Detector de perda de dados: revelando automaticamente erros de perda de dados em Aplicativos Android

Oliviero Riganelli
oliviero.riganelli@unimib.it
Universidade de Milão - Bicocca Milão, Itália

Simone Paolo Mottadelli
s.mottadelli2@campus.unimib.it
Universidade de Milão - Bicocca Milão, Itália

Claudio Rota
c.rota30@campus.unimib.it
Universidade de Milão - Bicocca Milão, Itália

Daniela Micucci
daniela.micucci@unimib.it
Universidade de Milão - Bicocca Milão, Itália

Leonardo Mariani
leonardo.mariani@unimib.it
Universidade de Milão - Bicocca Milão, Itália

RESUMO

Os aplicativos Android devem funcionar corretamente mesmo que sua execução seja interrompida por eventos externos. Por exemplo, um aplicativo deve funcionar corretamente mesmo se uma chamada telefônica for recebida ou depois que seu layout for redesenhado porque o smartphone foi girado. Como esses eventos podem exigir a destruição, quando a execução for interrompida, e a recriação , quando a execução for retomada, a atividade em primeiro plano do aplicativo, a única maneira de evitar a perda de informações de estado é salvá -las e restaurá-las. Esse comportamento deve ser implementado explicitamente pelos desenvolvedores de aplicativos, que muitas vezes deixam de implementá-lo corretamente, liberando aplicativos afetados por problemas de perda de dados, ou seja, aplicativos que podem perder informações de estado quando sua execução é interrompida.

Embora várias técnicas possam ser usadas para gerar automaticamente casos de teste para aplicativos Android, os casos de teste obtidos raramente incluem as interações e as verificações necessárias para exercitar e revelar falhas de perda de dados. Para resolver esse problema, este artigo apresenta o Data Loss Detector (DLD), uma técnica de geração de casos de teste que integra uma estratégia de exploração, ações de revelação de perda de dados e duas estratégias oracle customizadas para a detecção de falhas de perda de dados.

O DLD revelou 75% das falhas em um benchmark de 54 versões de aplicativos Android afetadas por 110 falhas conhecidas de perda de dados e também revelou problemas de perda de dados desconhecidos, superando as abordagens concorrentes.

CONCEITOS DE CCS

· Software e sua engenharia ÿ Teste e depuração de software.

PALAVRAS-CHAVE

Android, perda de dados, geração de casos de teste, validação, aplicativos móveis.

Formato de referência ACM:
Oliviero Riganelli, Simone Paolo Mottadelli, Claudio Rota, Daniela Micucci, and Leonardo Mariani. 2020. Detector de perda de dados: revelando automaticamente Erros de perda de dados em aplicativos Android. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18ÿ22, 2020, Virtual Event, USA. ACM, Nova York, NY, EUA, 12 páginas. https: //doi.org/10.1145/3395363.3397379

ISSTA '20, 18 e 22 de julho de 2020, Evento virtual,

ACM ISBN 978-1-4503-8008-9/20/07. . . $
15,00 https://doi.org/10.1145/3395363.3397379

1. INTRODUÇÃO

Na última década, os aplicativos móveis ganharam cada vez mais importância e popularidade. Estudos recentes revelaram que as pessoas passam, em média, mais de 3h por dia em seus smartphones [23] e que 90% desse tempo é tipicamente dedicado ao uso de aplicativos móveis [21]. De fato, as pessoas usam aplicativos móveis para realizar uma enorme variedade de tarefas, incluindo ler e-mails, ouvir música, fazer pedidos e pagamentos e jogar.

Entre os ecossistemas disponíveis para distribuição de aplicativos móveis, o ecossistema Android é o maior e mais utilizado: sua participação de mercado é de quase 75% [29] e sua loja oficial, a Google Play Store, inclui quase 3,0 milhões de aplicativos [30].].

Os aplicativos Android consistem em componentes, como atividades, fragmentos e serviços, cujo comportamento deve obedecer a ciclos de vida bem definidos [9, 11, 13]. Por exemplo, as atividades podem estar em estados como criado, pausado, retomado e interrompido, e as transições entre esses estados produzem retornos de chamada que devem ser tratados

Curiosamente, alguns desses retornos de chamada podem ser particularmente difíceis de implementar. Este é o caso dos callbacks produzidos por eventos de stop start, que são eventos de sistema que podem forçar a destruição e, em seguida, a (re)instanciação de uma atividade em execução. Os eventos stop-start ocorrem sempre que a execução de um aplicativo é interrompida e, em seguida, retomada. Casos típicos incluem atender uma chamada telefônica, alternar entre aplicativos e girar o smartphone para alterar seu layout.

Quando ocorre um evento stop-start, a tarefa difícil para o aplicativo é lidar com a destruição da atividade atual de forma que posteriormente seja possível retomar a execução no mesmo ponto em que foi interrompida. Isso é feito salvando os valores de todas as variáveis de estado relevantes antes que a atividade seja destruída e recuperando esses valores quando a execução for retomada. Com exceção de alguns casos específicos (por exemplo, os widgets com uma propriedade android:id não vazia), é responsabilidade do desenvolvedor implementar esse comportamento. Em particular, os desenvolvedores precisam implementar tanto a lógica necessária para salvar o estado de uma atividade no método de retorno de chamada onSaveInstanceState() quanto a lógica para retomar seu estado no método de retorno de chamada onRestoreInstanceState() [10]. Infelizmente, essa implementação pode estar errada e pode introduzir mau comportamento nos aplicativos [16].

Quando um evento stop-start não é tratado adequadamente, diz-se que o aplicativo Android é afetado por uma falha de perda de dados, ou seja, uma falha que faz com que uma ou mais variáveis de estado percam seus valores. Falhas de perda de dados podem afetar a correção dos aplicativos de várias maneiras diferentes. Nos melhores casos, eles forçam os usuários a inserir novamente entradas que já haviam sido inseridas, deteriorando a qualidade da experiência do usuário. Nos piores casos, eles geram atividades com um estado inconsistente, o que faz com que os aplicativos produzam saídas erradas ou até mesmo travam.

As falhas de perda de dados podem ser extremamente difundidas. Adamsen et ai. [1] considerou a execução de eventos do sistema em conjunto com suítes de testes e relatou que todos os quatro aplicativos usados em sua avaliação foram afetados por falhas de perda de dados. Riganelli et ai. [26] analisaram 428 aplicativos Android e descobriram que pelo menos 82 (19,6%) dos aplicativos foram afetados por falhas de perda de dados. Por fim, Amalfitano et al. [2] estudaram 68 aplicativos de código aberto relatando falhas de perda de dados em 60 deles (88,2%).

As técnicas de geração de casos de teste podem ser potencialmente usadas para revelar falhas de perda de dados. De fato, várias técnicas de geração automática de casos de teste estão disponíveis para aplicativos Android. Por exemplo, Monkey pode gerar casos de teste aleatoriamente [12], A3E pode gerar testes sistematicamente usando uma estratégia de profundidade [3], DroidBot [19] e Stoat [31] exploram uma abordagem baseada em modelo, e Sapienz usa algoritmos evolucionários [20]. Embora essas abordagens sejam capazes de revelar várias falhas interessantes, elas são ineficazes contra problemas de perda de dados por dois motivos: (i) não incluem operações que causam eventos de parada e partida e (ii) não são equipadas com oráculos fortes o suficiente para detectar falhas de perda de dados sem falhas.

Algumas técnicas foram projetadas para estender a capacidade de detecção de falhas de conjuntos de testes existentes para problemas de perda de dados. Por exemplo, Thor injeta sistematicamente sequências de eventos neutras, incluindo eventos de parada e partida, em casos de teste existentes para aumentar sua capacidade de detecção de falhas [1]. O Quantum se comporta de maneira semelhante, mas começa a partir de um modelo GUI do aplicativo [33]. Embora útil, sua aplicabilidade é limitada a aplicativos equipados com conjuntos de testes abrangentes ou modelos de GUI. O ALARic gera aleatoriamente casos de teste que incluem eventos stop-start e detecta problemas de perda de dados comparando o estado do aplicativo em teste antes e depois de um evento stop-start ser gerado [24]. O ALARic pode revelar falhas de perda de dados com sucesso, mas a estratégia de exploração e os oráculos adotados têm eficácia limitada, conforme relatado em nossa avaliação.

Neste artigo, apresentamos o Data Loss Detector (DLD), uma técnica de teste automático que pode revelar problemas de perda de dados em aplicativos Android. O DLD integra três recursos para revelar efetivamente problemas de perda de dados: (i) uma estratégia de exploração baseada em modelo tendencioso que direciona a exploração para (novos) estados de aplicativos que podem ser afetados por problemas de perda de dados, (ii) ações de revelação de perda de dados que aumentar a probabilidade de expor problemas de perda de dados e (iii) dois oráculos baseados em estado, um oráculo baseado em instantâneo e um oráculo baseado em propriedade, que possuem uma alta precisão de detecção de perda de dados, especialmente em conjunto.

Comparado com Thor [1] e Quantum [33], o DLD não requer um conjunto de testes pré-existente ou um modelo de GUI pré-existente, nem requer uma fase inicial de extração, mas gera iterativamente e continuamente casos de teste de acordo com o orçamento de tempo alocado . Finalmente, a exploração baseada em modelo tendencioso e as ações de revelação de perda de dados exploradas em DLD permitem uma detecção de perda de dados mais eficaz do que a estratégia implementada em ALARic [24].

Nós avaliamos empiricamente o DLD usando o benchmark de Riganelli et al. [26], que inclui 110 problemas de perda de dados que afetam 54 aplicativos lançamentos. O DLD detectou automaticamente 83 dos 110 (75%) problemas de perda de dados. O DLD também revelou 35 falhas de perda de dados que não faziam parte do benchmark, mas foram relatadas online em relatórios de bugs e 232 falhas de perda de dados anteriormente desconhecidas. Em geral, o DLD revelou três vezes as falhas de perda de dados reveladas por abordagens concorrentes. Finalmente enviamos as falhas de perda de dados que ainda afetam os aplicativos hoje em dia online para desenvolvedores de aplicativos que reagiram positivamente

Em poucas palavras, este artigo traz as seguintes contribuições.

