas numéricas aleatórias foi elaborado. O código gera 200 entradas para cada um dos três campos, de acordo com as seguintes regras: 10% de chance de ser um valor vazio, 10% de chance de ser -1, 10% de chance de ser 0 e 70% de chance de ser um valor aleatório entre 1 e 5. O erro provocado pelo valor vazio é o mesmo provocado por um valor não numérico. Foram executadas 10 rodadas de testes com a ferramenta. Para cada rodada, a ferramenta Crawljax foi executada com a aplicação sem erro, e uma classe de teste com código JUnit foi gerada. Logo após, o erro foi inserido, e a classe de teste gerada foi executada. A Tabela 9 sumariza os resultados.

Tabela 9 - Resultado do uso da ferramenta Crawljax na aplicação Triângulo

Rodada	1	2	3	4	5	6	7	8	9	10	Média
Cobertura de sentença (%)	100	100	100	100	96	100	100	100	100	100	**99,6**
Cobertura de condição (%)	100	100	100	100	94	100	100	100	100	100	**99,4**
# Falhas nos casos de teste	37	35	25	36	30	25	24	25	31	21	**28,9**
#Não triângulo	105	94	95	92	107	106	114	107	102	108	**103**
# Equilátero	5	8	3	3	0	7	6	8	7	6	**5,3**
# Isósceles	40	48	46	50	49	42	41	36	46	46	**44,4**
# Escaleno	13	15	31	19	14	20	15	24	14	19	**18,4**
# Estados	115	123	120	107	113	118	119	114	126	127	**118,2**

Para as 10 rodadas, apenas uma não obteve 100% de cobertura de sentença e condição, pois as regras de geração de entradas aleatórias elaboradas não geraram nenhum caso para teste de triângulo equilátero. Além disso, o erro foi identificado em todas as rodadas.

A aplicação, após verificar se os dados formam ou não um triângulo e de qual tipo, exibe na tela uma mensagem semelhante a: “ 3, 4, 5 é um triângulo equilátero”. Para cada entrada, os três primeiros números variam de acordo com a entrada. Por esse motivo, o número de estados gerados foi alto, mostrando que o comparador de estados funciona na mesma forma que o Oráculo, ou seja, comparando textualmente os códigos fontes. Um comparador mais inteligente poderia ser capaz de identificar dois estados como sendo um só, ao constatar que a mensagem depende da entrada, e que várias mensagens tem o mesmo formato: “X, Y, Z é um triângulo equilátero”. Nesse caso, apenas cinco estados existiriam: equilátero, isósceles, escaleno, não é um triângulo e dados inválidos, o último para a aplicação sem erro.

5.3.4.CAT

Para a ferramenta CAT, foram necessárias algumas rodadas até a ferramenta se adequar a usabilidade dos usuários, assim como também a competitividade. Ao final de cada rodada, um formulário de satisfação e opinião foi disponibilizado para os usuários. O formulário está exposto no Apêndice 3.

5.3.4.1.Rodada 1

Cinco pessoas foram reunidas em uma sala para que utilizassem o sistema

durante 8 minutos: dois graduandos na área de Tecnologia da Informação (TI), um mestrando na área de TI, uma pessoa com mestrado fora da área de TI e um doutorando na área de TI.

A pontuação era dada de três formas:

1. Um ponto a cada caso de teste criado;
2. Dois pontos para a avaliação do resultado, se é o resultado esperado ou não. Os pontos são atribuídos apenas quando a maioria tem a mesma opinião, por exemplo, o testador diz que o teste teve o resultado não esperado e a maioria dos testadores disse o mesmo;
3. Dois pontos para a avaliação da importância do caso de teste. Os pontos são atribuídos apenas quando a avaliação está próxima da média da multidão, com a diferença máxima de 1 unidade.

A Tabela 10 mostra os dados detalhados do experimento. Vale ressaltar que o usuário 5 não é de TI e, coincidência ou não, não criou nenhum caso de teste. Um problema ocorreu durante o teste, pois o usuário 1 possuía uma versão anterior de um arquivo JavaScript em cache, impossibilitando os outros usuários de executar os casos de teste criados por ele, ou seja, apenas 16 dos 22 casos de teste realmente foram utilizáveis. Ao tentar executar os outros 6 casos de teste, os usuários reclamaram de lentidão no sistema, quando na verdade era uma ausência de ações. Os 16 casos de teste foram avaliados, e pro usuário 1, dos 15 avaliados (resultado esperado) com a maioria, 7 deles apenas ele avaliou.

Tabela 10 - Classificação dos usuários e testes no primeiro experimento da aplicação Triângulo com a ferramenta CAT

Usuário	**1**	**2**	**3**	**4**	**5**	**Total**
# Casos de testes criados	6	5	5	6	0	22
# Casos de teste avaliados (resultado esperado)	16	6	2	2	2	28
# Casos de teste avaliados com a maioria (resultado esperado)	15	4	2	2	2	25
# Casos de teste avaliados (importância)	6	6	2	1	0	15
# Casos de teste avaliados com a maioria (importância)	4	4	2	1	0	11
Placar	**44**	**21**	**13**	**12**	**4**	

Essa rodada mostrou alguns fatores relevantes para a plataforma CAT, assim como as opiniões e sugestões preenchidas no formulário. A maioria das críticas foi à usabilidade da tela de avaliação do caso de teste, quiçá a tela mais importante da plataforma. Outro ponto relevante levantado pelos usuários foi a de um usuário poder simplesmente cadastrar indefinidamente casos de teste, pontuando sempre. Os

números apontam que o voto com a maioria deve ser diferente, levando em consideração apenas quando mais de uma avaliação existe para o caso de teste. De posse dessas conclusões, uma nova rodada foi realizada.

A cobertura de sentença foi de 96% enquanto a cobertura de condição foi de 77%. O erro foi detectado e, todo caso de teste avaliado com o resultado esperado errado apontou a presença do erro. Um fato curioso foi que nenhum caso de teste para triângulos isósceles foi feito, enquanto para o erro, das 48 chamadas ao método, 26 apontaram o erro.

Tabela 11 - Comparação da forma absoluta e da forma relativa da importância para a primeira rodada da aplicação Triângulo

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Forma absoluta	1/1	-	-	-	-	-	-	-	-	-
Forma relativa	0/1	1/1	-	-	-	-	-	-	-	-

A importância dos casos de teste foi contabilizada e ordenada de duas formas, uma levando em consideração a média das avaliações – forma absoluta – e outra levando em consideração o peso dos usuários – forma relativa. Cada caso de teste tem sua média, e duas classificações foram feitas, uma para cada forma. A Tabela 11 mostra a quantidade de erros identificados utilizando apenas uma parcela dos casos de teste, utilizando sempre os primeiros da classificação. A forma absoluta, ao utilizar apenas os dois casos de teste mais bem avaliados (10% dos 16 casos de teste utilizáveis), encontrou o erro, enquanto a forma relativa precisou de três casos de teste (20% dos 16 casos de teste utilizáveis). Para esse caso, a forma absoluta se saiu melhor.

5.3.4.2.Rodada 2

Diferente da rodada anterior, esses testes foram realizados remotamente, ou seja, cada um dos quatro usuários estava em sua casa, com seus computadores pessoais. Na rodada anterior, apenas Windows 7 e o navegador Chrome foram utilizados, enquanto na segunda rodada, 3 combinações de sistema operacional e navegador foram utilizados: Windows 7 e Chrome; Windows 7 e Firefox; e Mac OS e Chrome.

A pontuação continuou sendo atribuída pelos mesmos motivos da rodada anterior, porém a criação dos casos de teste foi reduzida de 1 ponto para 0,2. Uma nova restrição foi inserida, a partir de uma sugestão de usuário na rodada anterior: o

testador pode criar apenas cinco casos de teste logo que a sessão é iniciada. Desse ponto em diante, poderá cadastrar novos casos de tese apenas quando o número de avaliações for igual ou maior que o número de casos de teste cadastrados por ele, incentivando a avaliação de outros casos de teste. Além disso, uma avaliação com a maioria apenas será computada caso haja pelo menos duas avaliações.

Tabela 12 - Classificação dos usuários e testes no segundo experimento da aplicação Triângulo com a ferramenta CAT

Usuário	**1**	**2**	**3**	**4**	**Total**
# Casos de testes criados	7	3	3	0	13
# Casos de teste avaliados (resultado esperado)	6	5	3	3	17
# Casos de teste avaliados com a maioria (resultado esperado)	6	5	3	3	17
# Casos de teste avaliados (importância)	6	5	3	3	17
# Casos de teste avaliados com a maioria (importância)	5	5	2	2	14
Placar	**23,4**	**20,6**	**10,6**	**10**	

A rodada durou 15 minutos, e contou com quatro pessoas, todos mestrandos na área de TI, e não participantes da rodada anterior. O resumo da rodada está na Tabela 12, mostrando o número de casos de teste e avaliações criadas para cada usuário. Apesar do número menor de casos de teste em relação a rodada anterior, a cobertura foi maior, sendo a cobertura de sentença de 100% e a cobertura de condição de 88%, e o único erro foi encontrado, melhorando o teste em relação a rodada anterior. A recompensa para as duas rodadas foi uma barra de chocolate para o primeiro colocado.

Os usuários, ao se cadastrarem, preenchem um formulário sobre os conhecimentos em teste, questionando o nível de conhecimento em Selenium, JUnit e em automação de testes em geral, com valores subjetivos, sendo eles nenhum, básico, intermediário e avançado, como mostrado na Figura 7 da seção 4.6.2. A Tabela 13 mostra uma comparação entre as duas rodadas e o conhecimento em automação de teste dos usuários, com cada célula representando o número de pessoas que possuem o conhecimento em cada nível.

Tabela 13 - Conhecimento em testes automáticos dos usuários para a aplicação Triângulo

Nível de Conhecimento	**Rodada 1**			**Rodada 2**		
	Geral	JUnit	Selenium	Geral	JUnit	Selenium
# Nenhum	3	3	3	1	1	1
# Básico	2	2	2	1	1	1
# Intermediário	0	0	0	2	1	1
# Avançado	0	0	0	0	1	1

A melhora dos resultados de cobertura entre as duas rodadas pode ser atribuída a dois fatores: (i) a diferença entre o conhecimento dos usuários, pois na segunda rodada os usuários tinham maior conhecimento em automação, apesar do vencedor ter sido o único a indicar que não possuía conhecimento em automação, JUnit e Selenium, o que pode derrubar este argumento; (ii) a diferença de pontuação para a criação de caso de teste combinado com a limitação do número de casos de testes que podem ser criados, ambos incentivando os usuário a avaliar (ao invés de cadastrar), forçando a observação dos testes criados, resultando em casos de testes diferentes.

Ao término de cada rodada, o formulário de satisfação (ver Apêndice 3) foi enviado para cada participante para que eles pudessem opinar sobre a ferramenta. Na primeira rodada, três das cinco pessoas mencionaram a usabilidade como um problema, na seção de opiniões, críticas e sugestões. A facilidade de testar uma aplicação utilizando a ferramenta CAT obteve média 4 (de 1 a 5) e facilidade de entendimento da ferramenta CAT obteve média 3,6 (de 1 a 5). A partir desses dados, reposicionamentos e limpezas de itens nas telas foram efetuados, procurando melhorar a usabilidade da ferramenta. O processo deu resultado, pois na segunda rodada nenhum usuário mencionou a usabilidade como problema, além da média para a facilidade de testar uma aplicação utilizando a ferramenta CAT aumentar para 4,75 (de 1 a 5) e a média da facilidade de entendimento da ferramenta CAT aumentar para 4,5 (de 1 a 5).

Tabela 14 - Comparação da forma absoluta e da forma relativa da importância para a segunda rodada da aplicação Triângulo

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Forma absoluta	0/1	0/1	0/1	0/1	0/1	0/1	0/1	1/1	-	-
Forma relativa	0/1	0/1	0/1	0/1	1/1	-	-	-	-	-

Dos 13 casos de teste cadastrados, 5 não foram avaliados, e os mesmo foram os únicos a apresentar os erros, colocando em cheque a multidão no papel de testador. Apesar disso, as classificações dos casos de teste quanto à importância foi calculada das duas formas (absoluta e relativa), e novamente uma classificação foi feita para cada forma. Dessa vez a relativa saiu-se melhor, pois alguns casos de teste tiveram a importância média relativa menor que zero, ficando atrás dos que não foram avaliados (que automaticamente assumem o valor zero). A forma absoluta precisou de 10 dos 13 casos de teste para encontrar o erro, correspondendo aos 80% melhores casos de teste,

enquanto a forma relativa precisou de 7, correspondendo a 50% dos melhores casos de teste.

5.3.5.Comparativo

A Tabela 15 sumariza os resultados dos testes obtidos com os dois métodos e com as duas ferramentas, sendo pra ferramenta Crawljax considerado a média das 10 rodadas, enquanto para a ferramenta CAT apenas a segunda rodada sendo considerada, pois a primeira apresentou problemas de cache de arquivos JavaScript.

