Lucas Bezerra Perrusi

AUTOTESTCOVERAGE$_P$: UMA FERRAMENTA PARA COBERTURA DE TESTES DE INTEGRAÇÃO NO CONTEXTO ANDROID SEM USO DE CÓDIGO-FONTE

Trabalho de Graduação

Universidade Federal de Pernambuco
graduacao@cin.ufpe.br
www.cin.ufpe.br/~graduacao

RECIFE
2018

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

Lucas Bezerra Perrusi

AUTOTESTCOVERAGE$_P$: UMA FERRAMENTA PARA COBERTURA DE TESTES DE INTEGRAÇÃO NO CONTEXTO ANDROID SEM USO DE CÓDIGO-FONTE

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Alexandre Cabral Mota*

RECIFE
2018

Lucas Bezerra Perrusi

AutoTestCoverage$_P$: Uma ferramenta para cobertura de testes de integração no contexto Android sem uso de código-fonte/ Lucas Bezerra Perrusi. – RECIFE, 2018-

39 p. : il. (algumas color.) ; 30 cm.

Orientador Alexandre Cabral Mota

Trabalho de Graduação – Universidade Federal de Pernambuco, 2018.

1. Cobertura de código. 2. Teste de Regressão. I. Alexandre Mota. II. Universidade Federal de Pernambuco. III. Centro de Informática. IV. AutoTestCoveragep: Uma ferramenta para cobertura de testes de integração no contexto Android sem uso de código-fonte

CDU 02:141:005.7

Trabalho de graduação apresentado por **Lucas Bezerra Perrusi** ao programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **AutoTestCoverage$_P$: Uma ferramenta para cobertura de testes de integração no contexto Android sem uso de código-fonte**, orientada pelo **Prof. Alexandre Cabral Mota** e aprovada pela banca examinadora formada pelos professores:

Prof. Alexandre Cabral Mota
Centro de Informática/UFPE

Prof. Juliano Manabu Iyoda
Centro de Informática/UFPE

RECIFE
2018

Eu dedico este trabalho à minha familia.

Agradecimentos

Primeiramente gostaria de agradecer a minha família, Minha Mãe, Meu pai e minhas irmã, por sempre me apoiarem e estarem ao meu lado por todo o tempo. Gostaria de agradecer também a todas as pessoas que de alguma forma se envolveram neste trabalho, e que me ajudaram direta ou indiretamente. Gostaria também de agradecer as pessoas do convênio CIn/Motorola, por sempre estarem dispostas a ajudar e contribuir de alguma forma. Em especial gostaria de agradecer Alice Arashiro, Viviana Toledo, Eliot Maia, Bazante, Guilherme e Virgínia. Gostaria também de agradecer a todos os meus professores por me passarem todo o conhecimento que puderam, e pela disponibilidade e vontade de ensinar. Em especial Alexandre Mota, Leopoldo Teixeira , Augusto Sampaio, Juliano Iyoda. Por fim, gostaria de agradecer à todos os meus amigos.

Resumo

Análise de cobertura, quer seja de código, de requisitos ou de mudanças, é usada para avaliar qualidade dos testes. No contexto de testes de regressão, o objetivo é verificar se o que funcionava em uma versão anterior do sistema continua funcionando na versão atual. Para tal, pode-se simplesmente re-executar todos os testes existentes, mas se torna ideal uma análise de cobertura para garantir que os testes estão executando, pelo menos uma vez, as áreas que foram modificadas. No caso de nosso parceiro industrial não é possível ter acesso irrestrito ao código-fonte do sistema. Para solucionar isto propomos uma análise das interações entre aplicativos do sistema, que chamaremos de teste de regressão de plataforma, usando recuperação de informação, e assim poder fornecer a cobertura para um plano de teste, ou seja, fornecer uma noção de qualidade dos casos de testes tendo como base as ações realizadas pelo testador em relação as modificações feitas no código fonte do sistema.

Palavras-chave: Cobertura de Código, Teste de Regressão, Recuperação de Informação

Abstract

Coverage analysis, whether of code, requirements or changes, is used to evaluate the quality of the tests. In the context of regression tests, the objective is to verify that what worked in an earlier version of the system continues to work in the current version. To do so, one can simply re-run all existing tests, but a coverage analysis becomes ideal to ensure that the tests are covering, at least once, the areas that have been modified. In the case of our industrial partner it is not possible to have unrestricted access to the system source code. To solve this, we propose an analysis of the interactions between system applications, which we will call a platform regression test, using information retrieval, and thus provide the coverage for a test plan, that is, to provide a notion of quality of the tests based on the actions performed by the tester in relation to the modifications made to the system's source code.