- Descreve uma estratégia de exploração, ações de revelação de perda de dados e dois oráculos baseados em estado que podem ser incorporados em técnicas de teste para revelar problemas de perda de dados,
- Fornece a técnica Data Loss Detector (DLD), que é implementada como um extensão da ferramenta de geração de casos de teste DroidBot [19] para Android,
- Relata a maior avaliação empírica sobre detecção de perda de dados disponível até agora, considerando centenas de falhas de perda de dados,
- Entrega a ferramenta e o material experimental disponível gratuitamente online no seguinte url: https://bit.ly/30XLygW

Este artigo está organizado da seguinte forma. A Seção 2 discute as falhas de perda de dados e exemplifica as falhas típicas que podem ocorrer com aplicativos Android. A seção 3 descreve as principais contribuições técnicas deste artigo, incluindo a estratégia de exploração baseada em modelo tendencioso, as ações de revelação de perda de dados e os oráculos automáticos. A seção 4 relata resultados empíricos. A Seção 5 discute o trabalho relacionado. Por fim, a Seção 6 traz as considerações finais.

2 FALHAS DE PERDA DE DADOS

Nesta seção, descrevemos as falhas de perda de dados e os componentes do Android que podem ser afetados por essas falhas: atividades e fragme

Uma atividade do Android é um componente que implementa uma tela de um aplicativo e a lógica para lidar com essa tela. Para particionar uma tela em unidades menores, as atividades podem conter vários fragmentos, cada um contendo alguns elementos gráficos e a lógica para manipulá -los. Tanto as atividades quanto os fragmentos têm seu próprio ciclo de vida [9, 13].

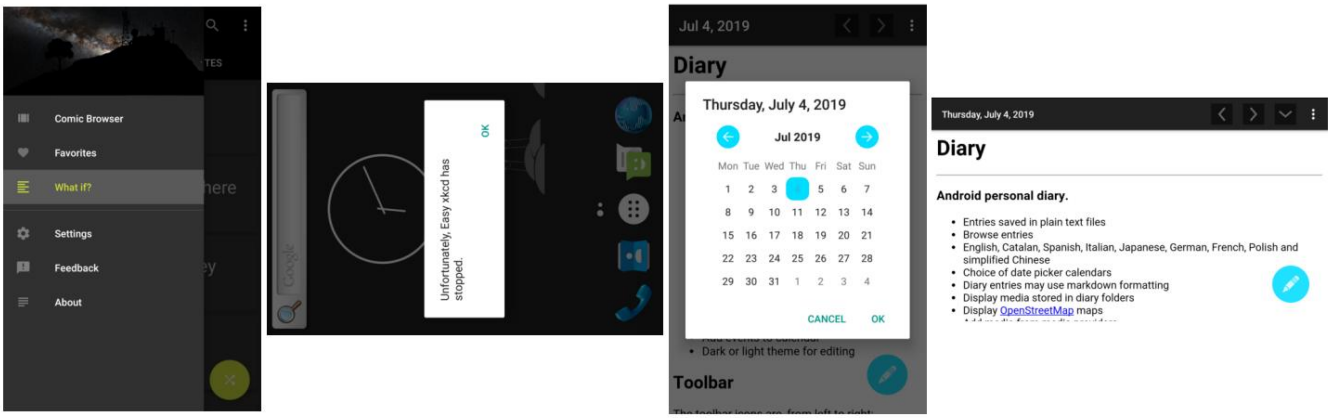
Sequências específicas de eventos do sistema podem ter um impacto direto sobre o ciclo de vida das atividades e fragmentos.

Definição 2.1. Um evento stop-start é uma sequência de eventos do sistema que podem fazer com que uma atividade em execução (ou fragmento) seja destruída e depois recriada[1] .

Os eventos stop-start são gerados quando a execução de uma atividade deve ser suspensa temporariamente. Existem muitas situações comuns que produzem eventos stop-start. Por exemplo, atender uma chamada telefônica requer a suspensão da execução do aplicativo e, portanto, também da atividade atual, até que a chamada termine. Mover um aplicativo para segundo plano pode fazer com que suas atividades sejam destruídas. Quando o aplicativo é movido novamente para o primeiro plano, o status da atividade do primeiro plano deve ser recriado. A rotação da tela finalmente causa a destruição da atividade atual que deve ser redesenhada com um novo layout.

Observe que todas essas situações são neutras do ponto de vista do usuário , ou seja, não se espera que alterem o status do aplicativo: os usuários esperam encontrar o status de um aplicativo inalterado após atenderem uma chamada telefônica, após o aplicativo ter foi movido para o segundo plano e depois para o primeiro plano, e depois que a tela

[1] No restante do artigo, nos referimos apenas a atividades por simplicidade, mas todos os conceitos se aplicam tanto a atividades quanto a fragmentos.



(a) Falha do aplicativo no Easy xkcd v6.0.4

(b) Desaparecimento de um diálogo no Diário v1.26

(c) Aparência de uma caixa de diálogo no CycleStreets v3.5

(d) Alteração dos valores EditText no BeeCount v2.7.4

(e) Perda do estado interno na Barra de Tarefas v3.0.3

Figura 1: Exemplos de falhas de perda de dados após um evento stop-start.

foi girado. No entanto, esse comportamento não é fornecido gratuitamente pelo Android, mas deve ser garantido pelos desenvolvedores que precisam implementar a lógica para salvar e recuperar o estado das atividades. Se essa lógica não for implementada corretamente, os eventos stop-start não serão mais neutros e as informações de estado poderão ser perdidas, causando um problema de perda de dados.

Definição 2.2. Um problema de perda de dados ocorre quando os dados são excluídos acidentalmente ou as variáveis de estado são atribuídas acidentalmente com valores padrão ou iniciais.

Falhas de perda de dados podem ser a fonte de diversas falhas [18]. De fato, a inicialização de algumas variáveis de programa com valores errados (por exemplo, para o valor padrão) pode ser a causa de comportamentos imprevisíveis. Com base em nossa avaliação, isolamos cinco padrões de falha principais que são específicos das atividades:

- trava: o aplicativo pode simplesmente travar. Isso é exemplificado na Figura 1 (a) onde o aplicativo Easy xkcd trava após uma rotação da tela.
- elementos GUI destruídos: alguns elementos gráficos podem desaparecer forçando o usuário a repetir as operações. Isso é exemplificado na Figura 1 (b) onde a caixa de diálogo do calendário no aplicativo Diário desaparece após uma rotação da tela.

• elementos de GUI fantasmas: alguns elementos gráficos podem aparecer erroneamente, forçando o usuário a realizar interações indesejadas e pouco claras . Isso é exemplificado na Figura 1 (c), onde uma caixa de diálogo aparece no aplicativo CycleStreets após uma rotação da tela. • valores modificados: alguns elementos podem alterar seus valores inesperadamente, resultando em mau comportamento do aplicativo. Isso é exemplificado na Figura 1 (d) onde vários campos de texto mudam para 0 no aplicativo BeeCount após uma rotação da tela.
• estado interno comprometido: o estado interno do aplicativo pode ser comprometido causando mau comportamento visível nas interações que seguem a ativação da perda de dados. Isto é exemplificado na Figura 1 (e). As rotações iniciais da tela comprometem o status do aplicativo da Barra de Tarefas sem produzir nenhum mau comportamento visível, até que a última rotação cause uma falha.

3 DETECTOR DE PERDA DE DADOS

O Data Loss Detector aborda falhas de perda de dados combinando três ingredientes principais: (i) uma estratégia de exploração baseada em modelo tendencioso, que aumenta a probabilidade de explorar estados que podem expor falhas de perda de dados; (ii) ações de revelação de perda de dados, que interagem com o aplicativo em teste estimulando comportamentos propensos à perda de dados e (iii) oráculos de perda de dados, que analisam o comportamento do aplicativo em teste para detectar falhas de perda de dados.

3.1 Exploração baseada em modelo tendencioso A

estratégia de geração de casos de teste implementada no DLD consiste em visitar tantos estados quanto possível e testar incrementalmente os estados recém-descobertos para detectar falhas de perda de dados. Para tanto, a estratégia constrói um modelo GUI que representa os estados visitados e as ações executadas. O modelo atende a dois propósitos principais: distinguir os estados já visitados (e testados) dos novos e direcionar a exploração para a execução de ações que podem potencialmente levar a estados nunca visitados antes.

Definição 3.1. Um modelo GUI é um estado finito não determinístico autômato (Q, **ÿ, q0, ÿ), onde** Q **é um conjunto finito de estados abstratos;** ÿ **é o conjunto finito de eventos que podem ser acionados a partir de tais estados abstratos, como cliques, swipes ou eventos stop-start; q0** ÿ Q **é o estado abstrato inicial;** ÿ : Q × ÿ ÿ ÿ(Q) **é a função de transição, que, dados** q ÿ Q **e** e ÿ ÿ, **retorna o conjunto de estados abstratos alcançáveis a partir de** q **executando e.**

Para testar efetivamente um aplicativo, é importante representar os estados em um nível de abstração apropriado. Uma representação muito abstrata colapsaria muitos estados concretos em um único estado abstrato, fazendo com que vários estados relevantes não fossem testados quanto à perda de dados. Uma representação muito concreta causaria uma enorme perda de tempo, testando estados de perda de dados com diferenças irrelevantes. O nível de abstração usado pelo DLD deriva das duas observações a seguir. • Os valores concretos não importam: uma representação de estado da GUI pode incluir todos os widgets com suas propriedades e valores. Como consequência, dois estados concretos que diferem para um único valor atribuído a um rótulo ou a um campo de entrada produziriam diferentes estados abstratos que seriam testados quanto à perda de dados. Isso é um desperdício de recursos, porque desde que alguns valores sejam atribuídos aos vários elementos da GUI, a perda de dados provavelmente aparecerá independentemente dos valores concretos atribuídos a esses elementos.

• Ignorar totalmente o conteúdo das telas é muito impreciso: uma representação do estado da GUI pode ser tão abstrata para considerar apenas o nome da atividade atual do Android como identificador do estado abstrato, ou seja, o número de estados abstratos corresponde ao número de Android atividades implementadas no aplicativo testado. Essa estratégia ignora totalmente o conteúdo das telas e provavelmente perderá todas as falhas de perda de dados que podem ser exercidas apenas em um estado específico de uma atividade.