Tabela 15 - Comparativo dos testes na aplicação Triângulo

	Path coverage	**Caixa-preta**	**Crawljax**	**CAT**
Cobertura de sentença (%)	100	100	99,6	100
Cobertura de condição (%)	77	100	99,4	88
Erros encontrados	0/1 (0%)	1/1 (100%)	1/1 (100%)	1/1 (100%)

A tabela mostra que o método *path coverage* obteve o pior resultado, sendo a única a não encontrar o erro. As outras 3 ferramentas tiveram um resultado muito bom, todos encontrando o erro e obtendo uma cobertura de pelo menos 85% em ambas as coberturas.

5.4.Experimentos com TuduList

MARCHETTO & TONELLA (2011) utilizaram a aplicação TuduList em seus experimentos e inseriram algumas falhas, baseadas no histórico de falhas da própria aplicação. Das falhas apresentadas, sete foram selecionadas e inseridas na aplicação para a realização dos testes. As falhas foram escolhidas por não serem sobrepostas, podendo estar presentes ao mesmo tempo na aplicação. Apesar disso, para os testes com a ferramenta Crawljax, apenas uma falha por vez foi adicionada. Após a adição de cada falha, a ferramenta foi utilizada. As falhas foram inseridas em conjunto para serem utilizadas na ferramenta CAT. As sete falhas adicionadas são:

1. Ao adicionar um item no modo avançado funciona corretamente, mas se uma nota for preenchida, um item adicional é inserido à listagem;
2. A funcionalidade de compartilhar listagem adiciona um item vazio à listagem quando o botão de esconder é clicado;
3. A funcionalidade de ordenar funciona, porém apaga os itens marcados como realizados;
4. A funcionalidade de adicionar um item de forma rápida adiciona uma

duplicata;

5. A funcionalidade de apagar item funciona como a edição ao invés de apagar o item;
6. Apagar a listagem atual funciona apenas caso a opção de adicionar listagem for clicada duas vezes;
7. Adicionar item no modo avançado não funciona quando o número de itens é maior que 3.

5.4.1.Crawljax

Para os testes com a ferramenta Crawljax, apenas o *plugin* CrawlOverview foi utilizado. O *plugin* TestCaseGenerator não foi utilizado por dois motivos distintos: o primeiro é que a aplicação utiliza um banco de dados, e os casos de teste gerados não são executados na mesma ordem da execução da ferramenta, o que resultaria em um teste diferente; segundo é que o Oráculo compara os códigos fonte em HTML, e os identificadores dos dados no banco são diferentes a cada nova inserção, resultando sempre em erros, pois a ferramenta utiliza os identificadores no código da página.

A ferramenta preenche alguns campos com valores textuais aleatórios - *strings* de 8 caracteres maiúsculos e minúsculos - de acordo com um algoritmo interno que determina quais campos serão preenchidos. Apesar dos resultados dos testes não serem diretamente ligados ao valor preenchido, valores fixos foram utilizados, para manter a igualdade entre os testes realizados, inclusive porque a ferramenta deixava alguns campos em branco.

Os erros foram inseridos um a um e a ferramenta foi utilizada para cada erro. A ferramenta também foi utilizada sem os erros, para que pudesse ser comparada. Para cada uma das situações anteriores a ferramenta foi executada três vezes e o banco de dados teve os dados apagados, exceto dados de usuários. Para cada uma das três vezes o resultado foi exatamente o mesmo. O resultado para a execução da ferramenta na aplicação sem erro, e com cada um dos erros está sumarizado na Tabela 16.

Tabela 16 - Resultado do primeiro uso da ferramenta Crawljax em TuduList

	Sem erro	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Cobertura de sentença (%)	43	43	43	43	39	42	42	43
Cobertura de condição (%)	20	20	20	20	16	20	19	20
# Estados	43	43	55	43	251	50	42	43
# Elementos clicáveis	47	47	59	47	279	57	47	47

Observando a tabela comparativa, apenas pelos números de estado e de elementos clicáveis identificados pela ferramenta é possível observar que as execuções com os erros 2, 4, 5 e 6 apresentam uma diferença. Uma observação mais detalhada foi conduzida comparando o diagrama sem erro e o diagrama da execução com cada erro, onde cada estado foi comparado, verificando as possíveis diferenças. A comparação foi feita de forma visual.

Para o erros 1, 3 e 7, a comparação feita estado a estado não apresentou nenhuma diferença para a execução sem erros. Para o erro 1, a causa é que a ferramenta não preenche nenhuma nota. Para o erro 3, a causa é semelhante, pois a ferramenta não marca nenhum item da listagem como completo. O erro 7, por sua vez, não é encontrado porque a ferramenta busca clicar apenas uma vez em cada elemento.

Para o erro 2, é possível ver uma diferença ao clicar em "Esconder", na tela de compartilhamento de listagem, quando aparecem dois itens ao invés de um na listagem, sendo um deles vazio. Cada item possui a opção de editar e excluir, gerando novos estados.

Para o erro 4, logo no estado inicial, a diferença fica evidente, por apresentar diferenças entre os elementos marcados como clicáveis, ao invés de marcar alguns itens como verde, como o de excluir a listagem atual, marcou-os como amarelo. Continuando a verificação, no estado que mostra a adição de um novo item de forma rápida, a listagem exibe dois itens, ao invés de um, mostrando o erro.

Para o erro 5, o estado que corresponde a exclusão de um item da listagem, ao invés de exibir a listagem vazia, exibe a tela de edição de listagem, mostrando o erro. Para o erro 6, um único estado não está presente, que é o estado após a opção de exclusão da listagem ser acionada, pois nada acontece ao fazê-lo.

Essa comparação visual deveria ser executada pelo Oráculo, porém, como mencionado, o Oráculo compara textualmente os códigos fontes em HTML, resultando em erros, por exemplo, quando o identificador é parâmetro de uma função JavaScript, e os identificadores são gerados automaticamente pelo banco de dados.

Para os erros 1 e 3, como os erros não foram revelados devido a falta de preenchimento de alguns campos, novos testes foram conduzidos, forçando o preenchimento de uma nota, e a marcação dos itens nas listagens. Para o erro 7, nenhuma solução foi encontrada. A ferramenta foi utilizada novamente para esses dois

erros, e também para a aplicação sem erro, e o resultado é exibido na Tabela 17.

Tabela 17 - Resultado do segundo uso da ferramenta Crawljax em TuduList

	Sem erro	1	3
Cobertura de sentença (%)	43	39	43
Cobertura de condição (%)	20	16	20
# Estados	43	265	43
# Elementos clicáveis	47	296	47

Para o erro 1, os números mostram a grande diferença entre o uso da ferramenta com e sem o erro, porém, para o erro 3 novamente os números não evidenciaram nenhuma diferença, nem a comparação visual dos diagramas.

Para o erro 1, assim como no erro 4, o estado inicial mostra as diferenças através dos elementos marcados como clicáveis, ao invés de marcar alguns itens como verde, como o de excluir a listagem atual, marcou-os como amarelo. Continuando a verificação, no estado que adiciona um item de forma avançada, preenchendo uma nota, a listagem exibe dois itens, ao invés de um, demonstrando a presença do erro.

5.4.2.CAT

A aplicação TuduList possui um banco de dados e normalmente o estado da aplicação (mais especificamente do banco de dados) depois do teste é restaurado, para cada teste executado. Para a ferramenta CAT essa restauração seria um desafio, uma vez que é intrusiva demais. Porém SPRENKLE *et al.* (2005) observou que a repetição dos testes em ordem aleatória sem restaurar o estado inicial resulta em uma maior cobertura e detecção de falhas, portanto, foi decidido que a restauração do estado não seria implantada. Os testes foram realizados apenas na parte interna da aplicação, assim como os testes através da ferramenta Crawljax. A ferramenta CAT possibilita, ao cadastrar o caso de teste base, gravar alguns passos para que o testador não tenha que fazê-lo, como o login. Todos os casos de teste foram cadastrados a partir do mesmo usuário, ou seja, um caso de teste interferindo no outro, possibilitando ainda mais a presença e reconhecimento de erros.

5.4.2.1.Rodada 1

Na primeira rodada, nove pessoas foram envolvidas nos testes, todas remotas em suas casas, com uma duração total de 73 minutos. Das nove pessoas envolvidas, uma possui graduação incompleta, duas possuem mestrado completo e seis possuem mestrado incompleto, todos na área de TI. Dos participantes que registraram casos de

teste, três combinações de sistema operacional e navegador foram utilizados: Windows 8 e Chrome; Windows 7 e Firefox; e Mac OS e Safari.

A duração prevista era de apenas 30 minutos, porém assim que a rodada se iniciou houve um problema de energia no servidor, causando a queda do mesmo. Essa queda durou em torno de 15 minutos, até que o servidor pudesse ser novamente colocado no ar. Apesar disso, o servidor ainda se mostrou muito instável no início dos testes, fazendo com que os testadores não conseguissem realizar suas ações. Além disso, dois novos erros foram identificados na plataforma CAT, devido a novas funcionalidades, e.g., inserção de uma descrição ao término do cadastro do caso de teste. A pontuação permaneceu a mesma da rodada 2 da ferramenta Triângulo.

Apesar de todos esses problemas, a rodada prosseguiu, e obteve um resultado melhor que o esperado. Alguns testadores não conseguiram cadastrar casos de teste, e alguns casos de teste foram gravados com erros, por conta da instabilidade. Ainda assim, 10 casos de teste foram cadastrados, com 4 deles sem nenhum evento, ou seja, apenas 6 casos de testes utilizáveis foram cadastrados, um número baixo, com três usuários distintos cadastrando dois cada. 34 avaliações de resultado esperado foram efetuadas e 30 de importância. A Tabela 18 sumariza o resultado e a classificação dos usuários para a rodada.

Tabela 18 - Classificação dos usuários e testes no primeiro experimento da aplicação TuduList com a ferramenta CAT

Usuário	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
# Casos de testes criados	0	2	0	2	0	0	2	0	0	6
# Casos de teste avaliados (resultado esperado)	6	4	6	4	4	5	3	1	1	34
# Casos de teste avaliados com a maioria (resultado esperado)	5	2	4	4	3	4	2	1	0	25
# Casos de teste avaliados (importância)	6	4	4	4	2	5	3	1	1	30
# Casos de teste avaliados com a maioria (importância)	4	4	2	1	2	1	2	1	0	17
Placar	**18**	**12,4**	**12**	**10,4**	**10**	**10**	**8,4**	**4**	**0**	

Uma observação que pode ser feita é que os usuários que não cadastram caso de teste tem vantagem quando há poucos testes, uma vez que não se pode julgar o próprio caso de teste. Dessa forma, para a próxima rodada, a pontuação pela criação de um caso de teste será a importância média do caso de teste. Caso não haja pontuação média, receberá 0,2 pontos.

Dos sete erros inseridos, quatro (erros 4, 5, 6 e 7) foram identificados pelos testadores. A cobertura de sentença é de 57%, enquanto a cobertura de condição foi de 52%. Esses erros, porém foram reconhecidos e explicitados pelos usuários no formulário enviado ao final. Ao reproduzir os casos de teste, nenhum erro pode ser reconhecido, uma vez que a maioria dos casos de teste não foi gravada devido aos erros.

5.4.2.2.Rodada 2

A segunda rodada de testes da aplicação TuduList contou com sete pessoas de TI, com duas pessoas cursando graduação e cinco pessoas cursando o mestrado. A sessão durou 20 minutos, cada pessoa acessando de seus computadores pessoais, utilizando quatro combinações diferentes de sistema operacional e navegador: Windows Vista e Chrome, Windows 7 e Chrome, Windows 8 e Chrome e Ubuntu e Chrome. Nessa rodada, nenhum erro foi relatado sobre a ferramenta CAT, assim como nenhuma instabilidade no servidor.

Foram criados 27 casos de teste ao todo, e a pontuação atribuída a cada um deles foi a respectiva média das avaliações de importância. A Tabela 19 sumariza os dados da rodada, com a pontuação de cada usuário e as avaliações. A consideração da média da avaliação para o valor do caso de teste conseguiu deixar mais justa a pontuação geral, balanceando entre as avaliações e as criações de caso de teste.

Tabela 19 - Classificação dos usuários e testes no segundo experimento da aplicação TuduList com a ferramenta CAT

Usuário	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Total**
# Casos de testes criados	8	5	2	5	2	3	2	27
Pontos pelos casos de teste	19,6	19,42	5	19,83	5,33	7,2	4,2	80,58
# Casos de teste avaliados (resultado esperado)	9	8	13	8	13	7	6	64
# Casos de teste avaliados com a maioria (resultado esperado)	6	5	6	5	7	3	3	35
# Casos de teste avaliados (importância)	9	7	13	8	13	5	6	61
# Casos de teste avaliados com a maioria (importância)	4	5	8	1	5	3	0	26
Placar	**39,6**	**39,42**	**33**	**31,83**	**29,33**	**19,2**	**10,2**	

A cobertura de sentença da rodada foi de 52%, enquanto a cobertura de condição obteve o valor de 35%. Todos os erros inseridos foram encontrados, e todos eles puderam ser vistos ao reproduzir os casos de teste. Alguns casos de teste

explicitaram até 4 erros, explicitando uma possível tendência da multidão em criar casos de teste mais complexos, com muitos passos, atuando como testes exploratórios, uma vez que o escopo do caso de teste base foi bastante abrangente.