Keywords: Code Coverage, Regression Testing, Information Retrieval

Lista de Figuras

3.1 Fluxo completo ATC 27
3.2 Interface ATC 29
3.3 Processos TBC 30
3.4 Exemplo resultado TBC 31
3.5 Processo Monitoramento Dispositivo Android 31

Lista de Tabelas

4.1 Experimento: 1, 2 e 3 34
4.2 Experimento: 4 34

Sumário

1 Introdução **19**

1.1 Definição do Problema 19

1.2 Proposta Geral 20

1.3 Objetivos 20

1.4 Estrutura do Documento 21

2 Fundamentação Teórica **23**

2.1 Teste de Regressão 23

2.2 Teste de Integração 24

2.3 Métricas de Cobertura 24

2.4 Recuperação de Informação 25

3 Desenvolvimento **27**

3.1 Recuperação de Informação 27

3.2 Captura dos Dados de Cobertura 28

3.3 Cruzamento das Informações 28

3.4 Protótipo da Ferramenta de Cobertura de Código 29

3.4.1 Módulo 1 - ToBeCovered 29

3.4.2 Módulo 2 - Captura dos Dados do Dispositivo Android 30

4 Experimentos **33**

4.1 Execução 33

4.1.1 Discussão Geral 35

4.2 Ameaças de Validação 35

5 Conclusão **37**

5.1 Trabalhos Relacionados 37

5.2 Trabalhos Futuros 38

Referências **39**

1

Introdução

1.1 Definição do Problema

Um sistema é construído em fases: requisitos, especificação, projeto, implementação e testes. Nesta fase, objetiva-se verificar se o sistema projetado está funcionando da maneira esperada. Mas esta fase não é isolada das demais. Ou seja, ao invés dela ocorrer em apenas um momento no processo de desenvolvimento do sistema, ela ocorre durante todas as outras fases do desenvolvimento.

Durante a fase de testes existem diferentes técnicas que podem ser aplicadas em diferentes momentos e de diferentes formas para avaliar o funcionamento do *software*. Dentre esses diferentes tipos de testes os principais são os testes de caixa preta onde o foco dos testes são aspectos externos do sistema, ou seja, os requisitos funcionais do sistema são analizados, os testes de caixa branca, diferente dos testes de caixa preta, onde os aspector externos são analizados, os aspectos internos, o código fonte, são o foco, os testes que unem as duas abordagens citadas anteriormente, são chamados de testes de caixa cinza. Outro tipo de de testes são os testes de regressão, que são realizados quando uma modificação é feita no sistema, os testes de usabilidade são feitos com um pequeno grupo de usuários para validar se os mesmos se satisfazem com o *software*. Outra técninca usada no processo de testes são os testes de stress, que tem como objetivo verificar os limites de potência e funcionamento, para avaliar em qual ponto o sistema passa a não funcionar adequadamente.

Para nosso trabalho iremos focar em testes de regressão, onde o objetivao é verificar se o que funcionava em uma versão anterior do sistema continua se apresentando da forma esperada na versão atual. Para tal, pode-se simplesmente re-executar conjunto de todos os testes existentes. Só que isto é inviável na prática e alguma forma de selecionar testes faz-se necessária. Como forma de selecionar testes para as áreas que foram modificadas, torna-se usual realizar alguma forma de análise de cobertura, quer seja de código, de requisitos, de mudanças.

Quando é possível ter acesso ao código do sistema e realizar a instrumentação na parte modificada, ou seja, introduzir pontos de verificação em partes específicas do código com o objetivo de obter informações sobre a execução do programa, torna-se mais fácil e

preciso medir o quanto um caso de teste exercitou a região desejada. No entanto, há várias situações reais (em empresas grandes, onde os processos são divididos em equipes fisicamente e culturalmente distantes) em que não é permitida a instrumentação usual do código para efeito de obter informações de cobertura. Neste trabalho nosso foco é exatamente neste contexto, pois estamos inseridos no convênio de pesquisa entre o Centro de Informática e a Motorola Mobility, onde não é possível ter acesso irrestrito ao código-fonte do sistema. Além disto, nossa principal tarefa é monitorar as interações entre aplicativos do sistema Android e não aplicativos específicos. Desta forma, neste contexto denomina-se por teste de regressão de plataforma.

1.2 Proposta Geral

Nossa proposta é capturar as modificações realizadas em uma versão do sistema Android a partir das *release notes* do sistema e capturar as ações realizadas durante a execução dos casos de teste, e com isso calcular a cobertura em relação à região modificada no código. Para tal, iremos utilizar uma ferramenta chamada AutoTestCoverage$_P$, que consiste de dois módulos:

1. O primeiro é baseado em recuperação de informação presente nas *release notes* para se definir quais regiões do sistema deverão ser exercitadas durante a execução dos casos de teste. Neste trabalho chamaremos esta região de ToBeCovered (TBC);

2. O segundo processo consiste em capturar as informações do dispositivo Android durante a execução dos testes, o que chamaremos neste trabalho de Trace. Para isto utilizaremos o Android Debug Bridge[1] (ADB) para coletar logs emitidos pelo código executado.

Para atingir o objetivo final deste trabalho é feito um cruzamento de informações entre o que foi coletado durante o primeiro processo, o TBC, e os dados coletados durante a segunda parte do processo, o Trace, e desta forma teremos o cálculo da cobertura para um plano de testes.

1.3 Objetivos

Nosso objetivo principal é avaliar a qualidade dos testes de regressão a nível de sistema em termos de cobertura de código. Ou seja, avaliar o alcance (regiões que exercitam) dos testes que tem como foco a interação entre aplicações após alterações nas mesmas.