Com base nessas duas observações, o DLD usa uma abstração que ignora os valores concretos, mas ainda discrimina os estados distintos relevantes associados a uma mesma atividade do Android. Para tanto, utiliza a abstração de habilitação, que já foi utilizada em outros contextos para distinguir os estados de aplicações de software [7, 8], e preliminarmente experimentada em Quantum para revelar falhas de interação da GUI [33]. Nesse caso, a ideia é distinguir estados não com base no conteúdo da tela, mas com base no conjunto de ações (ou seja, eventos) que são permitidas. Intuitivamente, vale a pena distinguir estados que possibilitam um conjunto diferente de comportamentos para o software. Por exemplo, dois valores diferentes em um campo de entrada produzem dois estados abstratos diferentes somente se um dos dois valores habilitar operações que, de outra forma, estariam desabilitadas.

Definição 3.2. Um estado abstrato q ÿ Q **associado a um estado concreto** s **é um par** (a, **E), onde** a **é o nome da atividade atual em se** E ÿ ÿ **é o conjunto de eventos habilitados em** s .

A estratégia de geração de casos de teste é direcionada para a execução de novas ações que podem levar a novos estados (abstratos) potencialmente afetados pela perda de dados. Em particular, toda vez que uma ação é executada, o DLD tem uma probabilidade ÿ de escolher um evento aleatoriamente e uma probabilidade 1 ÿÿ de escolher um evento que nunca foi executado no estado abstrato atual com base no modelo GUI disponível.

O DLD pode gerar cinco tipos de ações durante a exploração (quatro ações herdadas do DroidBot mais uma ação de rolagem adicionada para alcançar todas as visualizações em uma janela), além das ações de revelação de perda de dados descritas na próxima seção:

• TouchEvent, que executa um toque em uma visualização clicável; • LongTouchEvent, que executa um toque longo em uma visualização clicável; • SetTextEvent, que escreve texto dentro de uma view editável; • KeyEvent, que pressiona um botão de navegação (por exemplo, ÿHomeÿ). • ScrollEvent, que executa um swipe em uma visualização rolável;

O DLD atualiza incrementalmente o modelo de GUI após a execução de cada evento. A atividade de teste é interrompida após um orçamento que pode ser expresso como um número de ações a serem executadas ou como uma quantidade de tempo a ser alocada para teste.

3.2 Ações de Revelação de Perda de Dados A

atividade de exploração descrita na Seção 3.1 é combinada com a execução de ações de revelação de perda de dados que têm o objetivo de revelar problemas de perda de dados, se presentes.

O DLD inclui duas ações de revelação de perda de dados: uma é executada sistematicamente toda vez que um novo estado abstrato é alcançado (ação de revelação de perda de dados sistemática), enquanto a outra é executada probabilisticamente em cada estado abstrato (revelação de perda de dados probabilística). aça

A ação sistemática de revelação de perda de dados é composta pela seguinte sequência de etapas:

(1) preenchimento: o DLD interage com todas as visualizações de entrada (por exemplo, campo de texto, caixa de combinação, caixa de seleção) inserindo valores não vazios diferentes dos valores padrão, (2) estado de salvamento: o atual O estado da GUI é salvo para verificar posteriormente se ocorreu alguma perda de dados. A forma como o estado é salvo depende da estratégia oracle adotada (ver Seção 3.3), (3) dupla rotação da tela: a rotação da tela é um evento stop-start.

Ele é executado duas vezes para atingir um estado que deve ser exatamente o mesmo que foi salvo, se não ocorreu nenhuma perda de dados, (4) estado de verificação: o estado atual da GUI é comparado ao estado salvo para determinar se ocorreu alguma perda de dados. A forma como a comparação é realizada depende da estratégia do oráculo (consulte a Seção 3.3), (5) role para baixo: algumas atividades do Android podem incluir elementos visuais que abrangem o tamanho da tela. Para garantir que o estado abstrato alcançado seja totalmente testado quanto à perda de dados, incluindo os elementos que podem estar fora da tela, se a etapa de verificação do estado não revelou uma perda de dados, o DLD executa uma ação de rolagem para baixo que pode fazer com que novos elementos apareçam. Se isso acontecer, um novo estado abstrato pode ser alcançado, que seria novamente testado sistematicamente quanto à perda de dados.

A ação sistemática de revelação de perda de dados já garante uma validação precisa do espaço de estados, conforme definido por nossa estratégia de abstração. No entanto, pode haver certas falhas de perda de dados que dependem de informações de estado interno que não produzem nenhuma diferença no estado da GUI do aplicativo. Por exemplo, a janela para definir um temporizador no aplicativo AntennaPod gera uma perda de dados apenas se um podcast tiver sido carregado antes de outra janela. O fato de um podcast ter sido carregado não resulta em nenhuma diferença visível na janela do timer e, portanto, a estratégia de abstração não consegue capturar a diferença entre o estado que expõe a falha de perda de dados e aquele que não. Para resolver esses casos, quando um estado já visitado é encontrado, o DLD habilita a ação de revelação de perda de dados probabilística que tem a mesma probabilidade das outras ações serem selecionadas. A ação de revelação de perda de dados probabilística executa a sequência de eventos da etapa 2 a 5 da ação de revelação de perda de dados sistemática.

Em poucas palavras, a exploração funciona da seguinte forma. Quando um novo estado abstrato é encontrado, o DLD executa a ação sistemática de revelação de perda de dados. Caso contrário, o DLD identifica o conjunto de ações A que podem ser executadas no estado atual da GUI com base nos cinco tipos de ações suportadas (consulte a Seção 3.1). Ou seja, um subconjunto deles, A +, já foi executado com base no modelo GUI, enquanto os outros, A ÿ, ainda não foram executados. DLD executa com probabilidade ÿ uma ação aleatória, ou seja, uma ação no conjunto A ÿ {ação probabilistic data-loss-revealing action}, e com probabilidade 1ÿÿ uma ação que ainda não foi executada, ou seja, uma ação em A ÿ {ação de revelação de perda de dados probabilística}.

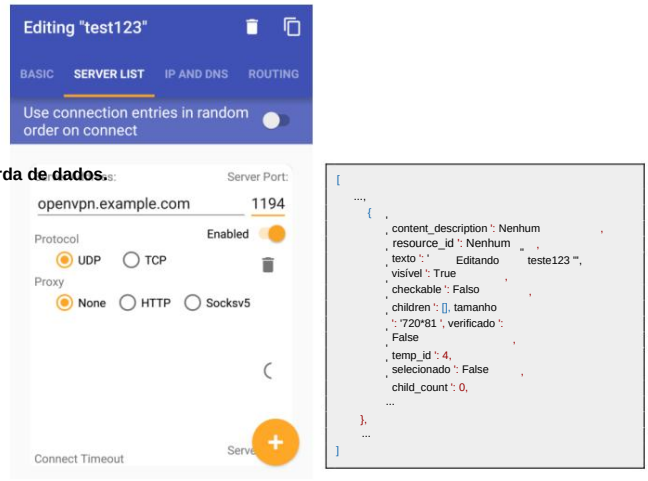
3.3 Oráculos de Perda de Dados

Os problemas de perda de dados nem sempre causam falhas. Pelo contrário, os aplicativos podem apresentar uma série de comportamentos inadequados, conforme discutido na Seção 2.

O DLD usa oráculos com base no fato de que as operações que exercem as falhas de perda de dados devem ser neutras, deixando o status do aplicativo inalterado. Conforme previsto na Seção 2.2, o básico

A estratégia de DLD é coletar o estado da GUI do aplicativo antes e depois da execução de ações que podem ter acionado uma falha de perda de dados e comparar os estados para detectá-la.

O DLD define duas estratégias de oráculo, que podem ser usadas de forma independente ou conjunta: o oráculo baseado em instantâneos e o oráculo baseado em propriedades. O oráculo baseado em instantâneo faz uma captura de tela do aplicativo antes e depois que uma perda de dados pode ter ocorrido e compara as imagens para detectar falhas. O oráculo baseado em propriedades analisa o estado da GUI e coleta todas as visualizações e todas as suas propriedades e compara esses dois conjuntos de propriedades para detectar falhas. A Figura 2 mostra as informações de estado capturadas pelos oráculos baseados em instantâneos e baseados em propriedades para um mesmo estado de GUI do aplicativo OpenVPN. O antigo oráculo armazena uma captura de tela do aplicativo, conforme mostrado na Figura 2 (a), enquanto o último oráculo armazena as propriedades das visualizaç



(a) Oráculo baseado em instantâneo (b) Oráculo baseado em propriedade

Figura 2: As informações salvas pelos dois tipos de oráculos para um mesmo estado do OpenVPN.

Metapa 2 a 5 da ação de revelação de perda de dados sistemática comparam as informações da seguinte forma.

Informações de estado do oráculo baseadas em instantâneo : o DLD primeiro faz uma captura de tela do dispositivo. A imagem gravada é então convertida em uma imagem em tons de cinza, que é mais rápida de comparar do que uma imagem colorida. Por fim, o DLD corta o cabeçalho e o rodapé da imagem porque contém informações que mudam ao longo do tempo, independentemente da perda de dados, como a hora atual e o nível da bateria. A imagem resultante é a representação recuperada do estado atual.

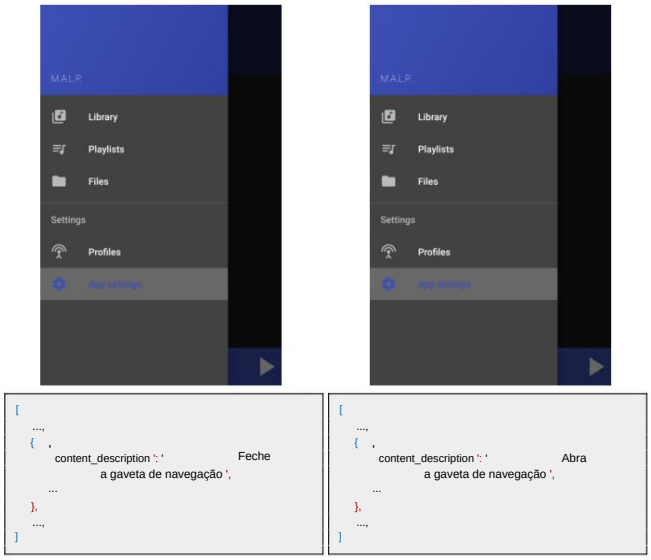
Comparação de estado: o DLD compara os dois estados comparando as duas imagens correspondentes pixel por pixel. Como um cursor piscando pode causar um pequeno nível de ruído na representação das imagens, a comparação falhará apenas se mais de 15 pixels a cada 10.000 pixels forem diferentes.