Da mesma forma que na aplicação Triângulo, os casos de teste foram classificados de acordo com a sua importância, tanto da forma absoluta, considerando a média pura, quanto da forma relativa, considerando a média calculada levando em conta as médias dos testadores. A Tabela 20 mostra quantos erros foram encontrados considerando apenas a porcentagem dos mais bem classificados para cada uma dessas formas. A evidência de uma diferença não é forte, até por motivos de existir um número baixo de casos de teste.

Tabela 20 - Comparação da forma absoluta e da forma relativa da importância para a segunda rodada da aplicação Triângulo

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Forma ingênua	2/7	4/7	5/7	7/7	-	-	-	-	-	-
Forma relativa	2/7	3/7	6/7	7/7	-	-	-	-	-	-

Um fato que ficou evidente para aplicação Triângulo é que sempre que as pessoas avaliaram um teste como “não passou”, esse teste apontava o erro que existia na aplicação. O mesmo não acontece com a aplicação TuduList. Isso foi notado na primeira rodada, por isso o formulário final enviado aos usuários foi modificado para a segunda rodada. Uma nova pergunta foi acrescentada (área destacada em tracejado no Apêndice 3): “Quando você julgava o resultado esperado, o que levava em consideração?”. As respostas eram: (i) O roteiro do caso de teste base, (ii) Sua percepção individual, (iii) Descrição do caso de teste e (iv) Outro. Das sete respostas, cinco indicaram a descrição do caso de teste, que foi inserida apenas para a aplicação TuduList. Ou seja, o testador criava um caso de teste e, ao terminar, descrevia o caso de teste como: “Ordenação apaga itens”, “Excluir tarefa acaba editando”, “Itens são duplicados na criação de forma rápida” e “Lista não pode ser apagada”. Como a descrição do caso de teste era o erro, os outros testadores avaliavam como correto, pois o resultado esperado por eles era exatamente o erro.

5.4.3.Comparativo

Para Crawljax, os valores foram tirados da média para cada erro. Para as execuções dos erros 1 e 3, foram consideradas as coberturas do segundo experimento.

Da mesma forma, como a primeira rodada da ferramenta CAT foi falha, por motivos de erro de codificação e de instabilidade do sistema, apenas a segunda rodada foi contabilizada.

A Tabela 21 mostra a comparação entre as duas ferramentas. A aplicação CAT obteve um resultado melhor em todas as formas de comparação, achando todos os erros, e tendo ambas as coberturas com mais de 10% de diferença.

Tabela 21 - Comparativo dos testes na aplicação TuduList

	Crawljax	**CAT**
Cobertura de sentença (%)	41,6	52
Cobertura de condição (%)	18,7	35
# Erros encontrados	5/7 (71,4%)	7/7(100%)
Erros encontrados	1, 2, 4, 5 e 6	Todos

5.5.Experimentos com The Organizer

MARCHETTO *et al.* (2008) utilizou a aplicação The Organizer para testes, e listou as falhas que inseriu, de diferentes tipos, como idioma, funcionalidade e navegação (link inválido). Apenas a parte interna do sistema foi avaliada, ou seja, para ambas as ferramentas, as funcionalidades de login, logoff e de registro de novos usuários não foram testadas. Por isso, apenas 5 erros dos 9 explicitados pelo autor foram utilizados nos testes. São eles:

1. Quando não houver nenhuma anotação, a mensagem de listagem vazia estará em outro idioma. O idioma da aplicação é o inglês e a mensagem “errada” está em alemão;
2. Ao adicionar uma nova anotação, armazena o texto de forma invertida;
3. O link para a tela “*My account*” está quebrado, redirecionando para uma página inválida;
4. A lista de tarefas vem com codificação errada, não apresentando caracteres como ‘ç’ ou ‘ã’;
5. O formulário de cadastro de contatos não armazena o campo referente ao nome do meio do contato.

5.5.1.Crawljax

Semelhante à forma como a ferramenta foi utilizada para testar a aplicação TuduList, apenas o *plugin* CrawlOverview foi utilizado. Um dos motivos é o mesmo

para a aplicação TuduList: os casos de teste gerados não são executados na mesma ordem da execução da ferramenta, o que resultaria em um teste diferente, pois os dados não são apagados do banco de dados. O outro motivo é que a nova execução de um caso de teste gerado não tem o mesmo comportamento da execução original, esbarrando em um problema Ajax, como se a tela não conseguisse ser carregada. Isso torna o conteúdo a ser comparado diferente, impossibilitando o uso.

A ferramenta preenche alguns campos com valores textuais aleatórios - de 8 caracteres maiúsculos e minúsculos - de acordo com um algoritmo interno que determina quais campos serão preenchidos. Outros campos para seleção de opções também são preenchidos aleatoriamente. Por serem muitos campos distintos, os valores aleatórios gerados foram utilizados sem modificação.

Os erros foram inseridos um a um, e para cada erro a ferramenta foi utilizada. A ferramenta também foi utilizada sem os erros, para que pudesse ser comparada. A ferramenta foi executada cinco vezes para cada erro e o banco de dados teve os dados apagados, exceto dados de usuários. Para cada erro inserido, as rodadas executadas obtiveram o mesmo resultado, gerando o mesmo diagrama de estados, apesar das diferenças dos dados inseridos. O resultado para a execução da ferramenta na aplicação sem erro, e com cada um dos erros está sumarizado na Tabela 22.

Tabela 22 - Resultado do primeiro uso da ferramenta Crawljax na aplicação The Organizer

	Sem erro	**1**	**2**	**3**	**4**	**5**
Cobertura de sentença (%)	72	72	73	69	72	72
Cobertura de condição (%)	47	47	49	43	48	47
# Estados	46	46	46	44	46	46
# Elementos clicáveis	57	57	57	54	57	57

Observando a tabela comparativa, apenas pelos números de estado e de elementos clicáveis identificados pela ferramenta é possível observar que o erro 3 apresentou uma diferença. Apesar de a cobertura ser diferente, não deve ser considerada como indício de diferença, por não ser parte integrante da ferramenta.

Uma observação mais detalhada foi conduzida comparando o diagrama sem erro e o diagrama da execução com cada erro, onde cada estado foi comparado, verificando as possíveis diferenças. A comparação é visual, estado por estado, uma vez que um Oráculo de boa qualidade não é disponibilizado pela ferramenta. A observação visual confirma o erro 3, não conseguindo visualizar a página com o link

errado.

Além do erro 3, o erro 1 também pode ser observado nitidamente, ao selecionar a opção de listar tarefas. O erro 5, por sua vez, é identificado no estado correspondente a editar um contato, exibindo na tela o valor nulo ao invés do dado inserido no campo correspondente ao nome do meio.

Para os erros 2 e 4, nenhuma diferença na comparação dos estados com a execução sem erros foi identificada. Para tentar identificar esses erros, novas execuções foram feitas, substituindo os valores aleatórios pelo valor "ABÇ", para verificar se o texto é invertido (pois com valores aleatórios fica inviável) e a codificação é alterada. Para cada erro foram executadas cinco rodadas, cada uma com os dados do banco apagados. A Tabela 23 mostra os resultados dessa execução.

Tabela 23 - Resultado do segundo uso da ferramenta Crawljax na aplicação The Organizer

	Sem erro	**2**	**4**
Cobertura de sentença (%)	69	70	69
Cobertura de condição (%)	43	45	44
# Estados	53	53	53
# Elementos clicáveis	43	43	43

Assim como na primeira rodada, nenhuma diferença numérica em comparação a execução sem erros foi identificada, apenas nas coberturas. Por isso, uma inspeção estado a estado foi feita visualmente, buscando por evidências dos erros. Na comparação, ambos os erros foram identificados. Para o erro 2, o estado correspondente a edição de uma anotação mostrou o texto com conteúdo "ÇBA", ou seja, invertido, exibindo o erro. Para o erro 4, o estado correspondente à listagem de tarefas exibiu o texto "AB??", não imprimindo o caractere 'Ç', por problemas de codificação do texto.

A ferramenta Crawljax conseguiu identificar os cinco erros presentes na aplicação, com a ressalva que essa identificação só foi possível através da comparação visual estado a estado, devido a carência de um Oráculo na ferramenta.

5.5.2.CAT

Para a ferramenta CAT, assim como para a ferramenta Crawljax, os testes foram realizados apenas na parte interna da aplicação, excluindo as funcionalidades de registro de novos usuários. O login na aplicação é feito automaticamente, através

dos passos iniciais gravados juntos ao caso de teste base. Os casos de teste foram cadastrados a partir do mesmo usuário, ou seja, um caso de teste interferindo no outro, possibilitando ainda mais a presença e reconhecimento de erros.

Apenas uma rodada foi feita para a aplicação. A ferramenta teve algumas modificações, decorrentes dos resultados das rodadas nas aplicações anteriores. O tempo para testes foi dividido em duas partes: a primeira, com 15 minutos de duração, possibilitava aos testadores cadastrar casos de teste livremente, sem mais limites do número de casos de testes que poderiam criar, e a segunda, onde apenas a avaliação dos casos de testes dos outros testadores estava disponível, com duração de 25 minutos.

Quatro combinações diferentes de navegador e sistema operacional foram utilizados: Mac OS X e Chrome; Windows 7 e Chrome; Mac OS X e Safari 6; e Windows 8 e Chrome. Sete testadores compuseram o time: dois graduandos e 5 mestrandos, todos na área de TI.

Para os testadores, foram feitas duas novas recomendações: (i) fazer casos de teste simples, ou seja, que não realizem muitos passos na ferramenta, e (ii) não descrever o erro, caso tenha identificado algum, na descrição do caso de teste, como ocorreu anteriormente, a fim de evitar julgamentos baseados na descrição do erro. Além disso, junto à avaliação ao término da execução do caso de teste, foi disponibilizado um campo textual para preenchimento do erro encontrado durante a execução, caso o testador tenha percebido algum.

Ao todo foram criados 70 casos de teste, um número bem maior que nos testes anteriores. Alguns casos de teste não puderam ser gravados, pois algumas vezes a aplicação resultava em um erro de Ajax, impossibilitando a conclusão dos testes, pois interrompia a página. Algumas situações que resultavam nesse erro foram relatadas fora da ferramenta, e são comentadas quando for apropriado.

A recompensa oferecida, assim como nas outras rodadas, foi uma barra de chocolate para o ganhador. A Tabela 24 sumariza os dados do experimento, com o número de casos de teste criados por cada usuário, avaliações e placar final.

Tabela 24 - Classificação dos usuários e testes no experimento da aplicação The Organizer com a ferramenta CAT

Usuário	1	2	3	4	5	6	7	Total
# Casos de testes criados	10	17	11	10	9	9	4	70
Pontos pelos casos de teste	30,2	48,42	43,6	33,47	39,33	22,57	18	235,59
# Casos de teste avaliados (resultado esperado)	47	17	23	17	16	15	16	151
# Casos de teste avaliados com a maioria (resultado esperado)	30	14	17	14	9	12	11	107
# Casos de teste avaliados (importância)	46	17	22	16	16	15	16	148
# Casos de teste avaliados com a maioria (importância)	21	14	8	7	7	11	14	82
Placar	**132,2**	**104,42**	**93,6**	**75,47**	**71,33**	**68,57**	**68**	

A cobertura de sentença foi de 81%, enquanto a cobertura de condição foi de 72%. Dos cinco erros inseridos, dois foram encontrados e relatados pelos usuários, no novo campo para descrição: erros 2 e 5. Os erros 1 e 4 foram encontrados ao executar cada caso de teste novamente. O erro 3 foi identificado e relatado externamente a ferramenta, pois o problema do link era Ajax, e impedia a gravação do caso de teste.

Além dos erros inseridos, outros erros foram relatados pelos testadores, através do campo de descrição do erro:

1. Ao cadastrar um compromisso, é permitido o cadastro da hora de término ser anterior à hora de início;
2. Ao salvar ou editar uma tarefa com situação completa e/ou prioridade alta ou baixa, esses dados não são salvos, sendo gravados com os dados padrões (situação incompleta e prioridade normal);
3. E-mail não possui validação;
4. Nome do contato permite apenas números;
5. Telefone não é validado, permitindo apenas letras;
6. Inserção de tarefa com data anterior a data de hoje.

Além desses erros, outro erro foi relatado externamente a ferramenta, por causar um erro Ajax. O erro era lançado ao tentar cadastrar uma anotação com o texto muito grande. Esse erro, juntamente com o erro 2 da listagem anterior são erros de funcionalidade, onde algo realmente não está funcionando. Os outros erros são de validação de campos que podem não ser considerados erros, dependendo da especificação da aplicação, ou serem considerados erros de aceitação, uma vez que o usuário espera que haja uma validação.