Tanto iremos estudar a cobertura de planos específicos, testes selecionados em função de regiões modificadas, quanto de bases de testes quase completas, chamadas de *master plans*.

[1] Android Debug Bridge - https://developer.android.com/studio/command-line/adb

1.4 Estrutura do Documento

Este trabalho está organizado da seguinte maneira:

- **Capítulo 2** introduz brevemente os principais conceitos usados neste trabalho. Cobertura de código, testes de regressão e recuperação de informação são os mais importantes para este trabalho.
- **Capítulo 3** apresenta a solução proposta para a solução do problema de cobertura em testes de plataforma em dispositivos Android.
- **Capítulo 4** explica sobre os experimentos que foram executados. Serão apresentados os resultados e a análise deles.
- **Capítulo 5** discute nossas conclusões, trabalhos relacionados e futuros.

2

Fundamentação Teórica

Este capítulo apresenta conceitos básicos relacionados à cobertura de código, teste de regressão, fase de testes de integração e recuperação de informação.

2.1 Teste de Regressão

Testes de regressão é uma técnica na área de testes de software de grande importância usada durante todas as fases do desenvolvimento e manutenção do software. Os testes de regressão são usados quando um programa sofre modificações, com o maior foco quando ocorrem alterações no código existente ou quando são adicionadas novas funções ao programa. Durante a fase de desenvolvimento, os testes de regressão são geralmente aplicados em um programa testado em que um erro foi detectado e corrigido. Durante a fase de manutenção, modificações com o obejetivo de adaptar ou melhorar a aplicação, e corrigir falhas, podem alterar a lógica do sistema os testes de regressão são geralmente aplicados após essas modificações.

O principal objetivo de um teste de regressão após uma modificação no código, a adição de uma nova funcionalidade ou o concerto de uma falha relatada em versões anteriores, é verificar a corretude da nova lógica acrescentada ao programa, verificar se o funcionamento do código não modificado não foi afetado e validar se as partes do código que foram modificadas estão funcionando de acordo com as especificações.

Portanto, os testes de regressão podem abranger todas as funcionalidades do sistema, apenas para as funcionalidades que foram modificadas ou podem ser novos casos de teste para as funcionalidades que provavelmente foram afetadas pela mudança. Quando um novo modo da camera do dispositivo é implementado ou quando um botão para uma função que ja estava funcionando da forma esperada é modificado, testes de regressão são utilizados para verificar se essas modificações não alteraram o funcionamento do sistema.

2.2 Teste de Integração

Dentre as principais fases de teste de software estão presentes a fase dos testes unitários, testes de integração, testes de sistema e por fim os testes de aceitação.

Para a fase de testes unitários cada unidade do programa é testada individualmente para garantir que elas estão funcionando conforme especificado. Em seguida é realizada a fase de testes de integração onde todas as unidades que compõem o sistema são testadas de forma integrada para garantir que todas os componentes estão interagindo de forma correta. A fase de testes de sistema visa testar todas as funcionalidades do sistema e a integração com outros possíveis sistemas. Por fim são realizados testes de aceitação, onde são testadas funcionalidades requeridas pelo cliente.

Testes de integração têm como objetivo verificar se os componentes do sistema, juntos, trabalham conforme descrito nas especificações do sistema e do projeto do programa, segundo Pfleeger (2004). Portanto neste trabalho iremos focar na fase de testes de integração, visto que para nosso contexto são realizados testes entre aplicações em dispositivos Android.

Para esta fase de testes é preciso definir quais componentes são o foco dos testes, em qual sequência as interfaces dos componentes serão testadas e qual técnica sera aplicada para exercitar essa interface. No contexto do nosso trabalho, os testes realizados têm como foco aplicações em um sistema móvel Android, onde são executados de forma manual. Portanto a interface dos componentes é acionada através da interação do testador com a tela do disposivo.

2.3 Métricas de Cobertura

As métricas de cobertura são fundamentais na fase de testes, porque com elas se torna possível definir o quão completo o teste é, baseado nos requisitos em que ele foi criado, para cobertura de requisitos, ou baseado na implementação do código, para cobertura de código-fonte. Para este trabalho, dado que nosso foco é realizar cobertura com para testes de regressão, iremos basear a nossa cobertura em relação ao código-fonte.

A cobertura em código-fonte tem como objetivo medir a quantidade de código executada durante o teste em comparação à quantidade de códigos com execução pendente. Este tipo de cobertura se baseia em duas principais medidas: em fluxos de controle ou fluxos de dados. A cobertura baseada em fluxo de controle tem como objetivo cobrir linhas de código, condições de ramificação, caminhos que percorrem o código ou outros elementos do fluxo de controle do software. Na cobertura de fluxo de dados, o objetivo é cobrir se os estados dos dados permanecem válidos durante as operações do sistema.

Para este trabalho iremos focar na cobertura de código para fluxo de controle, visto que, para o nosso cenário, não possuímos acesso total ao código-fonte das aplicações do sistema Android. Como não é possivel verificar o fluxo dos dados do sistema, teremos como foco verificar o fluxo percorrido pelo código-fonte durante a execução dos casos de teste.