Informações de estado do oráculo baseado em propriedades : o DLD recupera todas as visualizações, incluindo suas propriedades e sua organização hierárquica. Os valores são representados em um formato de dicionário Python.

Comparação de estado: o DLD compara os dois estados comparando sua representação baseada em Python. A comparação falha se um dos valores de atributo for diferente ou a estrutura hierárquica das visualizações não é preservado.

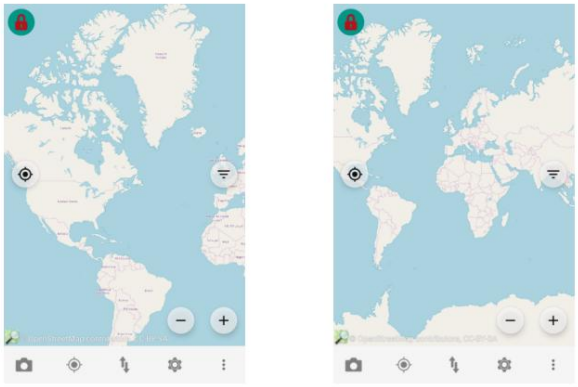
Observe que, embora as duas estratégias possam parecer redundantes, elas não são, conforme confirmado por nossos experimentos. Em particular, há vários problemas de perda de dados que só podem ser detectados por uma estratégia. Por exemplo, a Figura 3 mostra o caso de uma falha de perda de dados que foi detectada apenas pelo oráculo baseado em propriedade. Na verdade, os dois instantâneos do aplicativo MALP são visualmente idênticos, mas na verdade o descritor de conteúdo mudou seu valor. Por outro lado, a Figura 4 mostra o caso de uma falha de perda de dados que só foi detectada pelo oráculo baseado em snapshot. O conjunto de propriedades, não mostrado na figura, é exatamente o mesmo para os dois estados, mas o aplicativo perdeu o zoom, como é claramente visível nos instantâneos.

Curiosamente, os dois oráculos podem ser combinados, de modo que uma falha seja relatada se apenas uma das duas estratégias de oráculo relatar uma falha.



(a) Antes da dupla rotação **(b) Após a rotação dupla**

Figura 3: Uma falha de perda de dados detectada pelo oráculo baseado em propriedade somente no MALP 3d31062.



(a) Antes da dupla rotação **(b) Após a rotação dupla**

Figura 4: Uma falha de perda de dados detectada pelo oráculo baseado em instantâneo somente no Vespucci Osm Editor v10.2.

4 AVALIAÇÃO

Esta seção apresenta os resultados empíricos obtidos experimentando DLD com um benchmark de 110 falhas de perda de dados, em comparação com as técnicas de geração de casos de teste ALARic [24] e Quantum [33] . Primeiro descrevemos nossa implementação do DLD, depois apresentamos as questões de pesquisa e as aplicações do assunto. Finalmente apresentamos os resultados obtidos para cada questão de pesquisa em detalhes e discutimos as ameaças à validade.

4.1 Implementação

Implementamos o DLD como uma extensão do DroidBot [19], que é um gerador de casos de teste de última geração para Android que não implementar recursos para a detecção de falhas de perda de dados. O DLD herda do DroidBot a capacidade de executar uma sequência específica de ações antes de testar um aplicativo de destino. Esse recurso pode ser usado para configurar o estado inicial do aplicativo em teste. Em nossa avaliação, usamos esse recurso para: autenticar nos aplicativos que exigem login, configurar um projeto inicial no MGit e conceder per

Como resultado do processo de teste, o DLD gera um relatório com as falhas de perda de dados reveladas e casos de teste reproduzíveis. Cada perda de dados é descrita em termos de capturas de tela e do conjunto de GUI propriedades coletadas antes e depois que a perda de dados é observada. O DLD está disponível em https://bit.ly/30XLygW.

4.2 Perguntas de pesquisa

Avaliamos o DLD estudando as cinco perguntas de pesquisa a seguir.

- **RQ0 - Qual é o ÿ que proporciona a melhor exploração? Esta questão de pesquisa estuda o impacto do parâmetro ÿ na eficácia da exploração. O resultado desta pergunta de pesquisa é usado para configurar o DLD para abordar as outras perguntas de pesquisa.**
- **RQ1 - Qual é a eficácia do DLD com problemas de perda de dados? Esta questão de pesquisa investiga a eficácia do DLD considerando duas perspectivas capturadas pelos seguintes sub-RQs. ¶ RQ1.1 - Qual é o recurso de descoberta de perda de dados do DLD? ¶ RQ1.2 - Qual é a taxa de violações espúrias do oráculo relatadas por DLD?**
- **RQ2 - O DLD é mais eficaz do que as técnicas de ponta? Esta questão de pesquisa é decomposta nos seguintes sub-RQs. ¶ RQ2.1 - Qual é a eficácia relativa do DLD, ALARic e Quantum? Este sub-RQ compara DLD a ALARic e Quan vir.**

  ¶ RQ2.2 - Quais são os principais fatores que determinam a eficácia do DLD? Este sub-RQ investiga os fatores que permitem que o DLD revele mais falhas de perda de dados do que as técnicas concorrentes.
- **RQ3 - Qual é a relação entre os oráculos baseados em instantâneos e propriedades? Esta questão de pesquisa investiga a compensação entre as duas estratégias de oráculo, medindo as falhas de perda de dados reveladas apenas pelo oráculo baseado em instantâneo, apenas pelo oráculo baseado em propriedade e por ambos.**

- RQ4 - As falhas de perda de dados são relevantes para os desenvolvedores? Esta questão de pesquisa estuda como os desenvolvedores de aplicativos reagem à presença de problemas de perda de dados em seus aplicativos.

4.3 Aplicações de Assuntos Em nossa

avaliação empírica usamos o benchmark de Riganelli et al. [26], que inclui 110 problemas de perda de dados que afetam um total de 48 aplicativos Android e 54 versões de aplicativos. Cada falha de perda de dados é equipada com um caso de teste Appium [14] que pode ser executado para reproduzir o problema. Todos os experimentos foram conduzidos com o emulador Genymotion v3.0.2 Android usando um dispositivo Google Nexus 5 emulado equipado com Android 6.0 API 23 e 2 GB de RAM. Na avaliação, seguimos a prática de outros estudos [32] realizando três execuções de 3 horas cada por aplicativo testado para DLD e ALARic, para um total de 42 dias de computação ininterrupta.

4.4 RQ0 - Qual é o ÿ que proporciona a melhor exploração?

Esta questão de pesquisa investiga o impacto do parâmetro ÿ na eficácia da abordagem. Para realizar este estudo inicial selecionamos 3 aplicativos do benchmark. Para cobrir a gama de situações que podem ser enfrentadas com o benchmark completo, selecionamos os aplicativos com menor, médio e maior número de atividades, que são Eqate (2 atividades), Calendar Notification (13 atividades) e Twidere (52 atividades). atividades), respectivamente.

Para avaliar a capacidade de explorar o aplicativo e potencialmente revelar problemas de perda de dados, medimos a cobertura de atividade, que é a porcentagem de atividades cobertas em uma sessão de teste. Iniciamos com ÿ = 0, que consiste em uma estratégia que sempre privilegia a execução de ações que não foram executadas anteriormente, com base no modelo de GUI construído de forma incremental. Em seguida, aumentamos o parâmetro ÿ em 0,1 para estudar seu impacto nos resultados. Paramos com ÿ = 0,2, o que produziu uma diminuição significativa na eficácia da exploração. Os resultados estão resumidos na Tabela 1. Finalmente selecionamos ÿ = 0,1 para abordar as outras questões de pesquisa.

Tabela 1: Impacto do parâmetro ÿ na cobertura da atividade.

ÿ	Cobertura de atividade
ÿ = 0 (sempre novas ações)	52%
ÿ = 0,1 (ações aleatórias com probabilidade 0,1)	60%
ÿ = 0,2 (ações aleatórias com probabilidade 0,2)	44%

4.5 RQ1 - Qual é a eficácia do DLD com problemas de perda de dados?

Esta questão de pesquisa investiga a capacidade do DLD de revelar os problemas de perda de dados no benchmark. Para responder a essa pergunta de pesquisa, analisamos manualmente todos os relatórios produzidos pelo DLD para distinguir os problemas reais de perda de dados das violações espúrias irrelevantes. Em particular, classificamos uma perda de dados relatada como espúria se uma das duas condições a seguir for válida: (i) o estado do aplicativo após a rotação de tela dupla for feito muito cedo, enquanto a atividade ainda está sendo recriada, fazendo com que o oráculo falhe sua verificação ou (ii) a diferença informada pelo oráculo não pode ser considerada perda de dados (por exemplo, porque o aplicativo exibe o tempo na tela que obviamente muda após a tela ter sido girada duas vezes). Na grande maioria dos casos os relatórios foram suficientes para classificar os problemas de perda de dados. Reproduzimos o problema nos casos pouco claros. Para esta questão de pesquisa, detectamos problemas de perda de dados usando os oráculos baseados em instantâneos e baseados em propriedades. Analisamos sua capacidade relativa de detecção de falhas c

Verificamos cada problema de perda de dados revelado pelo DLD, distinguindo se é uma perda de dados de benchmark, ou seja, um problema que faz parte do benchmark que usamos; uma perda de dados online, ou seja, um problema já relatado online que não faz parte do benchmark (para todos os aplicativos no benchmark, pesquisamos online por falhas adicionais de perda de dados e usamos os instantâneos, o nome da atividade e os campos relatados perder seus valores para determinar se uma perda de dados descoberta corresponde à perda de dados online) ou uma nova perda de dados, ou seja, um problema de perda de dados desconhecido (com o melhor de nossa capacidade de procurar problemas relatados). Referimo-nos à união das falhas de perda de dados de refe

Relatamos o número de atividades afetadas por um problema de perda de dados encontrado pelo DLD. Eles correspondem intuitivamente a diferentes falhas e diferentes correções a serem implementadas em diferentes atividades. A única exceção é com as falhas de perda de dados conhecidas. Como algumas das falhas de perda de dados no benchmark afetam a mesma atividade, na verdade relatamos o número preciso de falhas no benchmark que foram reveladas. Por fim, quando apenas violações espúrias são detectadas para uma atividade, relatamos isso como uma