Tabela 25 - Comparação da forma absoluta e da forma relativa da importância para a aplicação The Organizer, com os 5 erros inseridos e com os 11 erros totais

Porcentagem	10	20	30	40	50	60	70	80	90	100
Forma absoluta	0/5	1/5	2/5	3/5	3/5	3/5	3/5	3/5	3/5	4/5
Forma relativa	1/5	3/5	3/5	3/5	3/5	3/5	4/5	4/5	4/5	4/5
Forma absoluta (11 erros)	0/11	3/11	4/11	6/11	6/11	7/11	9/11	9/11	9/11	10/11
Forma relativa (11 erros)	2/11	5/11	5/11	5/11	6/11	6/11	7/11	7/11	9/11	10/11

A Tabela 25 mostra a comparação entre a classificação dos casos de teste, quando estes são classificados da forma absoluta e da forma relativa. A tabela conta com duas linhas referentes apenas aos cinco erros inseridos propositalmente e com outras duas linhas para os onze erros (cinco inseridos e os seis listados anteriormente). A tabela mostra que utilizando apenas 30% dos casos de teste (os 30% mais bem qualificados) a forma relativa se saiu melhor, por achar mais erros. Para os erros inseridos propositalmente, a forma relativa também se saiu melhor. Para os erros não semeados, a partir dos 40%, a forma absoluta foi superior.

Acerca do último experimento, algumas observações devem ser relatadas. O pedido de simplicidade foi atendido pelos testadores e casos de teste envolvendo muitas funcionalidades não foram criados. O outro pedido, para não relatar o erro na descrição do caso de teste também foi atendido, não havendo descrições como “Ordenação apaga itens”, como aconteceu no segundo experimento da aplicação TuduList. Além desses, muitos casos de teste com o mesmo objetivo foram criados, apesar de não haver como um testador saber os testes criados por outro na fase de criação de casos de teste.

Dos 70 casos de teste, 22 identificaram erros, sendo 15 dos identificados através da descrição dos erros na ferramenta e 7 através da reprodução pelo responsável pela aplicação. Dos 7 casos de teste, 1 teve o voto da maioria apontando o resultado como não esperado, 2 apontaram o resultado como esperado, 3 não foram avaliados e 1 não teve definição, havendo empate. Dos outros 15 casos de teste, 6 foram avaliados como resultado não esperado, 8 como resultado esperado, e 1 não teve definição.

5.5.3.Comparativo

Para Crawljax, os valores foram tirados da média para cada erro. Para as execuções dos erros 2 e 4, foram consideradas as coberturas do segundo experimento. Para a ferramenta CAT, foi considerado o único experimento realizado.

Tabela 26 - Comparativo dos testes na aplicação The Organizer

	Crawljax	**CAT**
Cobertura de sentença (%)	70,4	81
Cobertura de condição (%)	45,2	72
# Erros encontrados sem comparação visual	1/5 (20%)	2/5 (40%)
# Erros encontrados com comparação visual	5/5 (100%)	5*/5 (100%)
# Erros não inseridos encontrados	0	7*
Erros inseridos encontrados	Todos	Todos

* Contabilizando 1 erro relatado por um meio externo à ferramenta

A Tabela 26 mostra a comparação dos resultados obtidos pelas duas ferramentas. A comparação visual mencionada na tabela para a ferramenta Crawljax é a comparação estado a estado, com as telas lado a lado, pelo autor, verificando alguma diferença visual. Para a ferramenta CAT é a execução do caso de teste pelo responsável pela aplicação, buscando alguma irregularidade, da mesma forma que é feita por outros testadores. O asterisco na tabela significa que um erro contabilizado foi relatado por fora da ferramenta. Da mesma forma que para a ferramenta TuduList, as coberturas foram 10% superiores a ferramenta Crawljax, enquanto os erros foram iguais ou superiores. O diferencial foi a identificação de erros reais pela multidão, o que não aconteceu com a ferramenta Crawljax.

5.6.Perfis e respostas dos testadores

Os testadores preencheram um formulário sobre o nível de conhecimento teórico e prático em testes, ao se cadastrarem na ferramenta, através do formulário presente na Figura 7, onde as respostas são exibidas no Apêndice 4. Esta seção dedica-se a estudar também as respostas fornecidas pelos testadores após cada rodada de experimento, para cada aplicação testada. O formulário em questão encontra-se no Apêndice 3 e as respostas de cada usuário para cada experimento pode ser visualizada no Apêndice 5.

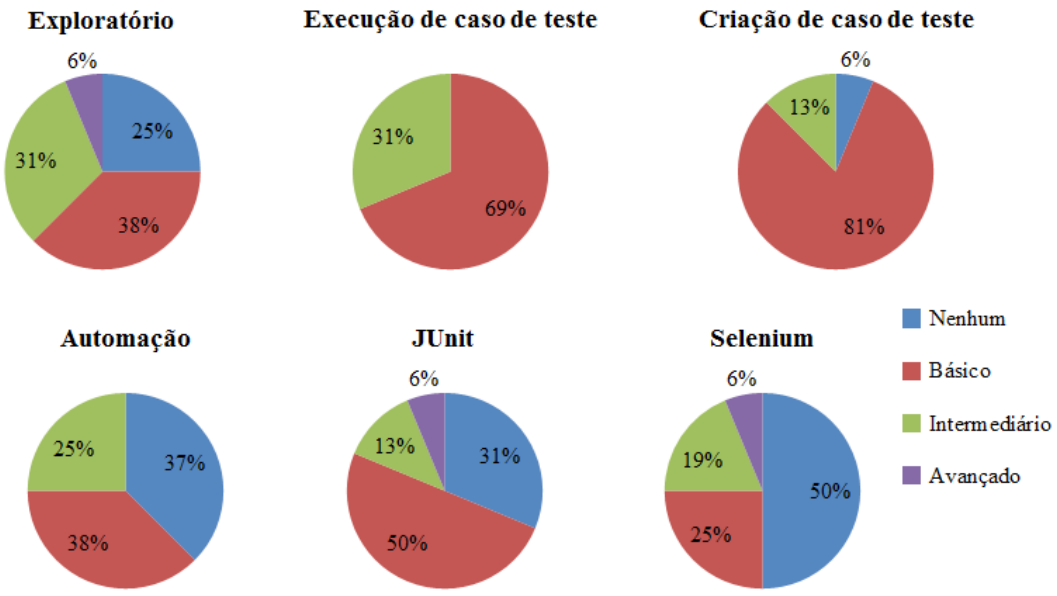

Figura 13 - Distribuição dos participantes quanto ao conhecimento em testes

Ao todo, 16 diferentes usuários participaram das rodadas de experimento. Apenas duas pessoas responderam possuir conhecimento avançado em algum quesito, uma em JUnit e Selenium e outra em testes exploratório. Essa última não era de TI, e talvez a resposta possa estar influenciada pelo não conhecimento do conceito. Cinco pessoas disseram que não possuíam nenhum conhecimento em nenhum campo de automação de testes. A Figura 13 ilustra como está distribuído o conhecimento para cada um dos quesitos.

É possível verificar que todos os participantes possuíam ao menos o conhecimento básico para a execução de caso de teste (manual), apesar de alguns deles não possuírem nenhum conhecimento em automação (2 de 5 com conhecimento intermediário e 3 de 11 com conhecimento básico).

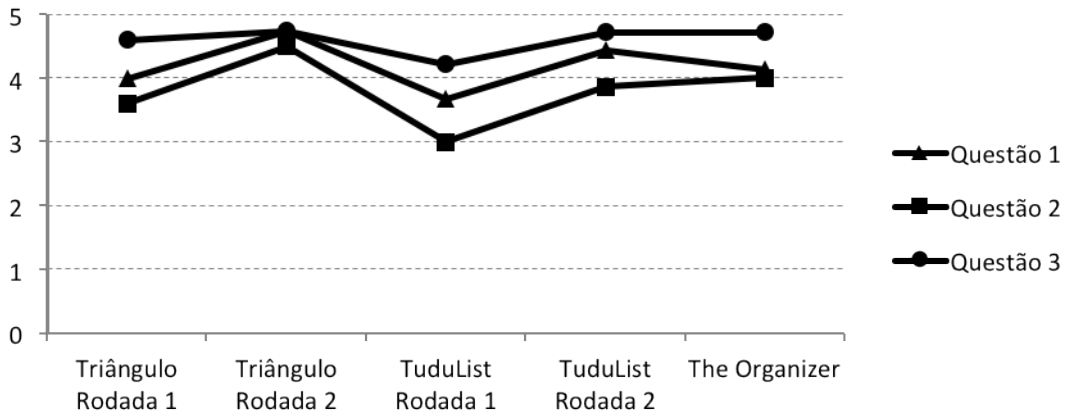

Figura 14 - Média das avaliações dos experimentos com a ferramenta CAT

O gráfico exibido na Figura 14 mostra a satisfação média dos testadores para as rodadas de teste. As questões ilustradas são:

- Questão 1: De 1 a 5, como você avalia a facilidade de testar uma aplicação utilizando a ferramenta CAT?
- Questão 2: De 1 a 5, como você avalia a facilidade de entendimento da ferramenta CAT?
- Questão 3: De 1 a 5, quão útil você julga a ferramenta?

É possível verificar que, na primeira rodada da aplicação TuduList, há uma queda em todas as questões, provavelmente por decorrência dos erros que a ferramenta apresentou devido a instabilidade do ambiente.

Um dos objetivos do trabalho era elaborar uma forma fácil para os testadores da aplicação do processo (objetivo 4 da seção 1.1). Observando as duas primeiras questões, que avaliam justamente esse objetivo, é possível verificar que a média foi em torno de 4, exceto pela primeira rodada da aplicação TuduList (mínima de 4 e máxima de 4,75 para a questão 1 e mínima de 3,6 e máxima de 4,5 para a questão 2). Além disso, o objetivo menciona a necessidade de não instalar nem configurar nada, e a ferramenta necessita apenas que o testador acesse o site e se cadastre. Esses dois pontos demonstram que objetivo foi atingido.

6.Conclusão e Trabalhos Futuros

O trabalho apresentou um processo de teste de software Web que utiliza pessoas externas ao desenvolvimento como fonte de casos de teste, mais precisamente uma multidão desconhecida, uma solução *crowdsourcing* para o problema. Testes de software são responsáveis por até 50% do tempo e mais de 50% do custo (MYERS, 2004), portanto, a solução proposta é uma forma de tornar esse processo mais barato e mais rápido.

O primeiro objetivo desse trabalho é a criação de um processo que utilize a multidão como fonte de criação e avaliação de casos de teste. O proceso criado foi colocado à prova através da ferramenta CAT, onde um conjunto de pessoas externas ao software foi designado para criar casos de teste, além de executar e avaliar casos de teste criados por outros.

A criação dos casos de teste foi bem sucedida, com a ferramenta capturando todas as ações que o testador executou dentro da aplicação, e elogiada pelos usuários. A avaliação foi realizada após outros testadores reproduzirem o caso de teste, julgando o resultado esperado e designando uma nota de 1 a 5 para o teste. Para as relevâncias atribuídas aos testes, não há evidências da forma absoluta ou relativa serem melhor que a outra, pois cada rodada mostrou um resultado diferente. Como trabalho futuro, uma comparação com outras formas de priorização de casos de teste deve ser feita, como a de SAMPATH *et al.* (2008). Em conjunto, outro processo a ser realizado é a redução de casos de teste, uma vez que o número de casos de teste gerados pode ser muito grande, objetivando, por exemplo, atingir a mesma cobertura, como em JONES & HARROLD (2003).

O terceiro objetivo visa descobrir se a multidão é uma boa julgadora de testes, julgando se o resultado do teste é o esperado ou não. A aplicação Triângulo foi bem sucedida na primeira rodada, enquanto na segunda foi inconclusiva, quando os casos de teste com erros não foram avaliados. Para a aplicação TuduList, a primeira rodada também foi inconclusiva, pois os erros atrapalharam a execução. Para a segunda rodada o resultado foi ruim de um ponto de vista, devido às descrições dos casos de teste apontarem os erros, influenciando os testadores executores a avaliarem o resultado como esperado, por identificarem o erro descrito. De outro ponto de vista foi bom, pois pôde-se concluir que a multidão toma como parâmetro para avaliar o

resultado esperado a descrição do caso de teste. Considerando a aplicação The Organizer, para os casos de teste com erros identificados pela multidão, 6 de 15 apontaram erros quando realmente houveram erros, enquanto 8 não apontaram, sendo um resultado razoável. A avaliação do resultado esperado, em geral, foi razoável, variando de acordo com cada situação e aplicação, e nenhuma conclusão final pôde ser alcançada. Apesar disso, aprendeu-se que a multidão considera, majoritariamente, a descrição do caso de teste para avaliar se o resultado obtido é o resultado esperado.