Para definirmos com precisão nossa métrica, tomaremos que cobertura de teste baseada em códigos é calculada pela seguinte equação:

$$CoberturaDeTeste = Ie/TIcm \tag{2.1}$$

Em que: Ie é o número de itens executados, expresso como instruções de código, ramificações de código, caminhos de código, pontos de decisão do estado de dados ou nomes de elementos de dados. Ticm é o número total de itens no código que foram módificados na versão do sistema que esta sendo testado.

REDEFINIR TIic

Assim podemos obter uma percentagem em relação aos casos de teste, de quanto do código-fonte foi exercitado.

Em relação a testes de regressão, o objetivo, como foi mostrado na seção anterior, é avaliar se o código inserido ou modificado alterou o funcionamento das funcionalidades testadas em versões anteriores. A cobertura de código-fonte se torna uma métrica fundamental para avaliar se o teste realizado está de fato exercitando as áreas do código relevantes para aquela versão do sistema. Assim, é possível determinar, usando cobertura de código-fonte, se todos os critérios para aquela suíte de testes foram alcançados e o quanto de esforço restante é necessário.

2.4 Recuperação de Informação

Recuperação de Informação (*Information Retrieval* - IR) é uma área que tem como objetivo lidar com o armazenamento automático e recuperação de dados, geralmente textos. IR geralmente consiste em 3 etapas: pré-processamento de texto, indexação e recuperação. O pré processamento consiste em normalização de texto, remoção de *stop word* e *stemming*. Um normalizador de texto remove pontuações, trata a fonte do texto (maiúscula e minúscula), coloca os termos em token, etc. No caso da fase de remoção da *stop word*, uma aplicação de IR descarta os termos muito frequentes tais como preposições, artigos, para aumentar a eficiência e reduzir as combinações irrelevantes. Finalmente, stemming combina variações de um mesmo termo para melhorar a combinação entre consultas e documentos.

No caso do ATC_P o pré processamento consiste apenas das duas primeiras parte citadas, a normalização e a remoção de *stop words*. Como trabalhamos com nomes de arquivos, não podemos procurar por variações dos nomes, visto que isso poderia causar falsos positivos. Feita a indexação dos nomes, então se inicia o processo de extração da informação, onde uma busca por determinadas palavras chaves é feita e assim a informação necessária é extraída. Por fim é realizado um processo de filtragem da informação para que seja feito o cálculo da cobertura. Esses processos serão melhor explicados nas próximas seções.

3

Desenvolvimento

A solução proposta para o cálculo da cobertura no contexto de dispositivos Android se subdivide em dois módulos, onde o primeiro foca em definir as regiões em que o código-fonte foi alterado e o segundo foca na captura das informações enquanto o teste é realizado. Esses dois módulos serão abordados com mais detalhes nos tópicos seguintes.

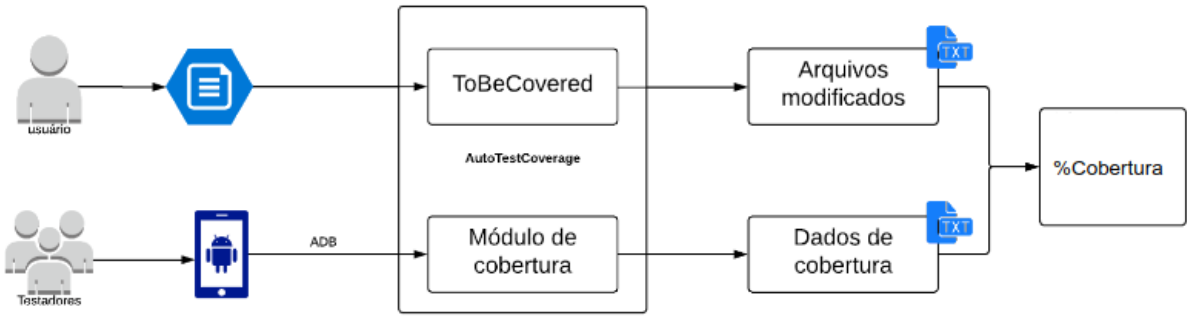

Figure 3.1: Fluxo completo ATC

3.1 Recuperação de Informação

Para esta etapa da nossa solução, desenvolvemos uma ferramenta que define quais regiões do código-fonte foram alteradas para determinada versão do sistema. Desta forma, é possível verificar com exatidão em quais áreas o teste de regressão deve focar.

Estas mudanças que ocorrem durante o desenvolvimento e manutenção do sistema são documentadas em Solicitações de Mudanças (*Change Requests* - CRs). No caso de CRs envolvendo falhas durante a execução de um caso de teste, nelas são armazenadas as falhas encontradas durante os testes, a região em que a falha ocorreu, as causas que geraram a falha durante o teste e a modificação feita no código para que a falha seja corrigida. Para CRs de desenvolvimento, onde novas funcionalidades são adicionadas ao sistema, ficam armazenadas as áreas em que foram feitas alterações e as regiões do código em que as funcionalidades foram adicionadas.

Este módulo da nossa ferramenta se chama ToBeCovered (TBC). Ele foca em recuperar informação das *release notes* usadas como base na seleção dos casos de teste. Para o nosso contexto, não possuímos acesso irrestrito ao código-fonte do sistema, porém através das *release notes*, é possível acessar as CRs relacionadas a uma determinada versão, coletar quais os arquivos foram modificados e consequentemente quais as funcionalidades do sistema devem ser testadas.