A coluna DLD da Tabela 2 (parte esquerda da tabela) mostra os resultados que o DLD obteve para todas as versões de aplicativos consideradas no estudo. Como cada linha representa o resultado de três execuções, quando aplicável, informamos os valores médios e os valores totais. Coluna # Atividades indica o número total de atividades em cada aplicativo. A coluna Cobertura de atividade média (total) informa a cobertura de atividade alcançada em média e no total nas três execuções. A coluna Atividades com perda de dados indica o número de atividades afetadas por pelo menos uma perda de dados revelada pelo DLD. Column Benchmark Data Loss avg (total)/existing indica o número médio e total de falhas de perda de dados que foram reveladas pelo DLD dentre as presentes no benchmark. Por exemplo, no aplicativo BookCatalogue, o DLD revelou 5 falhas de perda de dados do benchmark em média, alcançou um total de 6 falhas de perda de dados reveladas nas três execuções, de um total de 7 falhas de perda de dados presentes no benchmark. Da mesma forma, a coluna Online Data Loss avg (total)/ existing indica o número médio e total de falhas de perda de dados relatadas online reveladas pelo DLD. A coluna New Data Loss avg (total) indica o número médio e total de falhas de perda de dados anteriormente desconhecidas reveladas. A coluna Spurious Data Loss avg (total) indica o número médio e total de atividades que originaram apenas violações falsas. As falhas de coluna relatam o número total de atividades que falharam devido a fal

A parte superior da tabela lista os aplicativos em que nenhuma configuração inicial foi necessária, enquanto a parte inferior da tabela lista os aplicativos que foram abordados conforme discutido na Seção 4.1.

4.5.1 RQ1.1 - Qual é a capacidade de descoberta de perda de dados do DLD?.

Em termos de exploração, o DLD conseguiu visitar 66% das atividades, revelando 298 atividades afetadas por falhas de perda de dados (41% do total de atividades). Ele detectou 83 das 110 falhas no benchmark (75%), 35 das 58 (60%) falhas adicionais de perda de dados que encontramos online e revelou 232 novas falhas de perda de dados (uma média de

Tabela 2: Resultados para DLD e comparação com ALARic.

			DLD							ALARic				
Nome do aplicativo	Lançamento	# Atividades	Cobertura de Atividade média (total)	Atividades com Perda de dados	Referência Perda de dados média (total)/existente	Conectados Perda de dados média (total)/existente	Novo Perda de dados média (total)	Espúrio Perda de dados média (total)	Falhas	Atividades com Perda de dados	Referência Perda de dados média (total)/existente	Conectados Perda de dados média (total)/existente	Novo Perda de dados média (total)	Espúrio Perda de dados média (total)
Gerenciador de Arquivos Surpreender	v3.1.0-beta.1	4	100% (100%)	3	3 (3)/5	2 (2)/2	1(1)	0 (0)	1	1	1 (1)/5	0 (0)/2	0 (0)	0 (0)
AntenaPod	v1.5.2.0	16	33% (44%)	5	5 (5)/7	2 (2)/11	3 (3)	1 (1)	1	1	0 (0)/7	1 (1)/11	0 (0)	0 (0)
BeeCount	v2.4.7	8	96% (100%)	7	1 (1)/3	1 (1)/5	5 (5)	1 (1)	0	5	0 (0)/3	0 (0)/5	5 (5)	1 (1)
Catálogo de livros	v5.2.0-RC3a	35	66% (71%)	21	5 (6)/7	-	12 (15)	0 (0)	1	7	2 (2)/7	-	4 (5)	6 (7)
Notificação de calendário	v3.14.159	13	67% (69%)	8	3 (3)/3	1 (1)/1	4 (5)	0 (0)	1	2	1 (1)/3	0 (0)/1	1 (1)	0 (0)
Ciclovias	v3.5	11	55% (55%)	6	1 (1)/1	-	5 (5)	0 (0)	0	1	1 (1)/1	-	0 (0)	0 (0)
Diário	v1.26	3	100% (100%)	3	1 (2)/2	-	2 (2)	0 (0)	0	2	1 (1)/2	-	1 (1)	0 (0)
DNS66	v0.3.3	5	100% (100%)	3	1 (1)/1	-	2 (2)	0 (0)	0	1	0 (0)/1	-	1 (1)	2 (2)
Visualizador de Documentos	v2.7.9	9	48% (56%)	2	1 (1)/1	-	2 (2)	1 (1)	0	1	1 (1)/1	-	0 (0)	4 (4)
Fácil xkcd	v6.0.4	9	74% (78%)	4	1 (1)/1	-	3 (3)	0 (0)	3	1	0 (0)/1	-	1 (1)	1 (1)
Equacionar	v1.6	2	100% (100%)	2	2 (2)/2	1 (1)/1	1 (1)	0 (0)	1	1	2 (2)/2	1 (1)/1	0 (0)	0 (0)
Calendário Etar	v1.0.10	12	42% (42%)	3	4 (4)/5	2 (3)/5	1 (1)	0 (0)	0	0	0 (0)/5	0 (0)/5	0 (0)	0 (0)
Foco do Firefox	v4.0	6	44% (50%)	3	0 (0)/1	-	3 (3)	0 (0)	0	2	0 (0)/1	-	2 (2)	0 (0)
Flym	v1.3.4	6	83% (83%)	4	0 (0)/1	-	4 (4)	0 (0)	0	3	0 (0)/1	-	3 (3)	0 (0)
Ponte de gadgets	v0.25.1	20	28% (30%)	4	1 (1)/1	-	2 (3)	2 (2)	2	0	0 (0)/1	-	0 (0)	0 (0)
Lançador KISS	v2.25.0	2	100% (100%)	1	1 (1)/1	-	0 (0)	1 (1)	0	1	0 (0)/1	-	1 (1)	0 (0)
Rastreador de hábitos de loop	v1.6.2	7	71% (71%)	4	2 (2)/6	-	1 (2)	1 (1)	0	2	0 (0)/6	-	2 (2)	2 (2)
MALP	3d31062	2	100% (100%)	1	1 (1)/1	-	0 (0)	0 (0)	1	0	0 (0)/1	-	0 (0)	0 (0)
MALP	v1.1.0	4	33% (50%)	2	2 (3)/4	-	1 (1)	0 (0)	1	1	0 (0)/4	-	1 (1)	0 (0)
Familiares de MTG	v3.5.5	2	50% (50%)	1	1 (1)/1	-	0 (0)	0 (0)	0	1	0 (0)/1	-	1 (1)	0 (0)
Bloco de anotações	v2.3	3	67% (67%)	2	1 (1)/1	-	1 (1)	0 (0)	0	1	1 (1)/1	-	0 (0)	0 (0)
Notas Omni	v5.4.3	17	29% (35%)	4	0 (0)/1	-	4 (4)	0 (0)	0	1	0 (0)/1	-	1 (1)	1 (1)
OpenTasks	v1.1.13	9	78% (78%)	7	1 (1)/1	-	6 (6)	0 (0)	0	3	0 (0)/1	-	3 (3)	1 (1)
OpenVPN para Android	v0.7.5	13	46% (46%)	5	1 (1)/1	-	4 (4)	0 (0)	1	4	0 (0)/1	-	3 (4)	1 (1)
Passe Android	v3.3.3	14	36% (36%)	4	2 (3)/3	8 (8)/8	1 (1)	0 (0)	1	3	1 (1)/3	4 (4)/8	1 (1)	1 (1)
Tabela periódica	v1.1.1	3	100% (100%)	3	2 (2)/2	-	1 (1)	0 (0)	0	0	0 (0)/2	-	0 (0)	2 (2)
Aldrava de porta	v1.0.8	6	50% (50%)	2	2 (2)/3	0 (0)/1	0 (0)	0 (0)	0	2	0 (0)/3	0 (0)/1	2 (2)	1 (1)
Tempos de oração	v3.6.6	22	32% (36%)	8	3 (6)/7	1 (1)/2	4 (5)	0 (0)	1	5	1 (1)/7	0 (0)/2	3 (4)	0 (0)
QuasselDroid	v0.11.5	5	40% (40%)	2	1 (1)/1	-	1 (1)	0 (0)	1	0	0 (0)/1	-	0 (0)	1 (1)
Sorteio Simples	v3.1.5	7	86% (86%)	3	0 (0)/1	-	3 (3)	0 (0)	0	0	0 (0)/1	-	0 (0)	1 (1)
Gerenciador de arquivos simples	v2.6.0	8	88% (88%)	5	1 (1)/1	-	3 (4)	0 (0)	2	1	1 (1)/1	-	0 (0)	1 (1)
Gerenciador de arquivos simples	v3.2.0	8	75% (75%)	5	1 (1)/1	-	3 (4)	0 (0)	1	1	1 (1)/1	-	0 (0)	0 (0)
Galeria simples	v1.50	11	48% (55%)	5	1 (2)/4	1 (1)/7	3 (3)	0 (0)	0	1	1 (1)/4	0 (0)/7	0 (0)	1 (1)
Paciência Simples	v2.0.1	7	93% (100%)	2	1 (1)/1	-	1 (1)	0 (0)	1	1	0 (0)/1	-	1 (1)	2 (2)
Simpletask	v10.0.7	11	67% (73%)	7	1 (1)/1	-	6 (6)	0 (0)	1	4	1 (1)/1	-	3 (3)	0 (0)
Sincronização	v0.9.5	9	93% (100%)	8	3 (4)/5	2 (2)/2	4 (5)	1 (1)	2	2	1 (1)/5	1 (1)/2	0 (0)	0 (0)
Barra de tarefas	v3.0.3	21	26% (29%)	3	2 (2)/2	12 (13)/13	2 (2)	1 (1)	1	1	1 (1)/2	1 (1)/13	0 (0)	0 (0)
Tarefas Astrid To-Do List Clone	v6.0.6	45	19% (27%)	10	0 (0)/1	-	7 (10)	1 (1)	2	1	0 (0)/1	-	1 (1)	0 (0)
Editor de Vespucci Osm	v10.2	19	42% (47%)	7	1 (1)/1	-	5 (6)	1 (1)	0	5	1 (1)/1	-	4 (4)	0 (0)
Verificador de Vlille	v4.4.0	6	67% (67%)	3	1 (1)/1	-	2 (2)	0 (0)	0	1	0 (0)/1	-	1 (1)	0 (0)
Analisador WiFi	1.9.2	4	75% (75%)	3	3 (3)/3	-	1 (1)	0 (0)	0	0	0 (0)/3	-	0 (0)	0 (0)
Relógio mundial e clima	v1.8.6	4	100% (100%)	4	0 (0)/1	-	4 (4)	0 (0)	0	1	0 (0)/1	-	1 (1)	1 (1)
TOTAL		428	65% (68%)	189	73/97	35/58	132	0,26 (11)	26	71	19/97	8/58	50	0,74 (31)
Apps testados após ações de configuração inicial														
Conversas	v1.14.0	21	41% (57%)	6	0 (0)/1	-	5 (6)	0 (0)	1					
Conversas	v1.23.8	23	40% (57%)	10	1 (1)/1	-	6 (9)	0 (0)	0					
Correio K-9	v5.010	28	41% (46%)	8	0 (0)/1	-	6 (8)	0 (0)	1					
Correio K-9	v5.207	27	51% (63%)	7	1 (1)/1	-	5 (7)	0 (0)	1					
Correio K-9	v5.401	29	54% (59%)	8	1 (1)/1	-	6 (7)	0 (0)	1					
Mgit	v1.5.0	10	97% (100%)	9	1 (1)/1	-	7 (8)	1 (1)	2					
OctoDroid	v4.0.3	44	56% (66%)	21	2 (2)/2	-	16 (19)	0 (0)	3					
OctoDroid	v4.2.0	46	44% (57%)	22	1 (1)/1	-	17 (21)	1 (2)	0					
QuickLyric	v2.1	4	75% (75%)	3	1 (1)/1	-	2 (2)	0 (0)	0					
SMS Backup Plus	v1.5.11-Beta18	7	14% (14%)	1	1 (1)/1	-	0 (0)	0 (0)	0					
Tusky para Mastodonte	v1.0.3	12	78% (83%)	8	1 (1)/1	-	7 (7)	0 (0)	3					
Twidere	v3.7.3	52	14% (17%)	6	0 (0)/1	-	6 (6)	0 (0)	0					
TOTAL (todos os aplicativos)		731	62% (66%)	298	83/110	35/58	232	0,26 (14)	38					