O segundo objetivo é a avaliação do processo utilizando métricas apropriadas, no caso a cobertura (sentença e condição) e a quantidade de erros inseridos encontrados (*fault seeding*). Para esse objetivo a ferramenta CAT foi desenvolvida, obedecendo ao processo proposto. A ferramenta Crawljax foi eleita para comparação, por ser programável e bastante utilizada na literatura, ferramenta essa que gera um diagrama de estados da aplicação, através da navegação. Além dessa, para a aplicação Triângulo, uma comparação com outras duas formas, uma caixa-branca e outra caixa-preta também foi conduzida. A cobertura foi bastante satisfatória no teste com a aplicação Triângulo, perdendo apenas na cobertura de condição para as técnicas caixa-preta e Crawljax, obtendo valor máximo na cobertura de sentença e achando todos os erros, empatando com a técnica caixa-preta. Para TuduList e The Organizer, mais próximas de aplicações reais, saiu-se melhor que Crawljax em todos os critérios: cobertura de sentença, cobertura de condição e o número de erros achados. Na aplicação The Organizer encontrou erros não forçados, ou seja, erros que existiam na aplicação, um resultado animador.

O quarto objetivo era aplicar o processo de uma forma fácil de entender e usar para os usuários. Isso foi atingido de forma extremamente satisfatória, como mostrado na seção 5.6.

Um trabalho futuro bastante interessante é a exportação dos casos de teste gravados pela multidão para scripts de execução automática, por exemplo, o Selenium.

Problemas também foram identificados ao longo do processo, alguns deles técnicos. Uma constatação que pode ser feita acerca da rodada de testes da aplicação The Organizer é que alguns casos de testes repetidos foram gravados pelos testadores, necessitando de uma aplicação de técnicas de redução.

Algumas aplicações utilizam "*alerts*" em JavaScript, que consiste em uma

caixa tipo *pop-up*, com alguma mensagem e um botão de confirmação. Essas caixas não conseguem ter seus eventos capturados pela ferramenta CAT, porém não foi empecilho, pois os testadores que estavam reproduzindo confirmavam a ação.

A ferramenta CAT seria mais completa caso fosse possível restaurar o banco de dados para o estado original após cada cadastro ou execução de caso de teste. Porém, por ser utilizada de forma concomitante por várias pessoas, a restauração se torna uma tarefa extremamente complicada. Uma possível solução seria uma espécie de transação em nível de sessão do usuário, ou seja, ao final de cada cadastro ou execução de caso de teste, um *rollback* seria executado. Além disso, ao utilizarem a ferramenta, todos os testadores estavam autenticados com o mesmo usuário, provocando uma espécie de caos, desejado por um lado, por aumentar as chances de novos erros, e indesejado por outro, por afetar os dados de outros testes sendo executados ao mesmo tempo.

A Tabela 2 da seção 3.6.2.6. mostra um comparativo entre ferramentas de testes de software que utilizam a multidão. A ferramenta CAT pode ser caracterizada através dos mesmos critérios, e o resultado é mostrado na Tabela 27, uma extensão da tabela anterior. Analisando o novo comparativo, é possível observar que ela possui uma variedade maior de tipos de teste disponível, sendo a única que gera dados de teste, compara resultados e realiza *Playback*, além da reputação como forma de motivação. Porém, apenas aplicações Web podem ser testadas.

Tabela 27 - Características das ferramentas de teste de software com *crowdsourcing*, incluindo a ferramenta CAT

Ferramenta	Tipos de aplicação testáveis	Tipos de teste disponível	Tipo da ferramenta	Motivação da multidão	Quem pode testar
Call-For-Testing	Web	Aceitação	*Recorder* e *Bug Tracker*	-	-
CrowdStudy	Web	Usabilidade	*Recorder*	-Implícito (usuários da aplicação) e dinheiro (usuários do MTurk)	Usuários da aplicação e usuários do MTurk
CrowdTest	-Tradicional -Móvel -Web	-Funcionalidade -Segurança -Internacionalização -Interface	*Bug Tracker*	Dinheiro, pagando por bug	Qualquer um
Mob4Hire	Móvel	-Aceitação -Funcionalidade -Usabilidade	Pesquisa	Dinheiro, pagando por cada teste do usuário	Qualquer um
uTest	-Tradicional -Móvel -Web	-Funcionalidade -Carga -Internacionalização -Segurança -Usabilidade	-*Bug Tracker* (funcionalidade, carga, segurança e internacionalização) -Simuladores (carga e segurança) -*Scripts* (funcionalidade, utilizando o Selenium) -Pesquisa (usabilidade)	Dinheiro, pagando por bug	Qualquer um (funcionalidade, internacionalização e usabilidade) Profissionais de teste (carga, segurança, usabilidade e funcionalidade)
CAT	Web	-Funcionalidade -Navegabilidade -Compatibilidade -Internacionalização -Conteúdo -Usabilidade	-Gerador de dados de teste -*Record/Playback* -Comparador de resultados -*Bug Tracker*	Recompensa para o(s) melhor(es), reputação e diversão através da competição.	Qualquer um

Referências Bibliográficas

ABRAN, A., BOURQUE, P., 2004. "Software Testing". In: *SWEBOK: Guide to the software engineering Body of Knowledge*. S.l.: IEEE Computer Society.

VON AHN, L., 2005. *Human computation*. PhD dissertation. Pittsburgh, PA, USA: Carnegie Mellon University.

VON AHN, L., 2006, "Games with a Purpose". In: *Computer*. v. 39, pp. 92–94.

VON AHN, L., 2009. "Human computation". In: *Proceedings of the 46th Annual Design Automation Conference*. New York, NY, USA: ACM. 2009. pp. 418–419.

VON AHN, L., DABBISH, L., 2004. "Labeling images with a computer game". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM. 2004. pp. 319–326.

VON AHN, L., MAURER, B., MCMILLEN, C., *et al.*, 2008, "reCAPTCHA: Human-Based Character Recognition via Web Security Measures". In: *Science*. v. 321, n. 5895 (set.), pp. 1465–1468.

ALSHAHWAN, N., HARMAN, M., 2011. "Automated web application testing using search based software engineering". In: *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society. 2011. pp. 3–12.

ALSHAHWAN, N., HARMAN, M., 2012. "State aware test case regeneration for improving web application test suite coverage and fault detection". In: New York, NY, USA: ACM. 2012. pp. 45–55.

Altruísmo, 1998. Michaelis: moderno dicionário da língua portuguesa. 2 ed. São Paulo, Brasil: Melhoramentos.

AMALFITANO, D., FASOLINO, A.R., POLCARO, A., *et al.*, 2013, "The DynaRIA tool for the comprehension of Ajax web applications by dynamic analysis". In: *Innovations in Systems and Software Engineering*. pp. 1–17.

AMALFITANO, D., FASOLINO, A.R., TRAMONTANA, P., 2010. "Rich Internet Application Testing Using Execution Trace Data". In: *2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*. S.l.: s.n. 2010. pp. 274–283.

AMAZON MECHANICAL TURK, 2013. https://www.mturk.com/mturk/.

ANDREWS, J.H., BRIAND, L.C., LABICHE, Y., *et al.*, 2006, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria". In: *IEEE Transactions on Software Engineering*. v. 32, n. 8, pp. 608–624.

APP STORE, 2012. https://itunes.apple.com/br/genre/ios/id36?mt=8.

ARTZI, S., DOLBY, J., JENSEN, S.H., *et al.*, 2011. "A framework for automated testing of javascript web applications". In: *2011 33rd International Conference on Software Engineering (ICSE)*. S.l.: s.n. 2011. pp. 571–580.

BACH, J., 2001, "What is Exploratory Testing? And How it differs from Scripted Testing". In: *StickyMinds*.

BACH, J., 2002. "Exploratory Testing Explained". In: *The Testing Practitioner*. Den Bosch: UTN Publishers. pp. 209–221.

BASTOS, A., RIOS, E., CRISTALLI, R., *et al.*, 2007, *Base de conhecimento em teste de software*. 2 ed. São Paulo, Martins Fontes.

BEIZER, B., 1990, *Software testing techniques*. New York, NY, USA, Van Nostrand Reinhold Co.

BIOLCHINI, J.C. DE A., MIAN, P.G., NATALI, A.C.C., *et al.*, 2007, "Scientific research ontology to support systematic review in software engineering". In: *Adv. Eng. Inform.* v. 21, n. 2 (abr.), pp. 133–151.

BRABHAM, D.C., 2007. "Speaker's Corner: Diversity in the Crowd". In: *Crowdsourcing*. Disponível em: <http://crowdsourcing.typepad.com/cs/2007/04/speakers_corner.html>.

BRABHAM, D.C., 2008, "Crowdsourcing as a Model for Problem Solving: An introduction and cases". In: *Convergence: The International Journal of Research into New Media Technologies*. v. 14, n. 1, pp. 75–90.

LE BRETON, G., MARONNAUD, F., HALLÉ, S., 2013. "Automated exploration and analysis of ajax web applications with WebMole". In: Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee. 2013. pp. 245–248.

BUHRMESTER, M., KWANG, T., GOSLING, S.D., 2011, "Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data?". In: *Perspectives on Psychological Science*. v. 6, n. 1, pp. 3–5.

BURNS, D., 2012, *Selenium 2 Testing Tools: Beginner's Guide*. S.l., Packt Publishing.

CHESBROUGH, H., 2006. "Open Innovation: A New Paradigm for Understanding Industrial Innovation". In: CHESBROUGH, Henry, VANHAVERBEKE, Wim & WEST, Joel (orgs.), *Open Innovation: Researching a New Paradigm*. New York: Oxford University Press Inc. pp. 1–12.

COHN, J.P., 2008, "Citizen Science: Can Volunteers Do Real Research?". In: *BioScience*. v. 58, n. 3 (mar.), pp. 192–197.

COOK, G., 2011. "How crowdsourcing is changing science". In: *The Boston Globe*. Disponível em: <http://www.bostonglobe.com/ideas/2011/11/11/how-crowdsourcing-changing-science/dWL4DGWMq2YonHKC8uOXZN/story.html>. Acessado em: 17

Agosto 2012.

COPELAND, L., 2004, *A Practitioner's Guide to Software Test Design*. S.l., Artech House.

CROWDTEST, 2012. http://crowdtest.me/.

DIJKSTRA, E.W., 1970. 288: *Concern for Correctness as a Guiding Principle for Program Composition.* Eindhoven, the Netherlands. Technological University. Acessado em: 17 Novembro 2012. Disponível em: <http://www.cs.utexas.edu/~EWD/transcriptions/EWD02xx/EWD288.html>.

DOLINER, M., 2013. Disponível em: <http://cobertura.sourceforge.net/>. Acessado em: 13 Maio 2013.

DUBOIS, J., 2013. TUDU LISTS. Disponível em: <http://www.julien-dubois.com/tudu-lists>. Acessado em: 12 Maio 2013.

DUOLINGO, 2013. http://duolingo.com/.

DYBA, T., DINGSOYR, T., HANSSEN, G.K., 2007. "Applying Systematic Reviews to Diverse Study Types: An Experience Report". In: *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007*. S.l.: s.n. Setembro 2007. pp. 225 –234.

EAGLE, N., 2009. "txteagle: Mobile Crowdsourcing". In: Berlin, Heidelberg: Springer-Verlag. 2009. pp. 447–456.

EATON, C., MEMON, A.M., 2009. "Advances in Web Testing". In: MARVIN V. ZELKOWITZ (org.), *Advances in Computers*. S.l.: Elsevier. pp. 281–306.

ECLIPSE METRICS, 2010. http://metrics2.sourceforge.net/.

ELBAUM, S., KARRE, S., ROTHERMEL, G., 2003. "Improving web application testing with user session data". In: *Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society. 2003. pp. 49–59.

ELBAUM, S., ROTHERMEL, G., KARRE, S., *et al.*, 2005, "Leveraging user-session data to support Web application testing". In: *IEEE Transactions on Software Engineering*. v. 31, n. 3 (mar.), pp. 187–202.

FORTSON, L., MASTERS, K., NICHOL, R., *et al.*, 2011, "Galaxy Zoo: Morphological Classification and Citizen Science". In: *ArXiv eprints*. n. Sandage 1961, pp. 11.

GALAXYZOO, 2012. http://www.galaxyzoo.org/.

GAROUSI, V., MESBAH, A., BETIN-CAN, A., *et al.*, 2013a, "A Systematic Mapping Study of Web Application Testing". In: *Information and Software Technology*.

GAROUSI, V., MESBAH, A., BETIN-CAN, A., *et al.*, 2013b. Disponível em: <http://www.softqual.ucalgary.ca/projects/Web_Application_Testing_Repository/>. Acessado em: 11 Maio 2013.

GEIGER, D., SEEDORF, S., SCHULZE, T., *et al.*, 2011. "Managing the Crowd: Towards a Taxonomy of Crowdsourcing Processes". In: *AMCIS*. S.l.: s.n. 2011.

GIRÃO, A.F., RODRIGUES, S.A., ROCHA, E., *et al.*, 2012. "M-learning System for Learning how to Prepare a Negotiation". In: *eLmL 2012, The Fourth International Conference on Mobile, Hybrid, and On-line Learning*. S.l.: s.n. 30 Janeiro 2012. pp. 51–56.

GOMES, C., SCHNEIDER, D., MORAES, K., *et al.*, 2012. "Crowdsourcing for music: Survey and taxonomy". In: *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. S.l.: s.n. 2012. pp. 832–839.

GOOGLE PLAY, 2012. https://play.google.com/store.

HAJIABADI, H., KAHANI, M., 2011. "An automated model based approach to test web application using ontology". In: *2011 IEEE Conference on Open Systems (ICOS)*. S.l.: s.n. Setembro 2011. pp. 348–353.