A extração da informação ocorre primeiramente através da recuperação de informação de quais funcionalidades foram alteradas em uma determinada nota de atualização e quais os repositórios do Gerrit foram alterados. Assim, é possível, utilizando a API do próprio Gerrit[1], extrair as informações dos repositórios, e após um processo de filtragem da resposta retornada pela API do repositório, é possível montar uma lista de quais foram os arquivos modificados.

3.2 Captura dos Dados de Cobertura

Para calcular a cobertura do plano de testes, em nosso contexto, é preciso que o fluxo do código exercitado durante a execução de um caso de teste manual seja monitorado.

Para que seja possível monitorar as ações realizadas no dispositivo móvel Android, desenvolvemos uma interface para ser usada pelo testador manual. No início da execução de um conjunto de casos de teste, o testador informa ao sistema um conjunto de informações, como o plano de testes que será executado e o dispositivo que será testado. Em seguida é iniciado o processo de captura, onde nossa ferramenta realiza esse processo utilizado o Logcat. Onde o Logcat é um comando, presente do Android SDK, que coleta todos os logs gerados pelas classes exercitadas no código Android, cada mensagem de log emitida pelo sistema contem informaões da classe. Desta forma, é possível monitorar as ações feitas pelo testador ao nível do fluxo dos arquivos exercitados, sem que a ATC_P realize instrumentação no código-fonte do sistema.

Após a realização dos casos de teste, o Trace, contendo o conjunto de classes exercitadas durante os passos realizados pelo testador manual, é extraído, e então é realizado um processo de filtragem pela ATC_P no arquivo de logs, onde são analisados os arquivos exercitados durante o caso de teste e as funcionalidades testadas durante o execução do testador manual. O resultado desse processo é o Trace dos testes realizados, ou seja, a ferramenta gera os dados de cobertura relacionados as ações realizadas pelo testador .

3.3 Cruzamento das Informações

Para gerar o resultado da cobertura do plano de testes, é feito o cruzamento da informação gerada pelo TBC e da informação do Trace. Primeiro é extraído do TBC a lista de arquivos modificados e definida a quantidade total de modificações para a versão do sistema testado. Em seguida é feito um processo de extração no arquivo de Trace utilizando a lista das modificações

[1] Gerrit - https://www.gerritcodereview.com

como parâmetro. Dessa forma temos como resultado a lista de arquivos modificados exercitados durante o plano de testes presentes na versão testada, ou seja, uma interseção entre as informações presentes no Trace e a lista de arquivos presentes no TBC. Para nosso trabalho a equação do cálculo da cobertura é difinida como:

$$Cobertura = \#(TraceFiles \cap TBCFiles)/\#TBCFiles \tag{3.1}$$

Onde temos como numerador da divisão a quantidade de arquivos presentes no conjunto composto pela interseção entre as informações presentes no Trace (TraceFiles) e a lista de arquivos presentes no TBC (TBCFiles). No denominador temos a quantidade de arquivos modificados na versão do software testado. Após realizar essa divisão temos como resultado o percentual de cobertura.

3.4 Protótipo da Ferramenta de Cobertura de Código

O processo descrito anteriormente foi implementado em uma ferramenta protótipo, que chamamos de AutoTestCoverage$_P$. Ela foi implementada usando Python e Django, um framework para a web em Python de alto nível, tendo uma arquitetura MVC (Model-ViewController). Este protótipo tem dois módulos separados.

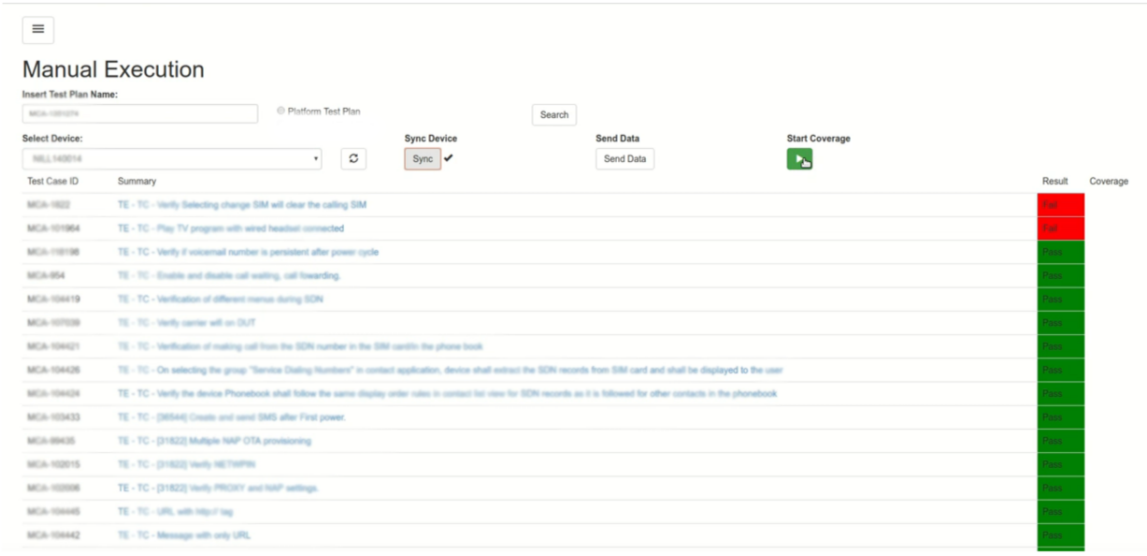

Figure 3.2: Interface ATC

3.4.1 Módulo 1 - ToBeCovered

Este módulo consiste em um servidor local, onde é realizado o processo de requisição das informações do respositório do sistema testado e a extração da informação dos arquivos modificados. Podemos dividir esses processos em 3 passos:

Figure 3.3: Processos TBC

- **Passo 1 (Extração das informações da nota de lançamento):**

  É passado para o servidor a versão do sistema a ser testada. Com essa informação, é feita uma extração da informação, como aplicações alteradas na nota de lançamento e o id do commit de cada modificação realizada. Com isso é possivel montar uma estrutura relacionando a versão do sistema, os componentes modificados e seus respectivos commits.

- **Passo 2 (Requisição ao repositório contendo as modificações da nota de lançamento):**

  Com a relação estabelecida entre cada aplicação modificada do sistema e seus respectivos commits, são realizadas pelo servidor da ATC$_P$, uma série de requisições ao respositório do sistema testado, onde temos um acesso de leitura as informações do commit. Cada requisição retorna informações como mensagens do commit, data das modificações, aplicação que sofreu a modificação e arquivos modificados. No Passo 3 essa informação é analisada e processada, (ver figura 3.3).

- **Passo 3 (Extração da informação dos arquivos modificados):**

  Nesta última fase do processo, após coletadas as informações dos commits presentes na nota de lançamento do sistema, é feita uma extração dos arquivos alterados, relacionando os mesmo a suas respectivas aplicações. Assim o servidor da ATC$_P$ retorna uma estrutura relacionando a versão do sistema informado pelo usuário que requisitou as informações, os componentes modificados presentes na *release notes* e seus respectivos arquivos alterados.

3.4.2 Módulo 2 - Captura dos Dados do Dispositivo Android

Esse módulo implementa o processo de captura dos dados de cobertura durante a execução dos casos de testes. Ou seja, nesta etapa é feito o monitoramento do dispositivo Android e então coletado quais arquivos estão sendo exercitados enquanto o testador realiza os passos presentes nos casos de teste. Este módulo se divide em duas etapas:

- **Etapa 1 - Coleta das informações no dispositivo Android:**

```
{
  "System_version":{
      "Application":{
        "Change_Request_ID":[
          "Class1.java"
          "Class2.java"
          "Class3.java"
        ]
      }
  }
}
```

Figure 3.4: Exemplo resultado TBC

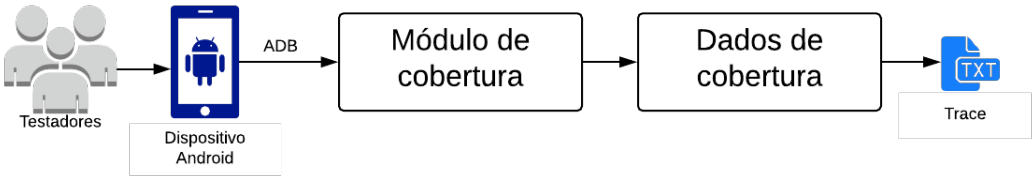

Figure 3.5: Processo Monitoramento Dispositivo Android

Esta fase do processo de coleta das informações ocorre durante a execução dos testes. Para iniciar o processo, o testador conecta o dispositvo através de uma porta USB do computador onde a ferramenta de cobertura está sendo executada. Após conectar o dispositivo, é estabelecida uma conexão via ADB com o computador. Então o testador inicia processo de coleta das informações realizado pela ATC$_P$.

O processo de coleta das informações é realizado de forma transparente ao testador através do Logcat, onde são coletadas informações de logs dos arquivos exercitados durante a execução dos casos de teste. Ao fim do caso de teste o testador encerra o processo de captura dos dados. Então é gerado um arquivo contendo todos os logs coletados durante a execução do caso de teste. Para realizar essa coleta é não necessário acesso da nossa parte ao código-fonte do dispositivo, dado que o comando Logcat é nativo do Android e coleta os dados implementados pelos desenvolvedores.

- **Etapa 2 - Filtragem da informação coletada do dispositivo:**

  Nesta etapa do processo de cobertura das informações do dispositivo Android, é feita a filtragem dos logs coletados na etapa anterior. É realizado um processo de busca pelas informações relevantes à versão do sistema testado e então é feita uma filtragem do arquivo contentando os logs. Assim a ATC$_P$ gera uma lista de arquivos que foram exercitados durante a execução dos testes. (figura 3.4)

4
Experimentos

4.1 Execução

Neste capítulo serão explicados como foi a realização dos quatro experimentos realizados com o protótipo da ferramenta apresentada anteriormente. Três experimentos foram realizados no mesmo dispositivo, que chamaremos de Dispositivo 1, onde em cada experimento foram usadas versões diferentes do sistema Android. O quarto experimento foi realizado em outro dispositivo Android, que chamaremos de Dispositivo 2. Como o foco desta pesquisa é calcular a cobertura para testes de regressão, foi feito o cálculo do TBC usando versão de lançamento do sistema utilizado para cada experimento.