4,3 novas falhas de perda de dados reveladas por aplicativo). No total, o DLD revelou 350 problemas de perda de dados em 54 versões de aplicativos, demonstrando um capacidade de detectar problemas de perda de dados. Observe que iniciamos o investigação empírica sabendo que menos de 110 atividades foram afetados por problemas de perda de dados e acabamos descobrindo 298 atividades afetadas por problemas de perda de dados. Curiosamente, a ação probabilística de revelação de perda de dados contribuiu revelando a perda de dados em 158 atividades já relatadas pela ação sistemática e em 17 atividades não relatadas pela ação sistemática de revelação de perda de dados.

Investigamos manualmente os 50 casos de falhas conhecidas de perda de dados que não foram revelados pelo DLD (27 casos no benchmark e 23 casos recuperados online). Isolamos três razões principais pelas quais problemas de perda de dados foram perdidos.

- Sequências de baixa probabilidade: revelar essas falhas de perda de dados requer a geração de uma sequência de eventos com baixa probabilidade de ser gerada, devido ao comprimento da sequência e/o o grande número de ações que podem ser geradas a cada passo do processo de teste.
- Configuração do ambiente: a detecção desses problemas de perda de dados requer uma configuração específica do ambiente. Por exemplo, um dado perda que afeta o aplicativo Document Viewer requer a presença de um documento a ser revelado. Essas falhas podem ser potencialmente revelado com esforço adicional na configuração do aplicativo em teste.

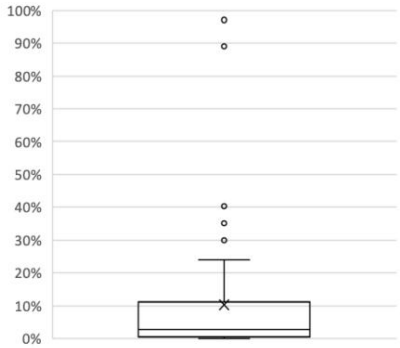


Figura 5: Porcentagem de violações espúrias do oráculo retornadas por aplicativo.

• **Ações não suportadas: essas falhas de perda de dados são impossíveis de revelam com DLD porque exigem a execução de operações que estão fora do escopo do DLD. Por exemplo, detectar um dos as falhas de perda de dados no aplicativo Tasks Astrid To-Do List Clone requer sair temporariamente do aplicativo e tirar uma foto com a câmera do dispositivo, o que não pode ser feito com DLD.**

No geral, encontramos 41 problemas de perda de dados que simplesmente têm baixa probabilidade de serem revelados, 4 problemas de perda de dados que requerem uma configuração do ambiente e 5 problemas de perda de dados que exigem uma extensa capacidade de exploração a ser revelada. Curiosamente, vários falhas podem ser potencialmente reveladas apenas executando o DLD por um mais tempo, ou refinando a estratégia de exploração de DLD, para que são geradas combinações específicas de ações. No entanto, gerando combinações complexas e longas de ações podem ser desafiadoras.

Por fim, apenas 38 das 298 atividades afetadas por falhas de perda de dados produzia crashes, o que confirma a necessidade de oráculos específicos para lidar com esses problemas.

4.5.2 RQ1.2 - Qual é o índice de violações espúrias do oráculo reportado por DLD?. DLD teve um bom desempenho em termos de oráculo espúrio violações: relatou apenas 1 atividade com perda de dados espúria apenas a cada 4 aplicativos testados, o que indica que o DLD é preciso e raramente irrita os testadores com alarmes falsos.

A **Figura 5 mostra a porcentagem de violações espúrias do oráculo retornadas por aplicativo. A porcentagem varia entre um mínimo de 0% e um max de 24%, com valor médio de 10,4% e valor mediano de 2,7%.**
De fato, o DLD produz uma porcentagem limitada de violações espúrias (menos de 11% para 75% dos aplicativos) que podem ser inspecionados de forma viável por engenheiros ao analisar a saída produzida pela técnica.
Os 5 outliers relatados na Figura 5 correspondem a aplicativos com elementos de difícil manuseio, como cronômetros e barras de progresso, que podem ser a fonte de um número anormal de violações espúrias devido às mudanças espontâneas que acontecem concomitantemente com o duplo rotações. Discutimos a origem dessas violações espúrias no RQ3.

4.6 RQ2 - O DLD é mais eficaz do que técnicas de última geração?

Esta questão de pesquisa compara o DLD ao ALARic [24] e Quântico [33]. ALARic representa o caso de uma abordagem alternativa automatizada para revelar falhas de perda de dados, enquanto Quantum representa o caso de uma abordagem que pode se beneficiar de um modelo para gerar casos de teste reveladores de perda de dados.

4.6.1 RQ2.1 - Qual é a eficácia relativa do DLD, ALARic, e quântico? A comparação com ALARic estuda a eficácia da estratégia de geração de teste definida no DLD, conforme descrito em Seção 3, para ALARic, que usa uma exploração aleatória (não tendenciosa) e uma representação de estados concretos.

Executamos o ALARic três vezes por 3 horas cada vez, conforme feito para DLD, e relatou os resultados na Tabela 2 (coluna ALARic). Observação que excluímos da comparação os aplicativos que foram testado com DLD explorando uma configuração ad-hoc, pois levaria a uma comparação injusta com o ALARic, que não implementa este característica. A **Tabela 2 mostra com fundo cinza os casos em que um técnica supera a outra.**

O DLD superou significativamente o ALARic em termos de perda de dados capacidade de descoberta. De fato, o ALARic revelou 71 atividades afetadas por falhas de perda de dados, enquanto o DLD revelou 189 atividades defeituosas, liberando um fator de melhoria de 2,7X. ALARic encontrou 19 falhas de perda de dado do benchmark, 8 falhas de perda de dados online e 50 novas perdas de dados falhas, panes. Enquanto o DLD revelou 73 das falhas de perda de dados no banco marca (fator de melhoria de 3,8X), 35 falhas de perda de dados online (4,4X fator de melhoria) e 132 novas perdas de dados (melhoria de 2,6X fator). No geral, o DLD revelou significativamente mais falhas de perda de dados do que ALARic.

O DLD teve um desempenho melhor do que o ALARic também em termos de perda de dados. Na verdade, o DLD produziu perda de dados espúria apenas para 11 atividades, enquanto ALARic produziu perda de dados espúria para 31 dos Atividades.

Em resumo, o DLD tem sido significativamente mais eficaz do que ALARic com os aplicativos estudados.

Como o Quantum não está disponível publicamente, não pudemos comparar Quantum para DLD em nosso conjunto de aplicativos. Assim, executamos DLD por 3 horas nos mesmos aplicativos usados na avaliação do Quantum [33] e comparou os resultados. Limitamos o experimento a 4 dos 6 aplicativos usados para avaliar o Quantum, pois para 2 aplicativos era impossível para recuperar a mesma versão utilizada no estudo original.

Como desconhecemos o esforço manual que foi necessário para definir manualmente os modelos usados pela Quantum, é difícil configurar um comparação justa entre as duas abordagens. No entanto, o obtido resultados ainda podem oferecer insights úteis sobre a eficácia relativa das duas abordagens.

Tabela 3: Comparação entre Quantum e DLD.

Aplicativo (versão)	Violação espúria de dados quânticos (com modelo manual)		DLD (automático)	
	perda		dados espúrios	violação de perda
OpenSudoku (1.1.5)	3	2	5	1
Gerenciador de Nexes (2.1.8)	7	2	11	1
VuDroid (1.4)	2	0	2	0
K9 Mail (4.317)	4	1	15	2

A **Tabela 3 mostra a perda de dados distinta e violações espúrias reportado por Quantum e DLD. Embora o DLD não possa se beneficiar de modelo manual, sua atividade tem sido mais efetiva em revelar uma série de falhas de perda de dados em comparação com o Quantum. Claro, nós**

não sei se as falhas de perda de dados reveladas pelo DLD incluem todas as falhas de perda de dados reveladas pela Quantum. Pode ser que o DLD não consiga alcançar algumas áreas do aplicativo em teste que podem ser alcançadas com o modelo manual. No entanto, os resultados sugerem que o DLD é bastante eficaz mesmo em comparação com técnicas que exploram modelos manuais.