HETZEL, B., 1993, *The Complete Guide to Software Testing*. 2. S.l., Wiley.

HOWE, J., 2006a. "Crowdsourcing: A Definition". In: *Crowdsourcing*. Disponível em: <http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html>.

HOWE, J., 2006b, "The Rise of Crowdsourcing". In: *Wired Magazine*.

HOWE, J., 2009, *O Poder das Multidões: Por que a força da coletividade está remodelando o futuro dos negócios.* Tradução por: Alessandra Mussi Araujo. 2 ed. Rio de Janeiro, Elsevier.

INNOCENTIVE, 2012. http://www.innocentive.com/.

ISTOCKPHOTO, 2012. http://www.istockphoto.com/.

ITKONEN, J., MANTYLA, M.V., LASSENIUS, C., 2007. "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing". In: *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007*. S.l.: s.n. Setembro 2007. pp. 61 –70.

JIA, Y., HARMAN, M., 2011, "An Analysis and Survey of the Development of Mutation Testing". In: *IEEE Transactions on Software Engineering*. v. 37, n. 5, pp. 649–678.

JONES, J.A., HARROLD, M.J., 2003, "Test-suite reduction and prioritization for modified condition/decision coverage". In: *IEEE Transactions on Software Engineering*. v. 29, n. 3, pp. 195–209.

JORGENSEN, P., 2002, *Software testing: a craftsman's approach*. 2nd ed. Boca

Raton, Fla, CRC Press.

JUNIT, 2012. https://github.com/kentbeck/junit/wiki.

KANER, C., 2002. "Testing Techniques". In: *Lessons Learned in Software Testing: A Context-Driven Approach*. New York: Wiley. pp. 31–63.

KANER, C., 2008. "A tutorial in exploratory testing". In: *QAI QUEST Conference*. Chicago. Abril 2008. Disponível em: <http://www.kaner.com/pdfs/QAIExploring.pdf>.

KANSKAR, A., SALUJA, N., 2012, "Efficient Agent Based Testing Framework for Web Applications". In: *International Journal of Scientific & Engineering Research*. v. 3, n. 2 (fev.).

KICKSTARTER, 2012. http://www.kickstarter.com/.

KITCHENHAM, B., CHARTERS, S., 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. S.l. Acessado em: 7 Julho 2013. Disponível em: <http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf>.

KITTUR, A., SMUS, B., KHAMKAR, S., *et al.*, 2011. "CrowdForge: crowdsourcing complex work". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. New York, NY, USA: ACM. 2011. pp. 43–52.

LAM, W., 2001, "Testing e-commerce systems: a practical guide". In: *IT Professional*. v. 3, n. 2 (abr.), pp. 19–27.

LÉVY, P., 1998, *A inteligência coletiva: por uma antropologia do ciberespaço*. Tradução por: Luiz Paulo Rouanet. 5 ed. São Paulo, Brasil, Edições Loyola.

DI LUCCA, G.A., FASOLINO, A.R., 2006a, "Testing Web-based applications: The state of the art and future trends". In: *Information and Software Technology*. v. 48, n. 12 (dez.), pp. 1172–1186.

DI LUCCA, G.A., FASOLINO, A.R., 2006b. "Web Application Testing". In: MENDES, Emilia & MOSLEY, Nile (orgs.), *Web Engineering*. S.l.: Springer Berlin Heidelberg. pp. 219–260.

MARCH, J.G., 1991, "Exploration and Exploitation in Organizational Learning". In: *Organization Science*. v. 2, n. 1 (fev.), pp. 71–87.

MARCHETTO, A., RICCA, F., TONELLA, P., 2008, "A case study-based comparison of web testing techniques applied to AJAX web applications". In: *International Journal on Software Tools for Technology Transfer*. v. 10, n. 6 (dez.), pp. 477–492.

MARCHETTO, A., TONELLA, P., 2011, "Using search-based algorithms for Ajax event sequence generation during testing". In: *Empirical Software Engineering*. v. 16, n. 1 (fev.), pp. 103–140.

MCMASTER, S., YUAN, X., 2012. "Developing a Feedback-Driven Automated Testing Tool for Web Applications". In: *2012 12th International Conference on Quality Software (QSIC)*. S.l.: s.n. 2012. pp. 210–213.

MESBAH, A., VAN DEURSEN, A., 2009. "Invariant-based automatic testing of AJAX user interfaces". In: *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society. 2009. pp. 210–220.

MESBAH, A., VAN DEURSEN, A., LENSELINK, S., 2012a, "Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes". In: *ACM Trans. Web*. v. 6, n. 1 (mar.), pp. 3:1–3:30.

MESBAH, A., VAN DEURSEN, A., ROEST, D., 2012b, "Invariant-Based Automatic Testing of Modern Web Applications". In: *Software Engineering, IEEE Transactions on*. v. 38, n. 1, pp. 35–53.

MILLS, D.L., 2008. "Testing for Web Applications". In: BRANDON, D.M., *Software Engineering for Modern Web Applications: Methodologies and Technologies*. S.l.: Igi Global. Premier Reference Source. pp. 207–216.

MIRSHOKRAIE, S., MESBAH, A., 2012. "JSART: JavaScript Assertion-Based Regression Testing". In: BRAMBILLA, Marco, TOKUDA, Takehiro & TOLKSDORF, Robert (orgs.), *Web Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. pp. 238–252.

MIRSHOKRAIE, S., MESBAH, A., PATTABIRAMAN, K., 2013. "Efficient JavaScript Mutation Testing". In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)*. S.l.: IEEE Computer Society. 2013.

MOB4HIRE, 2013. http://www.mob4hire.com/.

MYERS, G.J., 2004, *The Art of Software Testing*. S.l., Wiley. Business Data Processing: A Wiley Series.

NAIK, K., TRIPATHY, P., 2008, *Software testing and quality assurance: theory and practice*. Hoboken, N.J, John Wiley & Sons.

NEAR, J.P., JACKSON, D., 2012. "Rubicon: bounded verification of web applications". In: New York, NY, USA: ACM. 2012. pp. 60:1–60:11.

NEBELING, M., SPEICHER, M., GROSSNIKLAUS, M., *et al.*, 2012. "Crowdsourced web site evaluation with crowdstudy". In: *Proceedings of the 12th international conference on Web Engineering*. Berlin, Heidelberg: Springer-Verlag. 2012. pp. 494–497.

NGUYEN, H.Q., 2003, *Testing applications on the Web: test planning for mobile and Internet-based systems*. 2nd ed. Indianapolis, Ind, Wiley Pub.

OCARIZA, F.S., PATTABIRAMAN, K., MESBAH, A., 2012. "AutoFLox: An

Automatic Fault Localizer for Client-Side JavaScript". In: Washington, DC, USA: IEEE Computer Society. 2012. pp. 31–40.

ORDANINI, A., MICELI, L., PIZZETTI, M., *et al.*, 2011, "Crowd-funding: transforming customers into investors through innovative service platforms". In: *Journal of Service Management*. v. 22, n. 4 (set.), pp. 443–470.

PAGE, S.E., 2007. "Diversity and Problem Solving: Darwin's Brass Tacks". In: *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Society*. S.l.: Princeton University Press. pp. 131–174.

PERENS, B., 1999. "The Open Source Definition". In: DIBONA, Chris, OCKMAN, Sam & STONE, Mark (orgs.), *Open Sources: Voices from the Open Source Revolution*. 1. S.l.: O'Reilly, Sebastopol, Calif. pp. 171–188.

PFLEEGER, S.L., ATLEE, J.M., 2009, *Software Engineering: Theory and Practice*. 4. S.l., Prentice Hall.

PRAKASH, V., GOPALAKRISHNAN, S., 2011. "Testing efficiency exploited: Scripted versus exploratory testing". In: *2011 3rd International Conference on Electronics Computer Technology (ICECT)*. S.l.: s.n. Abril 2011. pp. 168 –172.

PRESSMAN, R.S., 2010, *Software engineering: a practitioner's approach*. 7. S.l., McGraw-Hill Higher Education. McGraw-Hill higher education.

PRESSMAN, R.S., LOWE, D., 2008. "Testing WebApps". In: *Web engineering: a practitioner's approach*. S.l.: McGraw-Hill Higher Education. pp. 359–396.

QUINN, A.J., BEDERSON, B.B., 2009. *A Taxonomy of Distributed Human Computation*. S.l. University of Maryland.

QUINN, A.J., BEDERSON, B.B., 2011. "Human computation: a survey and taxonomy of a growing field". In: *Proceedings of the 2011 annual conference on Human factors in computing systems*. New York, NY, USA: ACM. 2011. pp. 1403–1412.

RAN, L., DYRESON, C., ANDREWS, A., *et al.*, 2009, "Building test cases and oracles to automate the testing of web database applications". In: *Information and Software Technology*. v. 51, n. 2 (fev.), pp. 460–477.

RICCA, F., TONELLA, P., 2001. "Analysis and testing of Web applications". In: *Proceedings of the 23rd International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society. 2001. pp. 25–34.

ROEST, D., MESBAH, A., VAN DEURSEN, A., 2010. "Regression Testing Ajax Applications: Coping with Dynamism". In: *2010 Third International Conference on Software Testing, Verification and Validation (ICST)*. S.l.: s.n. 2010. pp. 127–136.

ROSS, J., IRANI, L., SILBERMAN, M.S., *et al.*, 2010. "Who are the crowdworkers? Shifting demographics in mechanical turk". In: *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*.

New York, NY, USA: ACM. 2010. pp. 2863–2872.

SACKS, M., 2012. "Web Testing Practices". In: *Pro Website Development and Operations*. S.l.: Apress. pp. 27–43.

SAMPATH, S., BRYCE, R.C., VISWANATH, G., *et al.*, 2008. "Prioritizing User-Session-Based Test Cases for Web Applications Testing". In: *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*. Washington, DC, USA: IEEE Computer Society. Abril 2008. pp. 141–150.

SCHENK, E., GUITTARD, C., 2011, "Towards a characterization of crowdsourcing practices". In: *Journal of Innovation Economics*. n. 1, pp. 93–107.

SCHNEIDER, D., SOUZA, J. DE, MORAES, K., 2011, "Multidões: a nova onda do CSCW?". In: *SBSC & CRIWG-VIII Simpósio Brasileiro de Sistemas Colaborativos. Brasil*. pp. 24–31.

SELENIUM, 2013. http://docs.seleniumhq.org/.

SOMMERVILLE, I., 2008. "Teste de software". In: Tradução por: S.S.S. Melnikoff, R. Arakaki, E. De Andrade Barbosa, *Engenharia de software*. S.l.: Pearson Addison-Wesley. pp. 355–373.

SPRENKLE, S., ESQUIVEL, H., HAZELWOOD, B., *et al.*, 2008. "WebVizOr: A Visualization Tool for Applying Automated Oracles and Analyzing Test Results of Web Applications". In: *Practice and Research Techniques, 2008. TAIC PART '08. Testing: Academic Industrial Conference*. S.l.: s.n. 2008. pp. 89–93.

SPRENKLE, S., GIBSON, E., SAMPATH, S., *et al.*, 2005. "Automated replay and failure detection for web applications". In: *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, NY, USA: ACM. 2005. pp. 253–262.

SUROWIECKI, J., 2006, *A Sabedoria da Multidões*. Tradução por: Alexandre Martins. Rio de Janeiro, Record.

TAHCHIEV, P., LEME, F., MASSOL, V., *et al.*, 2010, *JUnit in Action*. 2 ed. Stamford, CT, USA, Manning Publications.

TESTNG, 2013. http://testng.org/.

THREADLESS, 2012. http://www.threadless.com.

Tradução por: BSTQB, 2012, *Glossário Padrão de Termos Utilizados em Teste de Software*. Tradução por: BSTQB. versão 2.2. S.l., s.n. Acessado em: 20 Fevereiro 2013.

UTEST, 2012. http://www.utest.com/.

VUKOVIC, M., 2009. "Crowdsourcing for Enterprises". In: *Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society. Julho

2009. pp. 686–692.

WANG, F.-Y., CARLEY, K.M., ZENG, D., *et al.*, 2007, "Social Computing: From Social Informatics to Social Intelligence". In: *IEEE Intelligent Systems*. v. 22, n. 2, pp. 79–83.

WEBSTER, J., WATSON, R.T., 2002, "Analyzing the Past to Prepare for the Future: Writing a Literature Review". In: *MIS Quarterly*. v. 26, n. 2 (jun.), pp. xiii–xxiii.

WIKIPEDIA, 2013. http://www.wikipedia.org/.

XIONG, W., BAJWA, H., MAURER, F., 2005. "WIT: A Framework for In-container Testing of Web-Portal Applications". In: LOWE, David & GAEDKE, Martin (orgs.), *Web Engineering*. S.l.: Springer Berlin Heidelberg. Lecture Notes in Computer Science, 3579. pp. 87–97.

YOUTUBE, 2013. http://www.youtube.com.