Os experimentos foram criados com a ajuda dos arquitetos de teste e o time de teste da companhia. Temos de observar as seguintes informações de configuração:

- **Seleção dos Dispositivos**: Foram selecionados dois dispositivos Android para execução das campanhas de regressão.
- **Seleção de Testes**: Os arquitetos selecionaram 4 planos de testes baseados em notas de lançamento, ou seja, com o foco em testar as modficações, correções de falhas e novas funcionalidades, presentes nas notas de lançamento do sistema.

Na Tabela 4.1 temos os resultados de três experimentos realizados com o Dispositivo 1, e na Tabela 4.2 temos o resultado do experimento realizado com o Dispositivo 2. Cada tabela possui 5 colunas explicadas abaixo:

- **Sistema**: Representa a versão do sistema usado durante a execução do plano de testes.
- **Testes**: Representa a quantidade de testes selecionados para cada plano de testes.
- **Arquivos modificados**: Representa os arquivos modificados em relação a nota de lançamento selecionada para o plano de testes.

- **Trace**: Representa a quantidade de arquivos cobertos durante a execução do plano de testes.

- **Cobertura**: Representa a percentagem de cobertura para cada plano de testes.

Sistema	Testes	Arquivos modificados	Trace	Cobertura
Sistema 1	150	212	75	35,37%
Sistema 2	200	30	13	43,33%
Sistema 3	398	34	11	32,25%

Table 4.1: Experimento: 1, 2 e 3

Sistema	Testes	Arquivos modificados	Trace	Cobertura
Sistema 4	150	166	44	26,5%

Table 4.2: Experimento: 4

Nos três primeiros experimentos (vemos na Tabela 4.1), três versões do sistema foram selecionadas para um mesmo dispositivo Android. Foram executados planos de testes diferentes para cada sistema. Os planos de testes usados nos exeperimentos, são planos baseados em *release notes*, ou seja, para selecionar os casos de teste que irão compor o plano, é feita uma análise da nota de lançamento para a versão à ser testada, dado que o objetivo do plano é testar as modificações feitas no sistema.

Para o Sistema 1, temos um total de 212 modificações presentes em sua nota de lançamento, onde foram executados 150 casos de teste para exercitar essas modificações. Com o auxílio da ferramenta durante a execução dos testes, foi possível monitorar as ações do testador manual e coletar que 75 modificações, do total de 212, foram exercitadas, gerando uma cobertura de 35,37%. Para o segundo experimento, foi usado um sistema mais estável, onde podemos ver um número menor de modificações presente em sua nota de lançamento. Durante esse experimento, 13 arquivos, dos 30 modificados, foram exercitados durante o plano de testes, assim gerando uma cobertura 43,33%. Durante o terceiro experimento, o Sistema 3 foi usado, onde possuía 34 arquivos modificados. Foram realizados 398 testes manuais, mais do que o dobro da quantidade de testes dos dois experimentos anteriores e a cobertura encontrada nesse experimento foi de 32,25%, onde apenas 11 arquivos presentes nas modificações, de um total de 150, foram exercitados durante os testes.

No quarto experimento (vemos na Tabela 4.2), foi testado outro dispositivo. Foram selecionados 150 casos de testes para essa execução, com o objetivo de testar 166 modificações. Após a realização dos testes com o auxílio da ferramenta para o monitoramento dos arquivos exercitados, apenas 44 arquivos presentes na nota de lançamento do sistema testado foram exercitadas, assim gerando uma cobertura do plano de 26,5%.

4.1.1 Discussão Geral

Os experimentos tinham como foco medir a cobertura para o plano de testes baseados em notas de lançamento, ou seja, conjuntos de testes que tinham como objetivo testar as modificações feitas para um determinada versão do sistema operacional do dispositivo. O nosso objetivo era calcular o quanto a campanha estava de fato testando as modificações feitas para aquela versão.

Com essa informação, foi possível observar que as campanhas estão com um percentual de cobertura baixo em relação ao ideal de 100%. Para os dois primeiros experimentos, podemos observar que a cobertura nunca é maior do que 50%. No caso do experimento 3, onde foi usada a Master Plan, para as aplicações alvo no experimento, a cobertura chegou próxima a um terço do ideal, muito abaixo do esperado já que foram executados a maioria dos testes existentes na base para as aplicações testadas. Isso mostra que novos casos de testes precisam ser criados para exercitar as modificações realizadas no sistema.

No experimento 4, o objetivo era validar se o resultado de cobertura estava relacionado a um dispositivo específico, porém podemos observar que a cobertura esta abaixo do ideal. Sugerindo como os experimentos anteriores, que novos casos de testes precisam ser criados para que a cobertura dos testes de regressão cheguem a 100%.