4.6.2 RQ2.2-Quais são os principais fatores que determinam a eficácia do DLD?. Para investigar o motivo da diferença no desempenho entre DLD e ALARic, estudamos o motivo pelo qual o ALARic não revelou falhas reveladas pelo DLD. Em particular, contamos o número de atividades defeituosas não alcançadas pelo ALARic e o número de atividades defeituosas alcançadas pelo ALARic sem revelar a falha de perda de dados. Isso produziu os seguintes resultados:

- O ALARic não atinge 52% das atividades defeituosas reveladas apenas pelo DLD, ou seja, aproximadamente metade das falhas de perda de dados adicionais reveladas pelo DLD se devem a uma melhor estratégia de exploração,
- O ALARic atinge a atividade defeituosa sem revelar a perda de dados para 48% das atividades defeituosas reveladas apenas pelo DLD. Em poucas palavras, o teste sistemático refinado dos estados implementados no DLD é mais eficaz do que a estratégia de Alaric,
- 12 das atividades defeituosas perdidas pelo Alaric exigem que um oráculo baseado em instantâneo seja revelado, mas apenas 4 delas são alcançadas pelo ALARic.

4.7 RQ3 - Qual é a compensação entre os oráculos baseados em instantâneos e propriedades?

Esta questão de pesquisa investiga a complementaridade entre os oráculos baseados em instantâneos e os oráculos baseados em propriedades. Já discutimos na Seção 3.3 as diferenças qualitativas entre esses dois tipos de oráculos e relatamos exemplos de falhas de perda de dados que poderiam ser detectadas por apenas um tipo de oráculo. Aqui, avaliamos quantitativamente o impacto de cada classe de oráculos, tanto nas falhas de perda de dados reveladas quanto nas violações espúrias dos oráculos.

A Figura 6 mostra a porcentagem de falhas de perda de dados detectadas e o número de violações espúrias de oráculo produzidas por apenas uma estratégia , seja baseada em instantâneo ou baseada em propriedade, ou ambas. No caso das violações espúrias, temos uma categoria adicional que são as violações causadas pela recreação de atividade lenta após a rotação da tela, conforme antecipado na seção que discute o RQ1.

Nenhuma das duas abordagens foi capaz de revelar todos os problemas de perda de dados. Uma grande parte das falhas (73,1%) foram detectadas por ambos os oráculos, o que implica que a maioria das falhas de perda de dados causa tanto propriedades que perdem seus valores quanto problemas visíveis no aplicativo. No entanto, ainda existem 26,9% das falhas que exigem que um tipo específico de oráculo seja revelado.

Em termos de capacidade absoluta de descoberta de falhas, ambos os oráculos foram eficazes, com os oráculos baseados em propriedades e baseados em instantâneos revelando 90,9% e 82,3% das falhas, respectivamente.

Um pequeno número de violações espúrias de oráculos (5,3%) é causado por uma recreação de atividade lenta, que faz com que os oráculos recuperem informações de estado incorretas. Essa porcentagem pode ser reduzida ou eliminada ajustando cuidadosamente o tempo dos oráculos.

Curiosamente, o oráculo baseado em propriedade também é mais eficaz em termos de violações espúrias relatadas. De fato, apenas 0,1% das violações espúrias são produzidas exclusivamente pela propriedade oráculo, enquanto 21,1% das violações espúrias são produzidas exclusivamente pelo oráculo baseado em instantâneo. A maior proporção das violações espúrias (73,6%) é produzida por ambas as estratégias.

Embora o oráculo baseado em instantâneo produza mais violações espúrias do que o oráculo baseado em propriedade, essas violações raramente fazem com que as atividades corretas sejam relatadas ao testador (1 atividade a cada 4 aplicativos), portanto, é relativamente prejudicial usá-lo no p Pelo contrário, incluí-lo na análise aumenta o número de falhas de perda de dados reveladas em 9,2%, o que é um aumento não trivial da capacidade de descoberta de falhas do DLD.

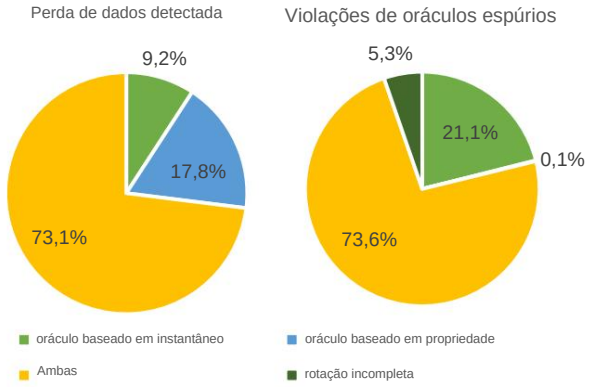


Figura 6: Porcentagem de falhas de perda de dados detectadas e violações espúrias por estratégia da Oracle.

4.8 RQ4 - As falhas de perda de dados são relevantes para os desenvolvedores?

Por fim, investigamos se as falhas de perda de dados são relevantes para os desenvolvedores de aplicativos . Para isso, para cada aplicativo do benchmark, identificamos e baixamos a versão mais recente do mesmo aplicativo. Executamos novamente o DLD por 9 horas (três execuções de 3 horas) em cada aplicativo e revelamos 195 falhas de perda de dados que ainda afetam esses aplicativos hoje em dia. Finalmente , enviamos um relatório de bug online para cada perda de da

Até o fim do prazo da conferência, recebemos feedback de 98 dos relatórios enviados online. Os desenvolvedores confirmaram os bugs para 88 relatórios (90% dos relatórios com feedback). Apenas em 10 casos os desenvolvedores rejeitaram o relatório defendendo que a falha era uma falha de estrutura, alegando que a falha não era reproduzível ou não dando explicação. Podemos, assim, concluir que as falhas de perda de dados reveladas são significativas para os desenvolvedores.

Em 33 dos 88 casos, os desenvolvedores alegaram que o custo de correção desses bugs pode ser muito alto em comparação com o impacto no aplicativo. É claro que essa decisão depende da consequência específica da perda de dados e da complexidade da atividade afetada pela falha. Isso também sugere que a definição de uma estratégia de reparo automático [15] que possa resolver falhas de perda de dados pode ser extremamente benéfica para melhorar a relação custo-benefício do processo de correção de bugs.

4.9 Ameaças à validade

As principais ameaças internas à validade do nosso estudo são o trabalho manual realizado para identificar as violações espúrias entre as falhas de perda de dados relatadas na avaliação. Distinguindo um dado genuíno

a perda de um espúrio é, no entanto, bastante simples, como também confirmado pelos relatórios de bugs enviados on-line a desenvolvedores que foram quase todos aceitos, com rejeições devido a falhas consideradas fora do limite do aplicativo testado ou comportamentos que poderiam ser considerados aceitável, embora não ideal.

As principais ameaças externas à validade dizem respeito à generalização dos resultados. O número significativo de falhas e aplicativos considerados mitiga essa ameaça. O fato de termos repetido a avaliação com as versões mais recentes dos aplicativos revelando novamente muitas falhas de perda de dados é outro fator de mitigação.

Com relação à comparação entre DLD e ALARic, a consistência dos resultados em todos os aplicativos testados é um forte fator a favor da generalidade dos resultados. Os resultados da comparação entre DLD e Quantum não podem ser generalizados devido ao tamanho limitado e à configuração do experimento, mas ainda fornecem informações úteis sobre a eficácia do DLD.

5 TRABALHOS RELACIONADOS

Várias técnicas abrangendo uma ampla gama de abordagens estão disponíveis para gerar casos de teste para aplicativos Android. Por exemplo, o Monkey gera entradas de teste de forma totalmente aleatória sem interpretar a GUI do aplicativo em teste [12]. A3E gera sistematicamente entradas de teste seguindo uma estratégia de profundidade em primeiro lugar [3]. DroidBot [19] e Stoat [31] constroem um modelo baseado em estado do sistema em teste e geram casos de teste explorando informações sobre os eventos já testados nos estados visitados. Sapienz usa algoritmos evolucionários para gerar casos de teste [20]. Embora essas abordagens tenham revelado várias falhas interessantes em aplicações de código aberto [6] e industriais [32], elas são ineficazes contra problemas de perda de dados (e também contra a maioria das falhas sem travamento [33]). Na verdade, eles não incluem operações que causam eventos de parada e partida nem são equipados com oráculos que podem detectar falhas de perda de dados sem falhas, que são responsáveis pela maioria das falhas relatadas em nossa avaliação.

Thor [1] pode aumentar os conjuntos de testes existentes com sequências neutras de operações para revelar falhas adicionais. As sequências injetadas dizem respeito ao serviço de áudio, à conectividade e ao ciclo de vida das atividades, que podem revelar falhas de perda de dados. Da mesma forma, quando um modelo gerado pelo usuário do aplicativo em teste está disponível, Quan tum [33] pode gerar testes que podem revelar perda de dados, conforme relatado em uma avaliação em pequena escala. Diferentemente dessas abordagens, o DLD gera diretamente os casos de teste e não requer uma suíte de teste inicial nem um modelo do aplicativo em teste, mantendo uma alta eficácia conforme demonstrado na comparação com ALARic e Quantum.

CrashScope [22] e AppDoctor [17] podem gerar testes que podem revelar problemas de perda de dados, mas sua eficácia é limitada a falhas de travamento, que representam a minoria dos casos.

ALARic [2] é uma técnica de geração de casos de teste principalmente aleatória que, semelhante ao DLD, explora rotações de tela dupla para revelar falhas de perda de dados. No entanto, a estratégia de exploração tendenciosa, a abstração do estado de habilitação e as ações ad-hoc de revelação de perda de dados usadas pelo DLD superaram o ALARic em nossa avaliação.

Quando o código fonte do aplicativo está disponível, uma técnica de análise estática como KREfinder [28] pode ser usada para revelar problemas de perda de dados. No entanto, como a maioria das técnicas de análise estática, o KREfinder sofre problemas de escalabilidade e provavelmente relatará muitas violações espúrias . Pelo contrário, a eficácia do DLD não depende na complexidade e tamanho do código-fonte e raramente relata perda de dados espúria.