YU, L., ZHAO, W., DI, X., *et al.*, 2009. "Towards Call for Testing: An Application to User Acceptance Testing of Web Applications". In: *2009 Ieee 33rd International Computer Software and Applications Conference, Vols 1 and 2*. S.l.: s.n. pp. 166–171.

YUEN, M.-C., KING, I., LEUNG, K.-S., 2011. "A Survey of Crowdsourcing Systems". In: *Proceedings of the IEEE Third International Conference on Social Computing (SocialCom)*. S.l.: IEEE. Outubro 2011. pp. 766–773.

ZAMMETTI, F.W., 2006, *Practical Ajax projects with Java technology*. Berkeley, CA; New York, Apress ; Distributed to the Book trade by Springer-Verlag. Acessado em: 23 Agosto 2013.

ZHAO, Y., ZHU, Q., 2012, "Evaluation on crowdsourcing research: Current status and future direction". In: *Information Systems Frontiers*. pp. 1–18.

Apêndices

Para a pesquisa de aplicações que envolvessem o uso de *crowdsourcing* nos testes de software, foi realizada uma revisão sistemática de literatura. O Apêndice 1 explica com mais detalhes como o processo foi conduzido.

Na mesma linha de revisão sistemática, o Apêndice 2 apresenta a continuação do mapeamento sistemático apresentado por GAROUSI *et al.* (2013a), abrangendo anos não cobertos pelo estudo original.

O Apêndice 3 traz o formulário enviado aos usuários da ferramenta CAT, a ser preenchidos após a utilização da ferramenta. As respostas do formulário são exibidas no Apêndice 5, para cada experimento. O Apêndice 4 traz o perfil de todos os usuários participantes dos experimentos. Os Apêndice 6 e Apêndice 7 trazem os modelos de dados utilizados pela ferramenta CAT, tanto do módulo inserido dentro da aplicação a ser testada quanto da própria ferramenta.

Apêndice 1 - Revisão Sistemática sobre *Crowdsourcing* e Testes

A pesquisa por trabalhos relacionados foi elaborada utilizando a revisão sistemática de literatura. A escolha deste método foi devido a seu rigor na busca, por facilitar a identificação de lacunas em determinada área e fornecer insumos para posicionamento de novas pesquisas (KITCHENHAM & CHARTERS, 2007). O interesse desta forma de busca para o presente trabalho era encontrar trabalhos acadêmicos que possuíssem relação com *crowdsourcing* e ao mesmo tempo com testes de software.

A revisão sistemática da literatura é uma forma de estudo secundário que pode ser caracterizado como um método rigoroso para identificar, avaliar e sintetizar os estudos em um determinado assunto, visando a construção e um resumo (DYBA *et al.*, 2007). Estudos secundários são aqueles que buscam integrar resultados de estudos primários em um determinado campo de pesquisa, onde estudos primários são conduzidos para caracterizar um objeto em uso (BIOLCHINI *et al.*, 2007).

De acordo com KITCHENHAM & CHARTERS (2007), três fases consistem uma revisão sistemática de literatura:

1. Planejamento, onde os objetivos, as fontes a serem pesquisadas, critérios de inclusão e exclusão, e o protocolo para execução são definidos;
2. Execução, onde os passos definidos no protocolo do passo anterior são efetuados;
3. Relatório, onde os dados dos artigos são extraídos e sintetizados.

Comparando-se os métodos tradicionais de revisão com a revisão sistemática, algumas diferenças podem ser notadas: (i) a questão é geralmente ampla nos métodos tradicionais enquanto na revisão sistemática ela é focada; (ii) a forma avaliação é rígida na revisão sistemática e variável nos outros métodos; (iii) a síntese é qualitativa nos métodos tradicionais, podendo ser qualitativa e/ou quantitativa na revisão sistemática; (iv) a seleção pode ser tendenciosa nos métodos tradicionais, diferente da revisão sistemática, onde um critério é uniformemente aplicado; (v) a revisão sistemática exige um esforço consideravelmente maior (DYBA *et al.*, 2007, KITCHENHAM & CHARTERS, 2007).

Fases da Pesquisa

Para saber qual o estado da pesquisa em relação a aplicação de *crowdsourcing* em teste de software, juntamente com suas lacunas uma revisão sistemática da literatura foi realizada. Esta seção descreve com detalhes as três fases executadas na revisão sistemática.

Fase 1 - Planejamento

O primeiro passo dentro do planejamento foi elaborar uma questão de pesquisa. A questão formulada foi: "É possível realizar testes de software utilizando *crowdsourcing*?". Além disso, questões complementares foram criadas: (i) Quais tipos de teste a multidão é capaz de realizar? (ii) Quais habilidades as pessoas necessitam? (iii) Quais tipos de aplicação elas conseguem testar? (iv) Qual a motivação dos testadores? A resposta para estas questões serão o resultado dessa revisão sistemática.

Para realizar a busca, oito bases foram utilizadas para a pesquisa: ACM Digital Library[29], CiteseerX[30], IEEE[31], Scopus[32], SciELO[33], ScienceDirect[34], Springer[35] and Web of Knowledge[36], sem restrição de ano. Além disso, cinco conferências foram manualmente examinadas, a partir do ano de 2007: WISE (Web Information Systems Engineering), ICWE (International Conference on Web Engineering), ICST (International Conference on Software Testing, Verification and Validation), ICCCI (International Conference on Computational Collective Intelligence) e UBICOMP (International Conference on Ubiquitous Computing).

Apenas estudos em inglês foram considerados, onde os estudos primários deveriam conter iniciativas da utilização da multidão em teste de software, e estudos secundários deveriam citar ferramentas que fazem uso dessa forma de testes.

A *string* de busca utilizada foi composta dos termos "software testing" e "crowdsourcing", e seus sinônimos, resultando na *string*: "(((‘software test’ OR ‘software testing’ OR ‘software evaluation’ OR ‘software quality’ OR ‘software

[29] http://dl.acm.org/
[30] http://citeseer.ist.psu.edu/
[31] http://ieeexplore.ieee.org/
[32] http://www.scopus.com/
[33] http://www.scielo.org/
[34] http://www.sciencedirect.com/
[35] http://link.springer.com/
[36] http://isiwebofknowledge.com/

reliability’ OR ‘program test’ OR ‘program testing’ OR ‘program evaluation’ OR ‘program quality’ OR ‘program reliability’ OR ‘application test’ OR ‘application testing’ OR ‘application evaluation’ OR ‘application quality’ OR ‘application reliability’ OR ‘system test’ OR ‘system testing’ OR ‘system evaluation’ OR ‘system quality’ OR ‘system reliability’ OR ‘web application test’ OR ‘web application testing’ OR ‘web application evaluation’ OR ‘web application quality’ OR ‘web application reliability’ OR ‘quality assurance’ OR ‘call for testing’ OR ‘beta testing’ OR ‘testing model’) AND (‘crowdsourcing’ OR ‘crowdsource’ OR ‘crowd computing’ OR ‘crowd-based’ OR ‘crowding-based’ OR ‘wisdom of crowds’ OR ‘collective intelligence’)) OR ‘community testers’ OR ‘crowdtest’ OR ‘crowdtesting’)”. Os termos entre aspas simples não podem ser desmembrados, ou seja, conter apenas o termo software e *crowdsourcing* não é suficiente para a busca.

Fase 2 - Execução

A execução da busca nos mecanismos foi realizada em 10 de dezembro de 2012, resultando em 497 entradas. Destas, 36 eram repetições, restando 461. O título e o abstract das remanescentes foram analisados, restando 51 estudos a serem lidos. Esta leitura abordou primeiramente a introdução de cada um deles, caso ela deixasse mais claro que o objetivo do trabalho não era compatível com a pesquisa, o mesmo era excluído. Feito isso, o trabalho por completo era lido dos que ainda possuíam afinidade com o tema. Por fim, apenas 2 estudos obedeceram aos critérios de inclusão, um primário e um secundário. A Tabela 28 mostra os dados separados por cada mecanismo de busca.

Tabela 28 - Resultado da busca por artigos da revisão sistemática

	Journal	Conferência	Capítulo	Outro	Total
ACM	9	136	0	64	**209**
CiteseerX	1	4	0	0	**5**
IEEE	3	14	0	1	**18**
ScienceDirect	2	0	0	0	**2**
Scopus	12	31	0	2	**45**
SciELO	0	0	0	0	**0**
Springer	47	80	83	2	**212**
Web of Knowledge	0	6	0	0	**6**
Total	74	271	83	69	**497**
Não repetidos	70	242	83	66	**461**
Selecionado	8	40	2	1	**51**
Incluído	**1**	**1**	**0**	**0**	**2**

A busca manual resultou em mais um trabalho, proveniente do ICWE. Para os três artigos, foram executados os passos *forward* e *backward* (WEBSTER & WATSON, 2002). *Backward* é o passo de recuperar os artigos citados pelos três trabalhos e verificar se algum deles poderia ser incluído. *Forward* é o processo de buscar quem citou os artigos selecionados, e para isso o Google Scholar[37] foi utilizado, devido à funcionalidade "Citado por" existente. Estes passos resultaram em um novo trabalho secundário selecionado, resultando, ao fim, em dois trabalhos primários e dois trabalhos secundários.

É interessante ressaltar que durante a leitura destes 51 trabalhos foi observada a presença de alguns previamente estudados pelo autor, por exemplo, DI LUCCA & FASOLINO (2006a) e ELBAUM *et al.*(2003).

Fase 3 - Relatório

Cada um dos estudos primários possui uma ferramenta de teste de software que utiliza a multidão: CrowdStudy (NEBELING *et al.*, 2012) e Call-For-Testing(CFT) (YU *et al.*, 2009). Este número de ferramentas é bastante pequeno para que uma conclusão sólida seja extraída, por esse motivo outras duas ferramentas foram retiradas dos estudos secundários: uTest (UTEST, 2012) e Mob4Hire (MOB4HIRE, 2013), ambas retiradas de VUKOVIC (2009) e ZHAO & ZHU (2012). Além disso, outra ferramenta chamada Crowdtest (CROWDTEST, 2012) também foi selecionada para comparação, por ser a única brasileira encontrada sobre o assunto.

A partir dos dados extraídos de cada uma das ferramentas de teste, um comparativo foi elaborado. A partir desses dados, é possível responder as questões que levaram à pesquisa. A pergunta principal era: "É possível realizar testes de software utilizando *crowdsourcing*?". A resposta é sim. Especialmente as aplicações uTest e Mob4Hire, com sua grande quantidade de testadores, parcerias e projetos bem sucedidos demostram essa possibilidade. Apesar disso, os testes de multidão não tem o propósito de substituir os testes convencionais, e sim, complementar os testes. As outras questões e respostas são: (i) Quais tipos de teste a multidão é capaz de realizar? Aceitação, funcionalidade, carga, usabilidade, segurança, interface e internacionalização foram as respostas encontradas; (ii) Quais habilidades as pessoas necessitam? Dependendo do teste, apenas o conhecimento da língua é suficiente, outros tipos necessitam de um conhecimento específico; (iii) Quais tipos de aplicação

[37] http:// scholar.google.com

elas conseguem testar? Aplicações tradicionais, móveis e Web; (iv) Qual a motivação dos testadores? Normalmente dinheiro, apesar da ferramenta CrowdStudy utilizar as próprias interações dos usuários da aplicação para testes, uma forma implícita.

Apêndice 2 - Continuação do Mapeamento Sistemático

Mapeamento sistemático é uma forma de conduzir um estudo secundário, da mesma forma que uma revisão sistemática. O processo de ambos é semelhante, possuindo critérios de exclusão e inclusão, definição dos mecanismos e da *string* de busca, além de buscas manuais em conferências importantes da área.

As diferenças entre ambos os métodos está no foco que cada um possui. Revisões sistemáticas (SR) são elaboradas em temas mais específicos, enquanto mapeamentos (SM) são em temas mais abrangentes. Temas para conduzir SRs podem ser extraídos de um SM. O resultado de um SM é geralmente traduzido em gráficos e totalizações, resumindo os dados de forma a responder as questões, enquanto SRs possuem como resultado uma análise descritiva. Por esse motivo, SRs tem mais chances de alcançar um público maior e menos restrito a academia, enquanto SM atingem mais a academia, direcionando futuras pesquisas (KITCHENHAM & CHARTERS, 2007).

O trabalho de GAROUSI *et al.* (2013a) é um mapeamento sistemático sobre testes de funcionalidade em aplicações web, realizado durante o verão europeu de 2011, entre maio e agosto. Neste estudo são explicados os passos conduzidos para a realização do mapeamento sistemático, como os critérios de inclusão e exclusão, *string* e máquinas de busca utilizadas. Utilizando esses mesmos dados, a pesquisa foi reproduzida, em meados de 2013, mais precisamente em 18 de junho, contemplando os trabalhos publicados entre 2011 e 2013. A pesquisa será mais bem detalhada a seguir.

Quatro eram os objetivos do estudo original de GAROUSI *et al.* (2013a): (i) desenvolver um esquema genérico de classificação de estudos na área de teste de aplicação web, (ii) a realização em si do mapeamento sistemático, classificando de acordo com o esquema desenvolvido, (iii) uma análise das tendências demográficas e (iv) disponibilizar a classificação em um repositório online.