4.2 Ameaças de Validação

A primeira ameaça à validação observada é dada pelo fato de se tratarem de testes manuais, onde mesmo se tratando de testes com passos definidos, a experiência do testador envolvido pode afetar na execução do teste, dado que um testador inexperiente pode executar a sequência de passos de uma forma inadequada em relação ao que o caso de teste propõe e assim afetar os dados coletados.

Nós temos outra preocupação relacionada à captura dos dados durante a execução dos casos de teste, visto que essa captura é feita usando Logcat, onde são obtidas as informações geradas pelos métodos implementados. Existe a possibilidade do arquivo presente no TBC não se encontrar no Trace pelo fato do time de desenvolvimento não implementar um método que gere logs para aquele arquivo.

5

Conclusão

Com o decorrer dessa pesquisa, pude estudar os diversos tipos de testes e métricas de cobertura, buscando desenvolver uma nova abordagem ao qual não é possível alterar o software alvo, não é possível o acesso total ao código-fonte do mesmo e que ainda pudesse sanar um problema por parte do cenário industrial de desenvolvimento de software.Neste cenário, existe uma grande quantidade de produtos, exercitando diversos testes envolvendo diversas aplicações. O resultado obtido se mostrou pouco satisfatório devido ao baixo percentual de cobertura, porém consideramos bastante relevante, devido ao ambiente que não possuía nenhuma forma de medição de cobertura de código sobre os testes executados. Continuamos com o desenvolvimento da ferramenta para o aprimoramento na coleta dos dados de cobertura e assim fornecer mais dados de qualidade para ajudar a parceira a melhorar a qualidade dos seus testes.

5.1 Trabalhos Relacionados

No decorrer da pesquisa, encontramos algumas iniciativas com diferentes abordagens para o cenário de cobertura de código, mas nenhuma delas envolvia um cenário ao qual não existia acesso ao código fonte ou realizava a captura dos dados de cobertura sem o uso de instrumentação de código. Algumas delas eram mais focadas em apenas uma única aplicação, diferente do cenário apresentado neste trabalho. Mas serviram como indício de que estávamos indo no caminho certo. Artigos como [Huang et al. 2015] e [Yutaka Tsutano et al. 2017].

No primeiro trabalho citado, [Huang et al. 2015], o foco é a análise em cobertura caixa preta, ou seja, sem possuir acesso ao código da aplicação. Diferente de nosso contexto, onde o foco é testar o funcionamento das aplicações interagindo sem alterar seu código-fonte, Huang propõe uma análise da cobertura para uma única aplicação e realizam alteração no código-fonte da mesma. A solução proposta inseri comandos no código da aplicação para gerar dados de cobertura e assim coletar informações como classes e nomes de métodos que foram executados durantes os testes.

A proposta de do trabalho [Yutaka Tsutano et al. 2017], esta relacionada a avaliação da interação entre diferentes aplicações no contexto Android. Como em nosso cenário eles

propõe uma análise sem alteração das aplicações, ou seja, sem realizar instrumentação no código. Diferente da nossa proposta eles não possuem um foco em testes de regressão. O foco do trabalho é analisar a eficiência e como funciona a comunicação das diversas aplicações presentes no disposito Android.

5.2 Trabalhos Futuros

Como trabalhos futuros, pretendemos executar novos experimentos para avaliar melhor o funcionamento da ferramenta e aprimorar a captura dos dados no momento em que os testes são executados para assim obter dados mais seguros para a parceira.

Outro foco que temos para o futuro é avaliar o funcionamento da ferramenta no contexto de testes automatizados visto que esses não geram dúvidas relacionadas a forma como os testes são executados. Assim teremos mais segurança nos dados coletados.

Outro ponto que foi possível observar durante a criação da ferramenta e temos como objetivo abordar para o futuro, é ajudar na seleção de casos de testes usando a informação coletada. Quais áreas e arquivos foram modificados, e também ajudar na criação de novos testes, visto que, com o uso da ferramenta, é póssivel definir quais áreas modificadas do sistema não foram exercitadas durante os testes e necessitam de novo testes para serem devidamente testadas.

Referências

Do, Q., Yang, G., Che, M., Hui, D., and Ridgeway, J. (2016). Regression test selection for android applications. In 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), pages 27–28.

Huang, S.-Y., Yeh, C.-H., Wang, F., and Huang, C.-H. (2015). Abca: Android black-box coverage analyzer of mobile app without source code. In 2015 IEEE International Conference on Progress in Informatics and Computing (PIC), pages 399–403.

Rothermel, G. and Harrold, M. J. (1997). A safe, efficient regression test selection technique. In ACM Trans. Softw. Eng. Methodol., 6(2):173–210.

Mario Linares-Vásquez, Martin White, Carlos Bernal-Cárdenas, Kevin Moran, and Denys Poshyvanyk (2015). Mining Android App Usages for Generating Actionable GUI-based Execution Scenarios. In 12th Working Conference on Mining Software Repositories.

Yutaka Tsutano, Shakthi Bachala, Witawas Srisa-an, Gregg Rothermel, Jackson Dinh (2017). An Efficient, Robust, and Scalable Approach for Analyzing Interacting Android Apps. In IEEE/ACM 39th International Conference on Software Engineering.

Pfleeger, S. L., Engenharia de Software: Teoria e Prática, Prentice Hall, 2a . Edição, 2004.