As estratégias oraculares apresentadas neste artigo relacionam-se ao trabalho sobre testes metamórficos [27]. Testes metamórficos exploram relações metamórficas, que são relações em múltiplas execuções do software, para verificar a correção do comportamento observado. As sequências neutras de operações que o DLD usa para revelar problemas de perda de dados podem ser vistas como uma classe específica de relações metamórficas que relacionam execuções com e sem es

As relações entre execuções, como as usadas neste artigo para revelar problemas de perda de dados, também têm sido usadas para curar execuções, por exemplo, para produzir soluções alternativas automáticas [5]. Embora sequências neutras de eventos possam ser potencialmente usadas para obter soluções alternativas, as usadas neste artigo dificilmente podem ser usadas para curar execuções , pois muitas vezes são a causa de falhas, conform

Finalmente, falhas em aplicativos Android, incluindo falhas de perda de dados, podem ser tratadas com técnicas de recuperação. No entanto, poucas abordagens de cura podem funcionar no ambiente Android. Azim et ai. definiram uma técnica que pode desabilitar funcionalidades que não estão funcionando corretamente [4]. Embora essa abordagem possa ser explorada para evitar falhas de perda de dados, ela também reduz o conjunto de funcionalidades disponíveis para os usuários. DataLossHealer é uma solução de cura projetada para mitigar o impacto de falhas de perda de dados em campo [25]. Embora possa evitar algumas falhas de perda de dados, tem várias desvantagens. Por exemplo, ele introduz sobrecarga para salvar e restaurar dados na presença de falhas de perda de dados, pode resolver apenas algumas perdas de dados e requer o enraizamento do dispositivo. O DLD oferece uma solução mais eficaz, revelando falhas de perda de dados ant

6. CONCLUSÕES

Os aplicativos Android devem ser projetados para lidar com eventos stop-start, que são eventos externos que podem interromper a execução da atividade em execução. Quando um desses eventos é gerado, a atividade do Android em primeiro plano pode ser destruída e recriada posteriormente. Para evitar a perda de dados úteis durante esse processo, os aplicativos devem implementar explicitamente a lógica necessária para salvar os dados, quando a atividade for destruída, e restaurar os dados salvos, quando a atividade fo Infelizmente, esta lógica é muitas vezes defeituosa [1, 2, 24, 26].

Este artigo apresenta o Data Loss Detector (DLD), uma técnica de geração automática de casos de teste projetada para revelar falhas de perda de dados. O DLD explora uma estratégia de exploração voltada para a descoberta de novos estados de aplicativos, ações de revelação de perda de dados e duas estratégias dedicadas baseadas em oráculo para revelar automaticamente problemas de perda de dados. Em nossa avaliação com 110 falhas de perda de dados afetando 54 lançamentos de aplicativos , o DLD superou o ALARic [24] e teve um bom desempenho em comparação ao Quantum quando instruído com um modelo manual do aplicativo em teste. No geral, o DLD revelou 298 atividades afetadas por falhas de perda de dados, o que é um indicador claro da eficácia da abordagem e da abrangência do problema.

Estamos agora trabalhando na definição de soluções de reparo automático de programas para falhas de perda de dados.

AGRADECIMENTOS

Gostaríamos de agradecer a Vincenzo Riccio, Domenico Amalfitano e Anna Rita Fasolino por compartilharem conosco a implementação do ALARic .

REFERÊNCIAS

[1] Christoffer Quist Adamsen, Gianluca Mezzetti e Anders Mùller. 2015. Execução Sistemática de Suítes de Testes Android em Condições Adversas. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA).

[2] Domenico Amalfitano, Vincenzo Riccio, Ana CR Paiva, and Anna Rita Fasolino. 2018. Por que a mudança de orientação atrapalha meu aplicativo Android? De falhas de GUI a falhas de código. Teste, verificação e confiabilidade de software 28, 1 (2018), e1654.

[3] Md. Tanzirul Azim e Iulian Neamtiu. 2013. Exploração direcionada e em profundidade para testes sistemáticos de aplicativos Android. In Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Aplicativos (OOPSLA).

[4] Md. Tanzirul Azim, Iulian Neamtiu e Lisa M. Marvel. 2014. Rumo ao software de smartphone de autocura por meio de patches automatizados. In Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE).

[5] Antonio Carzaniga, Alessandra Gorla, Nicolò Perino e Mauro Pezzè. 2015. Soluções alternativas automáticas: explorando a redundância intrínseca de aplicativos da Web . Transações ACM em Engenharia de Software e Metodologias (TOSEM) 24, 3 (2015).

[6] Shauvik Roy Choudhary, Alessandra Gorla e Alessandro Orso. 2015. Geração de entrada de teste automatizada para Android: já chegamos? (E). In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE).

[7] Guido De Caso, Víctor Braberman, Diego Garbervetsky e Sebastián Uchitel. 2011. Programa Abstrações para Validação de Comportamento. In Proceedings of the International Conference on Software Engineering (ICSE).

[8] Guido De Caso, Victor Braberman, Diego Garbervetsky e Sebastian Uchitel. 2013. Abstrações de programa baseadas em habilitação para validação de comportamento. ACM Trans. Softw. Eng. Metodologia. 22, 3 (2013), 25:1 × 25:46.

[9] Desenvolvedores Android. [sd]. Fragmentos. https://developer.android.com/guide/componentes/fragmentos [10] Desenvolvedores Android. [sd]. Salvando estados da interface do usuário. https://developer.android.com/topic/libraries/architecture/saving-states.html

[11] Desenvolvedores Android. [sd]. Visão geral dos Serviços. https://developer.android.com/guia/componentes/serviços

[12] Desenvolvedores Android. [sd]. Macaco de Exercitador de UI/Aplicativo. https://desenvolvedor.android.com/studio/test/monkey [13] Desenvolvedores Android. [sd]. Entenda o Ciclo de Vida da Atividade. https://desenvolvedor. android.com/guide/components/activities/activity-lifecycle

[14] Fundação JS. [sd]. Appium. http://appium.io/ [15] Luca Gazzola, Daniela Micucci e Leonardo Mariani. 2019. Reparo Automático de Software: Uma Pesquisa. Transações IEEE em Engenharia de Software (TSE) 45, 1 (2019), 34×67.

[16] Cuixiong Hu e Iulian Neamtiu. 2011. Automatizando o teste de GUI para aplicativos Android. In Proceedings of the 6th International Workshop on Automation of Software Test (AST).

[17] Gang Hu, Xinhao Yuan, Yang Tang e Junfeng Yang. 2014. Detectando bugs de aplicativos móveis com eficiência e eficácia com o AppDoctor. In Actas da Nona Conferência Europeia de Sistemas Informáticos (EuroSys).

[18] Ajay Kumar Jha, Sunghee Lee e Woo Jin Lee. 2019. Caracterizando bugs de falha específicos do Android. In Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft).

[19] Yuanchun Li, Yang Ziyue, Guo Yao e Chen Xiangqun. 2017. DroidBot: Um gerador de entrada de teste guiado por interface do usuário leve para Android. In Proceedings of the International Conference on Software Engineering Companion (ICSE).

[20] Ke Mao, Mark Harman e Yue Jia. 2016. Sapienz: Testes Automatizados Multiobjetivos para Aplicativos Android. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA).

[21] Mobilloud. 2019. As pessoas gastaram 90% do seu tempo móvel usando aplicativos em 2019. https://www.mobiloud.com/blog/mobile-apps-vs-the-mobile-web/. [Conectados; acessado em janeiro de 2020].

[22] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome e D. Poshy vanyk. 2016. Automaticamente descobrindo, relatando e reproduzindo falhas de aplicativos Android. In Proceedings of the International Conference on Software Testing, Verification and Validation (ICST).

[23] RescueTime:blog. 2019. Estatísticas de tempo de tela 2019: veja quanto você usa seu telefone durante o dia de trabalho. https://blog.rescuetime.com/screen-time-stats-2018/. [Conectados; acessado em janeiro de 2020].

[24] Vincenzo Riccio, Domenico Amalfitano e Anna Rita Fasolino. 2018. Esse é o ciclo de vida que realmente queremos?: uma abordagem automatizada de teste de caixa preta para atividades do Android. In Proceedings of the International Workshop on User Interface Test Automation, e Workshop on TESting Techniques for event BasED Software (ISSTA/ECOOPWorkshops).

[25] Oliviero Riganelli, Daniela Micucci e Leonardo Mariani. 2016. Resolvendo problemas de perda de dados em aplicativos Android. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).

[26] Oliviero Riganelli, Marco Mobilio, Daniela Micucci e Leonardo Mariani. 2019. Uma referência de bugs de perda de dados para aplicativos Android. In Proceedings of the International Conference on Mining Software Repositories (MSR).

[27] Sergio Segura, Gordon Fraser, Ana B. Sanchez e Antonio Ruiz-Cortes. 2016. Uma Pesquisa sobre Testes Metamórficos. Transações IEEE em Engenharia de Software (TSE) 42, 9 (2016), 805×824.

[28] Z. Shan, T. Azim e I Neamtiu. 2016. Encontrando Erros de Retomar e Reinicie em Aplicativos Android. In Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA).

[29] Contador de estatísticas. 2020. Participação no mercado mundial de sistemas operacionais móveis - dezembro de 2019. http://gs.statcounter.com/os-market-share/mobile/worldwide. [Conectados; acessado em janeiro de 2020].

[30] Estatista. 2020. Número de aplicativos disponíveis na Google Play Store de dezembro de 2009 a dezembro de 2019. https://www.statista.com/statistics/266210/ number-of-available-applications-in-the-google-play-store/ [31] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu e Zhendong Su. 2017. Teste de GUI baseado em modelo estocástico guiado de aplicativos Android. Em Anais da Reunião Conjunta sobre Fundamentos da Engenharia de Software (FSE).

[32] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng e Tao Xie. 2018. Um estudo empírico de ferramentas de geração de testes Android em casos industriais. In Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE).

[33] Razieh Nokhbeh Zaeem, Mukul R. Prasad e Sarfraz Khurshid. 2014. Geração Automatizada de Oracles para Teste de Recursos de Interação com o Usuário de Aplicativos Móveis. In Proceedings of the International Conference on Software Testing, Verification and Validation (ICST).