O primeiro objetivo não será reproduzido, pois o esquema genérico continuou atendendo aos dados que os estudos demonstraram. Os outros três objetivos foram traçados nessa continuação do SM.

Execução da pesquisa

Para atender os três objetivos, as mesmas etapas do estudo original foram seguidas. As máquinas de busca utilizadas foram o IEEE Xplore[38], ACM Digital Library[39], Google Scholar[40], Microsoft Academic Search[41], CiteSeerX[42] e ScienceDirect[43]. Em cada uma das máquinas, a seguinte *string* foi utilizada: (web OR website OR ‘‘web application’’ OR Ajax OR JavaScript OR HTML OR DOM OR PHP OR J2EE OR Java servlet OR JSP OR.NET OR Ruby OR Python OR Perl OR CGI) AND (test OR testing OR analysis OR analyzing OR ‘‘dynamic analysis’’ OR ‘‘static analysis’’ OR verification).

Além disso, algumas conferências foram verificadas manualmente, por serem as principais em engenharia de software: TSE, ICSE, FSE, ICST, ISSTA, ICWE, WSE, WWW, TWEB e TOIT.

Após a realização das pesquisas, um montante inicial com 79 estudos foi gerado, sobre os quais foram aplicados os critérios de inclusão e exclusão. Estudos que não possuíam como foco os testes de funcionalidade, e.g. acessibilidade, desempenho e segurança foram excluídos. Apenas estudos escritos na língua inglesa e disponíveis eletronicamente foram incluídos. Estudos que tratavam de Web Services também foram excluídos, devido à diferença entre a natureza das aplicações. Aplicando estes critérios, apenas 43 estudos foram selecionados.

Seguindo os esquemas de classificação do estudo original, os estudos foram devidamente julgados, e o resultado pode ser visualizado em um repositório online, disponível em http://www.cos.ufrj.br/~girao/Web_Application_Testing_Repository/.

Dados demográficos

Dos 43 estudos, 3 foram mais citados, de acordo com o Google Scholar. O número de citações está disponível também no repositório, juntamente com o número de citações normalizadas. Essa normalização é definida como o número de citações dividido pela diferença entre os anos de 2014 e o ano da publicação. Os 3 mais

[38] http://ieeexplore.ieee.org
[39] http://dl.acm.org
[40] http://scholar.google.com
[41] http://academic.research.microsoft.com
[42] http://citeseer.ist.psu.edu
[43] http://www.sciencedirect.com

citados foram: (MESBAH *et al.*, 2012a), com 23 citações, normalizadas em 11,5 normalizada; (ALSHAHWAN & HARMAN, 2011), com 22 citações normalizadas em 7,33; e (MESBAH *et al.*, 2012b), com 15 citações normalizadas em 7,5.

O estudo original elencou também quais países foram os que mais publicaram na área, sendo o primeiro lugar dos Estados Unidos, seguidos por Itália e Canadá. Nestes dois últimos anos, os Estados Unidos continuou na frente com 8 publicações, seguido desta vez pela China com 7, e Canadá e Índia empatados em terceiro com 6.

Apêndice 3 - Formulário de pesquisa de satisfação da ferramenta CAT

Figura 15 - Formulário de satisfação da ferramenta CAT

O destaque em tracejado da figura refere-se a uma pergunta adicionada posteriormente, conforme explicado no texto.

Apêndice 4 - Perfil dos usuários da ferramenta CAT

A Tabela 29 ilustra a resposta de cada usuário cadastrado na ferramenta CAT, para o formulário da Figura 7. Apenas dois usuários não possuíam formação pela UFRJ, assinalados com *, e as respostas numéricas equivalem a: (0) Nenhum, (1) Básico, (2) Intermediário e (3) Avançado.

Tabela 29 - Perfl dos usuários da ferramenta CAT

Usuário	Nível de Escolaridade	Formação em TI	Conhecimento em Teste - Manual			Conhecimento em Teste - Automação		
			Exploratório	Criação de caso de teste	Execução de caso de teste	Geral	JUnit	Selenium
1	Mestrado incompleto	Sim	0	0	1	0	1	0
2	Superior incompleto	Sim	0	1	1	0	0	0
3*	Mestrado completo	Sim	0	1	1	1	1	0
4	Superior incompleto	Sim	0	1	1	1	1	1
5	Mestrado incompleto	Sim	1	1	1	0	0	0
6	Mestrado incompleto	Sim	1	1	1	0	0	0
7	Mestrado incompleto	Sim	1	1	1	1	1	0
8	Mestrado completo	Sim	1	1	1	1	1	1
9	Mestrado incompleto	Sim	1	1	1	1	1	1
10	Mestrado incompleto	Sim	1	1	1	2	3	3
11	Doutorado incompleto	Sim	2	1	1	1	1	1
12	Mestrado incompleto	Sim	2	1	2	0	0	0
13	Mestrado incompleto	Sim	2	1	2	2	1	2
14	Mestrado incompleto	Sim	2	2	2	2	2	2
15	Superior incompleto	Sim	2	2	2	2	2	2
16*	Mestrado completo	Não	3	1	2	0	0	0

Apêndice 5 - Resposta dos testadores ao formulário do Apêndice 3

As Tabelas 30 a 34 apresentam os resultados por ordem das respostas enviadas. Serão exibidos os resultados para as três questões com respostas numéricas e a questão destacada, quando possível.

As questões e respostas serão referenciadas nas tabelas conforme:

- **Questão 1**: De 1 a 5, como você avalia a facilidade de testar uma aplicação utilizando a ferramenta CAT?
- **Questão 2**: De 1 a 5, como você avalia a facilidade de entendimento da ferramenta CAT?
- **Questão 3**: De 1 a 5, quão útil você julga a ferramenta?
- **Questão 4**: Quando você julgava o resultado esperado, o que você levava em consideração?
    - **Resposta 1**: O roteiro do caso de teste base
    - **Resposta 2**: Sua percepção individual
    - **Resposta 3**: Descrição do caso de teste

Tabela 30 - Resposta para a primeira rodada da aplicação Triângulo

Usuário	1	2	3	4	5	Média
Questão 1	4	3	4	5	4	4
Questão 2	3	3	4	5	3	3,6
Questão 3	5	4	5	5	4	4,6

Tabela 31 - Resposta para a segunda rodada da aplicação Triângulo

Usuário	1	2	3	4	Média
Questão 1	5	4	5	5	4,75
Questão 2	5	4	5	4	4,5
Questão 3	5	4	5	5	4,75

Tabela 32 - Resposta para a primeira rodada da aplicação TuduList

Usuário	1	2	3	4	5	6	7	8	9	Média
Questão 1	4	4	2	4	5	2	4	4	4	3,67
Questão 2	4	2	1	4	4	2	3	5	2	3
Questão 3	5	4	2	4	5	5	5	4	4	4,22

Tabela 33 - Resposta para a segunda rodada da aplicação TuduList

Usuário	1	2	3	4	5	6	7	Média
Questão 1	3	5	5	4	5	5	4	4,43
Questão 2	3	4	5	3	5	4	3	3,86
Questão 3	5	5	5	4	5	4	5	4,71
Questão 4	1	3	3	3	2	3	3	-

Tabela 34 - Resposta para a rodada da aplicação The Organizer

Usuário	1	2	3	4	5	6	7	Média
Questão 1	4	4	3	5	4	5	4	4,14
Questão 2	4	3	2	5	5	4	5	4
Questão 3	5	5	4	5	5	5	4	4,71
Questão 4	3	3	3	3	3	1	3	-

Apêndice 6 - Modelos de Dados da ferramenta CAT

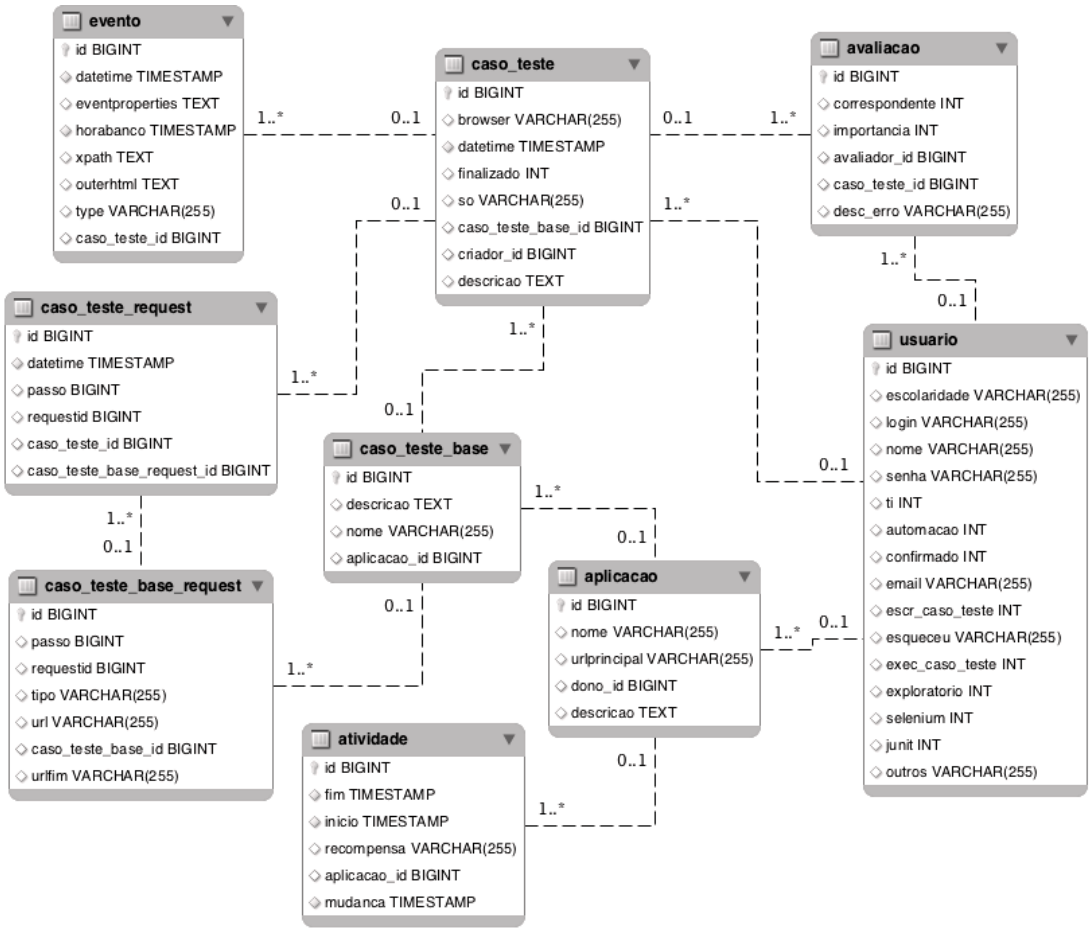

Figura 16 - Modelo de Dados da ferramenta CAT

Apêndice 7 - Modelos de Dados inseridos pela ferramenta CAT na aplicação testada

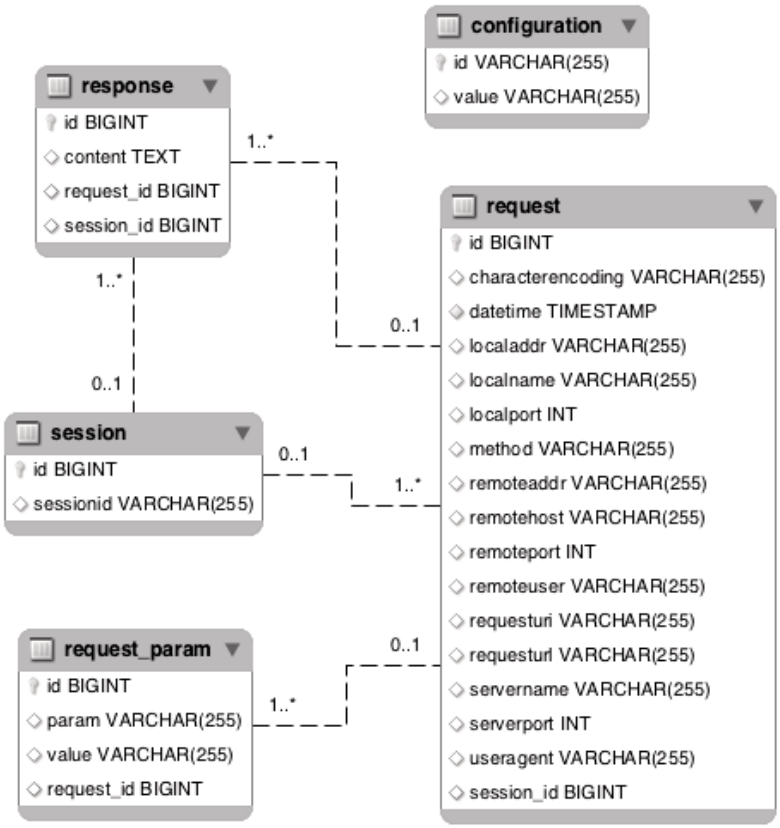

Figura 17 - Modelo de Dados inseridos pela ferramenta CAT na aplicação testada